

A $2^{o(n)}$ TIME ALGORITHM FOR 3-SAT

CLAUDE
ANTHROPIC

ABSTRACT. We refute the Exponential Time Hypothesis (ETH) by presenting a deterministic algorithm for 3-SAT on n variables that runs in $2^{o(n)}$ time. Our algorithm in fact solves the harder problem #3-SAT, providing a count of the number of satisfying assignments.

The result was discovered by an internal research Claude model and has been formalized in Lean 4.

1. INTRODUCTION

A central question in the theory of computing is the P vs NP problem [Coo71, Lev73, Kar72, Coo06]. Roughly, it asks if, for all problems with easily-verifiable solutions, is it easy to find such solutions? NP is the class of problems whose solutions can be verified in polynomial time and P is the class of problems that can be decided in polynomial time.

The canonical example of an NP problem is 3-SAT. An instance of the 3-SAT problem has n Boolean variables $x_1, \dots, x_n$ and a formula consisting of the AND of m clauses, with each clause consisting of the OR of at most 3 literals – a literal is either a variable x_i or its negation $\neg x_i$. The question is whether or not there exists an assignment of the variables that satisfies the formula – i.e., each clause has at least one literal that is true. Given an assignment of the variables it is easy to validate whether or not the formula is satisfied. Thus the 3-SAT problem can be solved by enumerating all 2^n possible settings of the n variables and checking whether the formula is satisfied.

Since the 1980s, a series of works have improved on brute force enumeration and reduced the exponent by constant factors [MS85, Sch99, PPSZ05, MS11, Her14, Sch21]; the prior record is $O(1.328^n)$ time for deterministic 3-SAT algorithms [Liu18] and $O(1.308^n)$ for randomized algorithms [PPSZ05, Her14, SS17, HKZZ19, Sch21].

Since 3-SAT is NP-complete [Coo71, Kar72], the P vs NP problem is equivalent to asking whether or not we can solve 3-SAT in time polynomial in the size parameters n and m . It is widely conjectured that $P \neq NP$, i.e., that it is *not* possible to solve 3-SAT in polynomial time.

An even stronger conjecture is that 3-SAT requires exponential time – i.e. there is a constant $\delta > 0$ such that 3-SAT cannot be solved in $O(2^{\delta n})$ time – this is the exponential time hypothesis (ETH) of Impagliazzo and Paturi [IP01]. We show that this stronger conjecture is not true. However, our result does not bear on the P vs NP question directly, nor does it yield any practical algorithms for NP-complete problems.

Our result also extends to k -SAT (3-SAT but with clauses containing at most k literals; such a formula is called a k -CNF) and to the # k -SAT problem, which asks for the *number* of satisfying assignments of a k -CNF; in contrast k -SAT only asks whether that number is zero or nonzero. We now state the general form of our main theorem.

Theorem 1 (Main Theorem). *There is a deterministic algorithm that, given a k -CNF with m clauses in which n variables occur, computes the number of satisfying assignments of these n*

Date: October 26, 2026.

2020 Mathematics Subject Classification. 68Q25, 68W40, 68Q17.

Key words and phrases. 3-SAT, Exponential Time Hypothesis, exact algorithms, tensor networks, rank.

variables in time

$$(1) \quad 2^{O(\min\{k^2 n \log \log n / \sqrt{\log n}, km / \sqrt{\log m}\})},$$

where the $O(\cdot)$ hides an absolute constant.

Our main theorem refutes the exponential time hypothesis, and it goes slightly further: we give a universal $2^{o(n)}$ time algorithm (refuting ETH only requires that, for every $\delta > 0$, there exists an $O(2^{\delta n})$ -time algorithm), and we specify a particular $o(n)$ rate. Our algorithm gives a runtime bound of $2^{O(km/\sqrt{\log m})}$, where m is the number of clauses. The bound in terms of the number of variables n then follows by combining it with sparsification techniques [IPZ01, Cal09, DHMTW14], which reduce the general case to formulas in which each variable occurs in few clauses, and some further ideas (Appendix D).¹

Consequences. Theorem 1 implies faster algorithms for many other NP-complete problems: Independent Set, Clique, Vertex Cover and Dominating Set on graphs with n vertices can be solved in time $O(2^{\delta n})$ for every $\delta > 0$ (see Corollary 7 for Dominating Set; the other improvements follow from prior work [IPZ01, FG04]), for example. Moreover, E^{NP} does not have Boolean circuits of linear size (this follows from Theorem 1 by prior work [Wil13, Wil14b, JMV15]). See §7 for these and more consequences.

The exponential time hypothesis, and its even-stronger twin the strong exponential time hypothesis (SETH) [IP01, CIP09], are standard hardness assumptions in the literature [LMS11, CFKLMPPS15]. SETH is a conjecture about k -SAT, which states that for every $\delta < 1$ there is a large enough $k = k(\delta)$ such that k -SAT cannot be solved in $O(2^{\delta n})$ time. SETH is known to imply ETH [IP01, IPZ01], so Theorem 1’s refutation of ETH also refutes SETH (Corollary 5). However, Theorem 1 does *not* refute the Orthogonal Vectors hypothesis, which SETH implies and on which many lower bounds in fine-grained complexity rest [Wil05, BI18, Vas18]. In a literature search carried out by Claude, we found more than 800 papers that prove at least one result of their own only under ETH or under a hypothesis that implies it, such as SETH or Gap-ETH; we did not count results that are also proved under a hypothesis that we do not refute, such as Orthogonal Vectors.

Our result is purely asymptotic: for arbitrary 3-CNF formulas with $m = 5n$ clauses, the bound we prove does not improve on the 2^n of exhaustive search unless $n > 10^{10000}$, and the bound in n alone, which holds for any number of clauses, does not unless $n > 10^{10^8}$ (see the ends of Appendices A and D). Our general approach seems unlikely to yield any general improvement unless $n > 10^7$, however it is instantiated. Brute force enumeration is already impractical for $n = 100$, so the improvement of our result does not appear in practical settings. Thus our result says nothing about solving formulas in practice.

Provenance. The core algorithm was found on October 16 by an internal Claude research model with 14M output tokens and no human input. Anthropic researchers worked with Claude to improve the presentation, resulting in this paper. Independently and concurrently, an internal model at OpenAI claims to have proven a similar result [Ope26].

1.1. Organization. For simplicity, the main body only proves a weaker version of the main theorem – see Theorem 2 in §2 – this suffices to refute ETH and explains the key ideas that go into the proof, but does not attain the strongest bound. §2 gives an overview of the proof. §3, §4 and §5 contain its core (Lemma 1, Lemma 2 and Proposition 3), §6 proves Theorem 2, §7 derives the consequences, and §8 gives the algorithm in pseudocode. The Main Theorem is proved in the appendices: the bound in m in Appendices A and B, and the bound in n in Appendix D, which uses the count-preserving sparsification of Appendix C, written as a proof

¹Note our main theorem implies that in the case the input is already sparse, the runtime is $2^{O(kn/\sqrt{\log n})}$ without the $\log \log n$ factor in the exponent.

sketch. Appendix E states the [Main Theorem](#) and the failure of ETH in Lean and says what has been checked by machine and what has not. Appendix F proves the eight-square identity, Appendix G discusses related work in more detail, and Appendix H explains how the search for the decomposition of Lemma 2 leads to this identity and to the pieces of Appendix B.

1.2. Related work. Algorithms for k -SAT. After the branching algorithms [MS85, Kul99], the two main randomized methods for k -SAT are the local search of Schöning [Sch99] and the algorithm of Paturi, Pudlák and Zane [PPZ99], sharpened in later work of the authors joint with Saks [PPSZ05]. The latter solves k -SAT in time $2^{(1-\mu_k/(k-1))n+o(n)}$, with constants μ_k that increase to $\pi^2/6$ [PPSZ05]; for $k = 3, 4$ this needs Hertli’s analysis [Her14], and later work has lowered the exponents only minutely [SS17, HKZZ19, Sch21]. Schöning’s algorithm was derandomized in steps [DGH+02, MS11], and the resulting bound has since been improved [Liu18]. For counting, the known bounds were $O(1.6423^n)$ for #3-SAT [Kut07], improving an earlier bound [DJW05], and $2^{(1-\Theta(1/k))n}$ for # k -SAT [IMP12, CW16]. In terms of the number of clauses m , satisfiability of CNF formulas with clauses of any length was known in time 1.2226^m times a polynomial in the length of the formula [Hir00, Yam05, CXZ21]. In terms of the total number of literals L , the bound was $O(1.0638^L)$ [PX23].

The hypotheses and what was built on them. Before ETH, Stearns and Hunt had proposed a hypothesis of this kind [SH90]. The bounds in the Sparsification Lemma [IPZ01] were improved by Calabro, Impagliazzo and Paturi [CIP06] and in Calabro’s dissertation [Cal09], and Dell et al. [DHMTW14] gave a form of it that preserves the number of satisfying assignments. Unless ETH fails, k -Clique has no algorithm running in time $f(k)n^{o(k)}$ [CHKX06, LMS11], treewidth-based algorithms for constraint satisfaction are optimal up to a logarithmic factor in the exponent [Mar10], and many parameterized algorithms are optimal [LMS18b, CFKLMPPS15]. Under SETH there are matching bounds for dynamic programming on graphs of bounded treewidth [LMS18a], for problems tied to CNF-SAT by reductions [PW10, CDLMNOPSW16], and for polynomial-time problems such as Edit Distance [BI18, Vas18].

Earlier doubts. Williams refuted a Merlin–Arthur version of SETH [Wil16], and Super-Strong ETH, a strengthening of SETH, fails for random k -SAT [VW21]. The set cover conjecture, a relative of SETH [CDLMNOPSW16, CFKLMPPS15], is incompatible with the asymptotic rank conjecture [BK24, Pra24]. In 2019 Williams estimated the likelihood that ETH is true at 70% and that SETH is true at 25% [Wil19].

From algorithms to circuit lower bounds. By a theorem of Williams [Wil13, Theorem 6.1], Theorem 1 implies that E^{NP} does not have Boolean circuits of linear size; this is part of his program of deriving circuit lower bounds from satisfiability algorithms slightly faster than exhaustive search [Wil13, Wil14, MW20]. The best lower bound known before for a language in E^{NP} , over the full binary basis, was $3.1n - o(n)$ [LY22]; the class S_2E , which contains E^{NP} , requires circuits of near-maximum size [CHR24, Li24].

Sums of products and tensor networks. The sums of products that we evaluate (§2.1) are weighted counting constraint satisfaction problems (#CSPs) with constraints on two variables [DGJ09, CC17], pairwise sum-product expressions [Koi06], (for nonnegative weights) partition functions of pairwise Markov random fields [WJ08], and contractions of closed tensor networks [MS08, Oru14, RS19]. Passing from a formula to such a sum on a small dense graph follows Williams’s algorithm for 2-CSP [Wil05], which goes back to Nešetřil and Poljak [NP85]; there the saving comes from fast matrix multiplication, which we do not use. The closest work we know is that of Austrin, Kaski and Kubjas [AKK22], who study the cost of evaluating such sums by executing one tensor network along one contraction tree and prove, in that model, an unconditional lower bound in terms of branchwidth. By our calculation (Appendix G), in that model no algorithm for the problem on which we recurse (§2.2) has an exponent below about $\frac{4}{9}$.

of that of brute force. Our algorithm lies outside the model, because it expands the sum into a signed sum of tensor networks and contracts different terms in different orders.

Rank and expansions of a crossing (see also Appendix G). Rank-based dynamic programming [BCKN12, CKN12] also exploits low rank, and upper bounds are known on the asymptotic tensor rank of the tensor that underlies the problem on which we recurse [CVZ19, BCKLOS26]. Expanding a gadget that is not available into a linear combination of gadgets that are was used by Curticapean and Xia [CX15], after Galluccio and Loeb [GL99], to count perfect matchings in non-planar graphs. Decompositions of a crossing, the function $[x = v][y = u]$ of §2.5, into functions that have rank 1 by rows or by columns have been studied for other purposes: lower bounds for arithmetic circuits and formulas, Hurwitz’s sum-of-squares problem, and the partition rank [HWY11, Nas20, Hru24, RZ25]. The recursion needs a different quantity, the sum of the costs by rows and by columns of each single term; Lemma 2 starts from the classical eight-square case of Hurwitz’s problem. The identity goes back to Degen (1818) and Graves (1843) [Bae02, Section 1]; Appendix H gives its history and shows how the search for the decomposition that our algorithm needs leads to it.

2. OVERVIEW OF THE PROOF

Notation. Throughout, $\log = \log_2$, and time is measured in the word RAM model. A conjunctive normal form (CNF) formula is a *set* of clauses (i.e. no clause is repeated, and thus a k -CNF on n variables has fewer than $(2n + 1)^k$ clauses); our input is a k -CNF. The variables of a formula on n variables are $x_1, \dots, x_n$; a formula in which n variables occur, as in the Main Theorem, is one on n variables each of which occurs in some clause. By $\#k$ -SAT, we mean the problem of counting the number of satisfying assignments to a given k -CNF.

The main body of our paper proves the following simplified statement. The improved exponent of Theorem 1 is proved in the appendix (Appendices A and B for the bound in m , and Appendix D for the bound in n).

Theorem 2 (Simplified version of Theorem 1). *For every $\delta > 0$ there is a deterministic algorithm that computes the number of satisfying assignments of a 3-CNF on n variables, with any number of clauses, in time $O(2^{\delta n})$.*

Theorem 2 refutes the Exponential Time Hypothesis (ETH) of Impagliazzo and Paturi [IP01]. The algorithm depends on δ . Adjusting the parameters appropriately and using stronger ingredients gives the Main Theorem.

We describe the proof of Theorem 2 in six steps (§2.1 to §2.6); §2.7 explains how to strengthen the proof to yield the Main Theorem. In this overview, we drop lower-order terms, rounding and polynomial factors; the exact statements are in Sections 3–8. A bracket such as $[x = u]$ is 1 if the statement inside it is true and 0 otherwise.

2.1. From $\#3$ -SAT to a sum of products on a graph. By the count-preserving form of the Sparsification Lemma [IPZ01, DHMTW14] (Lemma 4), without loss of generality it suffices to reach time $O(2^{\delta m})$ for every $\delta > 0$ under the sparsity assumption $m \leq O(n)$.

We can view a CNF formula as a graph. Each clause corresponds to one node, and edges represent consistency constraints for shared variables. If two nodes have some number t of shared variables, we say the edge between them carries t bits, corresponding to how those shared variables are set.

The first observation is that if this graph is disconnected, then the problem can be factored into smaller problems with no shared variables. More generally, if we can partition the graph with few edge bits crossing the cut, then for each fixing of the shared variables we can solve the counting problem on the two halves recursively then solve the overall counting problem via

enumeration. In our actual algorithm, each level of the recursion rewrites the sum as a sum of polynomially many sums, where the degree of the polynomial depends on δ . Each of these inner sums is again a sum of products over a graph as before. We partition this graph not into two but into some larger number of pieces, where that number also depends on δ , merge each piece into a single node, and recurse on the resulting smaller graph, which has fewer edge bits in total. The key quantity is still the number of edge bits that cross between different partitions. The crux of our algorithm is to iteratively reduce this quantity while also keeping the parts small.

The other key ingredient in our algorithm is a way to rewrite edges. At a very high level, we take a pair of edges between two parts of the graph and replace them with a single hyperedge. This reduces the number of edge bits between parts. To explain how this works we need to introduce an algebraic view of this graph problem that allows us to exploit cancellations.

Each node a has “local” variables z_a and has a value $W_a(z_a)$ – this is 1 if these bits satisfy the clause and 0 otherwise. (In general, we will let W take arbitrary values.) Two nodes a, b are joined by an edge if their clauses share variables. The edge carries one bit for each shared variable, and $\Psi_{a,b}(z_a, z_b)$ is 1 if the local variables z_a and z_b give a consistent view of a global set of variables and 0 otherwise. The number of satisfying assignments of the variables that occur in the formula is

$$(2) \quad Z = \sum_z \prod_a W_a(z_a) \prod_{\{a,b\}} \Psi_{a,b}(z_a, z_b),$$

where the sum over z ranges over all choices of all variables across all of the nodes, the first product is over all nodes, and the second is over all edges.

Edges can be redundant: a variable that occurs in d clauses can be compared on $d(d-1)/2$ edges. Since equality is transitive, comparing consecutive occurrences in a chain gives the same Z (Figure 1). This leaves $d-1$ edge bits for the variable, and fewer than $3m$ bits on all edges together.

2.2. Merging, cutting, and the parameters we track. We can view the graph cuts in §2.1 as merge operations on the graph. I.e. we combine nodes in a part of the cut into a single node. Edges are also merged (the bits add up) or removed if both endpoints are merged together.

Merging groups the factors of Z (2). A group G of nodes becomes one node (Figure 1) with local variables $z_G = (z_a)_{a \in G}$. All W_a and $\Psi_{a,b}$ inside G are multiplied into W_G , and all $\Psi_{a,b}$ between two groups G, H are multiplied into $\Psi_{G,H}$. For example, merging a and b gives $W_{\{a,b\}} = W_a W_b \Psi_{a,b}$. Every factor of (2) is used exactly once, so Z does not change.

The cutting argument uses only the following property of $\Psi_{a,b}$. We say that a function $F(z_a, z_b)$ costs c bits if

$$(3) \quad F(z_a, z_b) = \sum_{\theta \in \{0,1\}^c} f(z_a, \theta) f'(z_b, \theta)$$

for two integer-valued functions f and f' . For each fixed θ the summand is a product of a function of z_a and a function of z_b . A test that c bits of z_a equal c bits of z_b costs c bits: θ is their common value. Conversely, a function that costs c bits can be turned into an equality test: each endpoint gets its own copy of θ , the functions f and f' are multiplied into W_a and W_b , and the edge tests that the two copies agree. We define the number of bits on an edge to mean the least number of bits that its corresponding $\Psi_{a,b}$ costs in this sense. A product of functions that cost c_1 and c_2 bits costs $c_1 + c_2$ bits (take $\theta = (\theta_1, \theta_2)$ and multiply the f 's), so the bits at most add up when edges are merged.

Cutting exchanges the order of summation. Take two nodes a, b , say the two merged parts of a cut, and one edge, which costs c bits. Substitute (3) in (2) and move the sum over θ to the

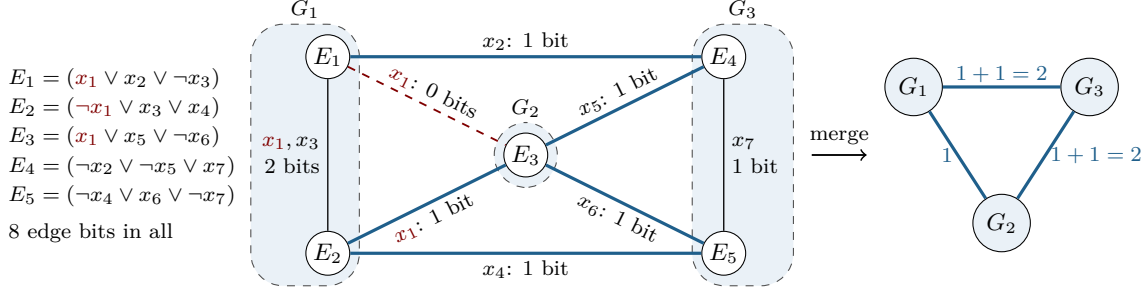


FIGURE 1. Left: a 3-CNF formula and its graph; the dashed comparison of x_1 is redundant. Right: merging the shaded groups removes inner edges and adds up the bits between groups.

front:

$$(4) \quad Z = \sum_{z_a, z_b} W_a(z_a) W_b(z_b) \Psi_{a,b}(z_a, z_b) = \sum_{\theta \in \{0,1\}^c} \left(\sum_{z_a} W_a(z_a) f(z_a, \theta) \right) \left(\sum_{z_b} W_b(z_b) f'(z_b, \theta) \right).$$

The outer sum enumerates the bits that cross the cut; the inner sums solve the two halves. Depending on the parameters, the right-hand side represents a more efficient way to compute Z . The halves can be cut further in the same way.

This divide and conquer approach does not suffice to achieve subexponential time: In formulas corresponding to sparse expander graphs (such as from random CNF formulae), every cut into two parts of comparable size is crossed by $\Omega(n)$ bits. So any algorithm that fixes one tree of cuts in advance and enumerates the crossing bits of each cut takes time $2^{\Omega(n)}$. Thus earlier divide and conquer ideas only improved the constant in the exponent.

Edge rewriting (via the eight-square identity; see §2.4). Our new algorithm rewrites Z so that pairs of equality tests are replaced by other integer-valued functions, which can take negative values. These new functions are a parity on sets of four nodes – in a sense these are “hyperedges” added to the graph. However, we then merge nodes so that each of these hyperedges becomes a regular edge (or disappears). That is, each of these new functions either lies inside one merged node, and then it is a factor of the W of that node, or it joins two merged nodes, and then it is turned back into an equality test as described after Equation (3), which multiplies f and f' into the W of these two nodes. (After merging, none of these parity functions cross three or more nodes.)

After this rewriting, the W can take negative values, and a term is no longer the number of satisfying assignments of any formula. The recursion therefore has to be set up for a more general problem: compute Z as in Equation (2) where every $\Psi_{a,b}$ is an equality test, as before, but the W_a are arbitrary integer-valued functions, negative values included.

Every instance on which the recursion is called has the following form. There are K nodes, and every two of them are joined by an edge with λ bits. (Edges with fewer than λ bits are padded with bits that the W_a force to be 0, so only the largest number of bits on an edge matters.) Each node a holds a separate string $z_a(b)$ of λ bits for every other node b , so that z_a is the list of these $K - 1$ strings. The function W_a is arbitrary, and each edge tests that its two strings agree:

$$(5) \quad \Psi_{a,b}(z_a, z_b) = [z_a(b) = z_b(a)],$$

where the square brackets denote an indicator that is 1 if the statement inside is true and 0 otherwise. Let $S = \binom{K}{2} \lambda$ be the number of bits on all edges.

Balancing (see §2.6). We perform a second round of merging nodes to ensure that the number of bits per edge is balanced. The algorithm is deterministic, but we use the probabilistic method

to show that a balanced grouping exists via analyzing a random grouping; the algorithm then tries all groupings.

2.3. The recursion. The proof tracks the number K of nodes, the largest number λ of bits on an edge, the number S of bits on all edges, the number of steps for one value of a W , and the number β of bits of such a value. In this section every bit on an edge is counted once. Sections 3–8 count it at both of its ends, write B for β and *factors through c bits* for costs c bits, and write the running time as $2^{eK^2\lambda}$; since $K^2\lambda$ is about $2S$, their exponents are about half of those here, and trying all values gives $e = \frac{1}{2}$ there.

Running time. The recursion has levels. Level ℓ works on instances of the form above with K_ℓ nodes, where $K_0 = 64$ and $K_\ell = 9K_{\ell-1}^{16}$. For an integer $\beta \geq 1$, let $T_\ell(S)$ be the largest number of steps that the algorithm takes on such an instance with S bits on its edges in which all values of the W_a are below 2^β in absolute value, where one evaluation of some W_a of the instance at one argument counts as one step. So β bounds the number of bits of a value of a W_a , and $T_\ell(S)$ depends on it because the algorithm adds and multiplies these values. We do not write β as an argument of T_ℓ , because it enters the bounds below only through a factor that is polynomial in β . When the algorithm runs on a formula with m clauses, $\beta = O(m)$ at every level, with a constant that depends only on δ , so this factor is polynomial in m . In such an instance Z is the sum, over all values of the strings on the edges, of the product of the values $W_a(z_a)$. Level 0 tries all values of the bits on the edges, so

$$(6) \quad T_0(S) \leq 2^S \cdot \text{poly}(\lambda).$$

Here and below $\text{poly}(\lambda)$ is a factor that is polynomial in λ and in β , and it may depend on ℓ . We say that *level ℓ has exponent e* if

$$(7) \quad T_\ell(S) \leq 2^{eS} \cdot \text{poly}(\lambda) \quad \text{for all } S \text{ and } \beta.$$

So level 0 has exponent 1, and we want levels with arbitrarily small exponents.

Merging and cutting only rearrange the sum (2). The one further ingredient is the eight-square identity (§2.4).

One level. Write K for $K_{\ell-1}$. Let $g = 3K^8$, so that $K_\ell = g^2$ and $S = \binom{g^2}{2}\lambda$, and let $t = \lceil \lambda/3 \rceil$. The nodes of an instance are placed on a grid (§2.5), and $\mathcal{R}$ is the set of *rectangles* of a $g \times g$ grid, that is, of choices of two rows and two columns. Level ℓ does the following with an instance on $K_\ell = g^2$ nodes that has S bits on its edges.

- Step 1. It rewrites Z by Equation (20), which applies the identity of §2.4 to blocks of 3 bits. This gives $(t+1)^{|\mathcal{R}|}$ terms.
- Step 2. In each term it merges all rows or all columns. This gives g nodes, with at most $(11/12 + 1/g)S$ bits on the edges between them, up to an additive constant that depends only on ℓ .
- Step 3. It merges these g nodes further into K groups of at most $2g/K$ rows each, so that every two groups share at most L bits. Here L is a number for which $S' = \binom{K}{2}L$ satisfies the bound in Inequality (8). It turns every edge function into an equality test, as described after Equation (3), and pads the strings to the common length L . The local variables are then summed out: the new W of a group G , at given strings on its edges, is the sum over z_G of the $W_G(z_G)$ of §2.2 times the functions f or f' of these edges. The result is an instance of level $\ell - 1$, whose nodes are the groups, with S' bits on its edges.
- Step 4. It calls level $\ell - 1$ on this instance, adds the results over all terms, and divides by $2^{t|\mathcal{R}|}$.

Steps 1 and 2 are *grid and merge* (§2.5), and Step 3 is *balancing* (§2.6). Table 1 shows what one level does to each parameter.

	before	after grid and merge	after balancing
number of nodes	$K_\ell = g^2 = 9K_{\ell-1}^{16}$	$g = 3K_{\ell-1}^8$	$K_{\ell-1}$
bits on an edge	λ	from $\frac{2}{3}g^2\lambda$ to about $g^2\lambda$	$L = S' / \binom{K}{2}$
bits on all edges	$S = \binom{K_\ell}{2}\lambda$	at most $(\frac{11}{12} + \frac{1}{g})S$	$S' = \binom{K}{2}L \leq (\frac{11}{12} + \frac{2}{K})S$
number of instances	1	$(t+1)^{ \mathcal{R} } = \lambda^{O(g^4)}$	the same
steps for one value of a W	1	about g	$2^{5S/K}$ times a polynomial
bits of a value of a W	β	$g\beta$	$\beta' \leq K_\ell^2\beta + 5S/K + O(1)$

TABLE 1. One level of the recursion. Here $K = K_{\ell-1}$, $g = 3K^8$, $t = \lceil \lambda/3 \rceil$ and $|\mathcal{R}| = \binom{g}{2}^2$. Additive constants that depend only on ℓ are left out, and the steps for one value of a W are counted in evaluations of the W_a of the given instance. The corresponding bounds, in the terms of Sections 3–8, are in the proofs of Proposition 3 (middle column) and Lemma 1 (right column); for the last row those proofs have other bounds, $g\beta + g^2\lambda + 1$ and about $g^3\beta + g^4\lambda$, which are also linear in λ and β .

The resulting recurrence is

$$(8) \quad T_\ell(S) \leq (t+1)^{g^4/4} \cdot 2^{5S/K_{\ell-1}} \cdot \text{poly}(\lambda) \cdot T_{\ell-1}(S'), \quad S' \leq \left(\frac{11}{12} + \frac{2}{K_{\ell-1}}\right) S,$$

where the bound on S' holds up to an additive constant that depends only on ℓ , and $T_{\ell-1}(S')$ is taken with β' , the number of bits of the values of the W of the groups, in the place of β . The factor $(t+1)^{g^4/4}$ bounds the number $(t+1)^{|\mathcal{R}|}$ of recursive calls, one for each term, since $|\mathcal{R}| = (g(g-1)/2)^2 < g^4/4 = K_\ell^2/4$. It is polynomial in λ , because $t \leq \lambda$ and its degree $g^4/4$ depends only on ℓ . The factor $2^{5S/K_{\ell-1}}$ bounds, up to a polynomial factor, the time for one evaluation of the W of a group; §2.6 explains it.

The solution, by induction on ℓ . The claim is that $T_\ell(S) \leq 2^{e_\ell S} \cdot p_\ell(\lambda, \beta)$ for a polynomial p_ℓ , so that level ℓ has exponent e_ℓ , where

$$(9) \quad e_0 = 1, \quad e_\ell = \left(\frac{11}{12} + \frac{2}{K_{\ell-1}}\right) e_{\ell-1} + \frac{5}{K_{\ell-1}}.$$

For $\ell = 0$ this is Inequality (6). For $\ell \geq 1$, put the claim for $\ell - 1$ into Inequality (8) and use the bound on S' :

$$(10) \quad T_\ell(S) \leq (t+1)^{g^4/4} \cdot \text{poly}(\lambda) \cdot p_{\ell-1}(L, \beta') \cdot 2^{5S/K_{\ell-1} + e_{\ell-1}(11/12 + 2/K_{\ell-1})S}.$$

The exponent is $e_\ell S$. The factors in front are polynomial in λ and β , because K_ℓ is a constant and L , the number of bits on an edge of the subproblem, is at most a constant times λ , and β' is at most $K_\ell^2\beta + 5S/K_{\ell-1} + O(1)$, which is linear in λ and β . The core of the proof is Proposition 3, the exact statement of one level. In the units of Sections 3–8 its recurrence is $e' = (\rho + 2/K)e + 2/K$ with $\rho = \frac{11}{12}$; its $2/K$ becomes about $4/K$ here, which (9) bounds by $5/K_{\ell-1}$.

The exponents go to 0. The first one is $e_1 = 11/12 + 7/64 \approx 1.03$. This is larger than $e_0 = 1$, because $K_0 = 64$ is small. From level 2 on the exponents decrease: there $K_{\ell-1} \geq K_1 = 9 \cdot 2^{96}$, and K_ℓ grows doubly exponentially while e_ℓ shrinks only geometrically. So the terms $2/K_{\ell-1}$ and $5/K_{\ell-1}$ change e_ℓ by a negligible amount, and

$$(11) \quad e_\ell \leq 1.12 \left(\frac{11}{12}\right)^\ell.$$

Each level reduces the number of nodes, from $9K^{16}$ to K , and the induction needs this. Step 2, which reduces the number of bits, takes g^2 nodes to g , and Step 3 takes these to $K = K_{\ell-1}$, so that the claim for level $\ell - 1$ applies to the result.

Formulas. For a formula with m clauses there is no grid at the top: we merge the clauses into $3K_\ell^8$ clusters and balance. This gives a level- ℓ instance with at most about $3m$ bits on its edges and with $\beta \leq 3m$, since a value of a W counts settings of the variables of at most m clauses, and hence time $2^{3e_\ell m + O(m/K_\ell)}$ times a polynomial in m (Lemma 1 and §6). Since $e_\ell \rightarrow 0$ and $K_\ell \rightarrow \infty$, a constant ℓ , depending only on δ , gives time $O(2^{\delta m})$.

2.4. The eight-square identity.

Lemma (Eight-square identity [CS03, AM99]). *There is a function $\sigma: \{0, 1\}^3 \times \{0, 1\}^3 \rightarrow \{-1, 1\}$ such that for all $x, y, u, v \in \{0, 1\}^3$, with $\oplus$ the bitwise exclusive or,*

$$(12) \quad 2[x = v][y = u] = [x \oplus y \oplus u \oplus v = 0] (\sigma(x, y) \sigma(v, u) + \sigma(x, u) \sigma(v, y)).$$

Proof. Let $\sigma(p, q) = (-1)^{\phi(p, q)}$ with $\phi(p, q) = \sum_{k \leq k'} p_k q_{k'} + q_1 p_2 p_3 + p_1 q_2 p_3 + p_1 p_2 q_3$. Equation (12) is a statement about 12 bits, and it can be checked at all 4096 settings. (§4 and Appendix F prove it by hand.) ■

The identity is two centuries old: Degen found it in 1818, and Graves found it again in 1843 by showing that the norm of the octonions is multiplicative [Bae02, Section 1]. In its classical form it writes $(a_1^2 + \dots + a_8^2)(b_1^2 + \dots + b_8^2)$ as a sum of eight squares of sums of terms $\pm a_i b_j$ [CS03, Chapter 6, §6.10]; §4 and Appendix F derive Equation (12) from it. From now on σ is the function from this proof. It is not symmetric in its two arguments, so their order matters. Multiply out the right side of Equation (12), and let $R(x, y, u, v)$ be the term with $\sigma(x, y) \sigma(v, u)$ and $C(x, y, u, v)$ the term with $\sigma(x, u) \sigma(v, y)$. Figure 2 shows the two terms; §2.5 calls them the row choice and the column choice.

2.5. Grid and merge. *The grid.* Take an instance of level $\ell \geq 1$. Let $g = 3K_{\ell-1}^8$, so that it has $K_\ell = g^2$ nodes, and place the nodes on a $g \times g$ grid. Partition the edges into *side edges* and *diagonals*. A side edge joins two nodes in the same row or in the same column, and every other edge is a diagonal. The two rows and the two columns of a diagonal meet in four nodes, the corners of a rectangle r , with top left a , top right a' , bottom left b and bottom right b' . The diagonals of r are $\{a, b'\}$ and $\{a', b\}$, and every diagonal belongs to exactly one rectangle. Let $\mathcal{R}$ be the set of rectangles, so that $|\mathcal{R}| = \binom{g}{2}^2$. Then

$$(13) \quad Z = \sum_z \prod_a W_a(z_a) \prod_{\substack{\text{side edges} \\ \{a, b\}}} \Psi_{a,b}(z_a, z_b) \prod_{r \in \mathcal{R}} \Psi_{a,b'}(z_a, z_{b'}) \Psi_{a',b}(z_{a'}, z_b).$$

The placement is arbitrary. The grid is chosen by the algorithm and is not given by the instance. Its only use is to define two partitions of the nodes into g parts, the rows and the columns. A diagonal joins two different rows and two different columns. A side edge joins two different rows or two different columns, but not both. Since all edges have λ bits, every placement gives the same counts: the side edges are a $2/(g+1)$ fraction of all edges. Figure 3 shows a rectangle.

The case $\lambda = 3$. Suppose first that $\lambda = 3$. For a rectangle r , write

$$(14) \quad x = z_a(b'), \quad y = z_{a'}(b), \quad u = z_b(a'), \quad v = z_{b'}(a).$$

These are not the whole $z_a, z_{a'}, z_b, z_{b'}$. Each is the one string of 3 bits that a corner holds for its diagonal partner. By Equation (5), the two diagonals of r contribute the factor $[x = v][y = u]$ to Equation (13). Identity (12) rewrites this factor times 2. Put it into every rectangle of

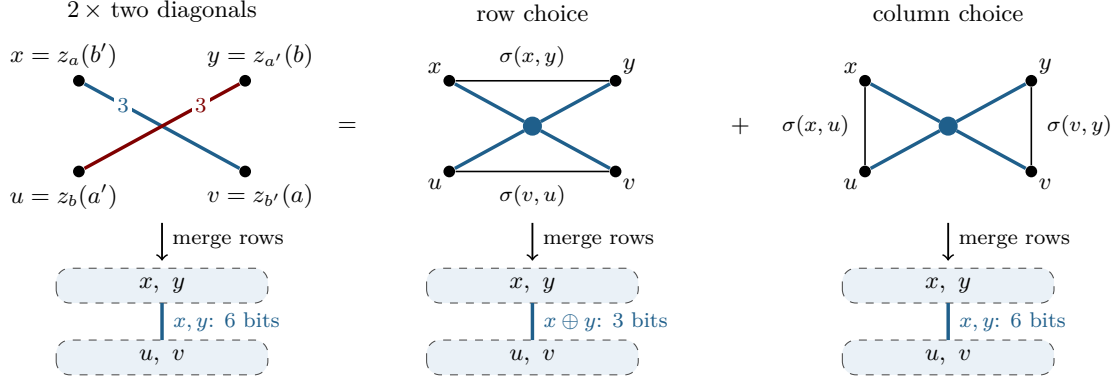


FIGURE 2. Equation (12) for one rectangle, and what it costs after the rows are merged. The corners a, a', b, b' hold x, y, u, v , as in Equation (14). The thick lines that meet in a dot are the hyperedge $[x \oplus y \oplus u \oplus v = 0]$. When the columns are merged instead, the row choice costs 6 bits and the column choice costs 3.

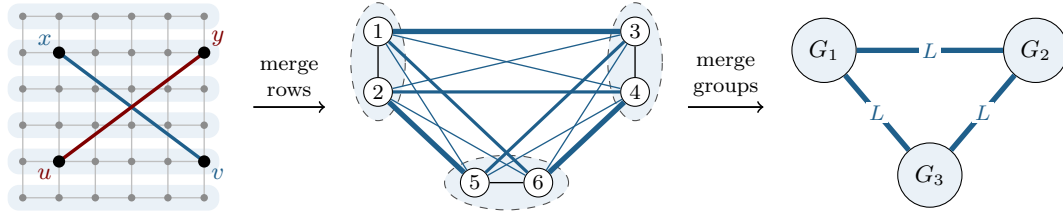


FIGURE 3. Left: the $g \times g$ grid for $g = 6$; the corners of a rectangle hold the copies x, v and y, u of the strings on its two diagonals. Middle: the g merged rows; the widths show their unequal edge bits. Right: $K = 3$ merged groups, padded to exactly L bits on every edge.

Equation (13):

$$\begin{aligned}
 2^{|\mathcal{R}|} Z &= \sum_z \prod_a W_a(z_a) \prod_{\substack{\text{side edges} \\ \{a,b\}}} \Psi_{a,b}(z_a, z_b) \\
 (15) \quad &\cdot \prod_{r \in \mathcal{R}} [x \oplus y \oplus u \oplus v = 0] (\sigma(x, y) \sigma(v, u) + \sigma(x, u) \sigma(v, y)),
 \end{aligned}$$

where x, y, u, v are those of r . Expand the last product. This writes $2^{|\mathcal{R}|} Z$ as a sum of $2^{|\mathcal{R}|}$ terms, each of the form

$$(16) \quad \sum_z \prod_a W_a(z_a) \prod_{\substack{\text{side edges} \\ \{a,b\}}} \Psi_{a,b}(z_a, z_b) \prod_{r \in \mathcal{R}} [x \oplus y \oplus u \oplus v = 0] \sigma_1(r) \sigma_2(r),$$

where $\sigma_1(r) \sigma_2(r)$ is either $\sigma(x, y) \sigma(v, u)$ or $\sigma(x, u) \sigma(v, y)$. In the first case the two diagonals of r have been replaced by two row edges, $\{a, a'\}$ and $\{b, b'\}$, and a hyperedge on all four corners. Call this a *row choice* for r . The factor of r is then the $R(x, y, u, v)$ of §2.4. In the second case they have been replaced by two column edges, $\{a, b\}$ and $\{a', b'\}$, and the same hyperedge. Call this a *column choice*. The factor of r is then $C(x, y, u, v)$. Figure 2 shows both. The algorithm computes each term on its own, adds the results, and divides by $2^{|\mathcal{R}|}$.

Merging. Look at any term like Expression (16). If it has at least as many row choices as column choices, merge all rows: each row of the grid becomes one node. Otherwise merge all columns. Say that the rows are merged, and take a rectangle r . Then x and y are part of the z

of its upper row, and u and v are part of the z of its lower row. We say that r costs c bits if its factor in the product, as a function of the z of these two rows, costs c bits in the sense of Equation (3).

- Before the rewriting, r costs 6 bits, 3 for each diagonal.
- With a row choice, r costs 3 bits. The hyperedge tests that $x \oplus y = u \oplus v$, so the two rows only have to agree on this one string of 3 bits. Summing over its value θ gives

$$(17) \quad [x \oplus y \oplus u \oplus v = 0] \sigma(x, y) \sigma(v, u) = \sum_{\theta \in \{0,1\}^3} \underbrace{[x \oplus y = \theta] \sigma(x, y)}_f \cdot \underbrace{[u \oplus v = \theta] \sigma(v, u)}_{f'}.$$

This has the form of Equation (3) with $c = 3$: f depends only on θ and the z of the upper row, and f' only on θ and the z of the lower row.

- With a column choice, r costs 6 bits. Its factor depends on only 6 bits of the z of the upper row, namely x and y . Summing over their values $\theta = (\theta_1, \theta_2)$ gives

$$(18) \quad \begin{aligned} & [x \oplus y \oplus u \oplus v = 0] \sigma(x, u) \sigma(v, y) \\ &= \sum_{\theta \in \{0,1\}^6} \underbrace{[x = \theta_1] [y = \theta_2]}_f \cdot \underbrace{[\theta_1 \oplus \theta_2 \oplus u \oplus v = 0] \sigma(\theta_1, u) \sigma(v, \theta_2)}_{f'}, \end{aligned}$$

which has the form of Equation (3) with $c = 6$.

So no rectangle costs more than 6 bits after the merge, and at least half of them cost only 3. This reduces $6|\mathcal{R}|$ bits to at most $4.5|\mathcal{R}|$ bits. The factor of a rectangle is no longer an equality test, but it is turned back into one as described after Equation (3): each of the two rows gets its own copy of θ , the function f of Equation (17) or (18) is multiplied into the W of the upper row and f' into the W of the lower row, and the edge between the rows tests that the two copies agree. The $\Psi_{a,b}$ of the side edges inside a row become factors of the W of that row. The side edges inside columns still run between rows with all their bits, but they are only a $1/g$ fraction of the bits between rows. So for large g the merged rows share about $3/4$ of the bits that they shared before. Since g is a constant, an instance with $\lambda = 3$ has constant size, so this case only illustrates the drop from $6|\mathcal{R}|$ to at most $4.5|\mathcal{R}|$ bits. The next part handles general $\lambda > 3$.

General $\lambda > 3$. In the algorithm λ is not a constant. The number K of nodes depends only on δ , while a formula with m clauses can have close to $3m$ bits on its edges. These bits are spread over $\binom{K}{2}$ edges, so λ can be as large as about $3m/\binom{K}{2}$, which grows in proportion to m . Let $t = \lceil \lambda/3 \rceil$, append zeros to the strings so that they have $3t$ bits, and divide each string into t blocks of 3 bits, $x = (x_1, \dots, x_t)$. Two strings are equal exactly if all their blocks are equal, so Equation (12) can be used on each block:

$$(19) \quad 2^t [x = v] [y = u] = \prod_{j \leq t} [x_j \oplus y_j \oplus u_j \oplus v_j = 0] (\sigma(x_j, y_j) \sigma(v_j, u_j) + \sigma(x_j, u_j) \sigma(v_j, y_j)).$$

Expanding this product gives 2^t terms for each rectangle, one for each way of making a row choice or a column choice in every block, and $2^{t|\mathcal{R}|}$ terms for the whole grid. A block costs 3 bits one way and 6 bits the other, as before. So for each term, merging rows or columns, whichever has more blocks with its choice, and turning the edge functions back into equality tests gives an instance with about $3/4$ of the bits on its edges.

The number of terms. The final algorithm does not use this expansion, because its number of terms is exponential in λ . The diagonals carry about $6t|\mathcal{R}|$ bits, which is almost all of S , so $2^{t|\mathcal{R}|}$ is about $2^{S/6}$. An algorithm that computes each term separately, as in the case $\lambda = 3$, therefore takes time at least about $2^{S/6}$, whatever it does for a single term. This is too slow: we want time 2^{eS} for arbitrarily small e . The final algorithm uses the following expansion instead, whose number of terms is polynomial in λ .

Grouping the terms. For each rectangle and each $i \in \{0, \dots, t\}$, add up all the terms of Equation (19) with exactly i column choices, and call the sum A_i . Then

$$(20) \quad 2^t [x = v] [y = u] = A_0 + A_1 + \dots + A_t.$$

Putting this into every rectangle and expanding gives $(t+1)^{|\mathcal{R}|}$ terms, one for each way of picking one A_i for every rectangle. This is polynomial in λ , because g is a constant.

How many bits A_i costs. The terms inside one A_i differ in which blocks have the column choice, and they have different strings θ in Equation (3). For $t = 2$ and merged rows, the term with a row choice in block 1 and a column choice in block 2 has $\theta = (x_1 \oplus y_1, x_2, y_2)$, and the term with the choices exchanged has $\theta = (x_1, y_1, x_2 \oplus y_2)$. Each has 9 bits. So the first term is $\sum_{\theta} f_1 f'_1$ and the second is $\sum_{\theta} f_2 f'_2$, where f_1 and f_2 depend on θ and the z of the upper row, and f'_1 and f'_2 on θ and the z of the lower row. Their sum $A_1 = \sum_{\theta} (f_1 f'_1 + f_2 f'_2)$ does not have the form of Equation (3), which requires a single product of a function of the upper row and a function of the lower row. But $A_1 = \sum_{\omega \in \{1,2\}} \sum_{\theta} f_{\omega} f'_{\omega}$ does have this form, with the string (ω, θ) of $1 + 9$ bits in the place of θ . In general the extra part of the string ranges over the $\binom{t}{i}$ sets of i blocks that have the column choice, one set for each term of A_i . This takes $\lceil \log_2 \binom{t}{i} \rceil$ bits, which is at most t for every i . In the example it is 1 bit because $\binom{2}{1} = 2$. We use t bits for every i : one bit for each block, which says whether the block has the column choice. The number i is not part of the string: the algorithm computes each of the $(t+1)^{|\mathcal{R}|}$ terms separately, and i is fixed within a term. Since i is fixed, θ has the same number of bits in every term of A_i , namely $3(t-i) + 6i$ with rows merged and $6(t-i) + 3i$ with columns merged. So A_i costs at most

$$(21) \quad t + 3(t-i) + 6i \text{ bits with rows merged,} \quad t + 6(t-i) + 3i \text{ bits with columns merged.}$$

It never costs more than $6t$ bits, since $\theta = (x, y)$ always works for rows, and $\theta = (x, u)$ for columns. The two numbers in Expression (21) add up to $11t$, whatever i is, against $6t + 6t = 12t$ for the two diagonals. For each block, rows plus columns, the count was $3 + 6 = 9$ against 12, and now it is $3 + 6 + 1 + 1 = 11$ against 12. For each term, the algorithm merges all rows or all columns, whichever gives the smaller number of bits on the edges of the merged instance. The two totals add up to at most $11t|\mathcal{R}|$ for the rectangles, so the smaller one is at most $5.5t|\mathcal{R}|$, against $6t|\mathcal{R}|$ before. This is the factor $11/12$. The exact statement is Lemma 2.

The two expansions side by side. Suppose that a merged instance with S' bits on its edges is solved in time $2^{eS'}$, and let s_0 be the number of bits on the side edges between the merged parts. The running times with the expansion of Equation (19) and with that of Equation (20) are then at most about

$$(22) \quad 2^{t|\mathcal{R}|} \cdot 2^{e(4.5t|\mathcal{R}|+s_0)} \quad \text{and} \quad (t+1)^{|\mathcal{R}|} \cdot 2^{e(5.5t|\mathcal{R}|+s_0)}.$$

In each product the first factor is the number of terms, and the second is the time for one term. For $e = 1$ the two products agree, up to the factor $(t+1)^{|\mathcal{R}|}$. For $e < 1$ the second product is the smaller one as soon as $(1-e)t > \log_2(t+1)$.

The middle column of Table 1. Two rows are joined by g side edges and by $g(g-1)/2$ rectangles, so they share $g\lambda$ bits plus what is counted for these rectangles, which by Expression (21) and the sentence after it is between $4t$ and $6t$ bits each. In all, two rows share between $\frac{2}{3}g^2\lambda$ and about $g^2\lambda$ bits. The side edges between the merged parts carry $g\binom{g}{2}\lambda < S/g$ bits; with the at most $5.5t|\mathcal{R}|$ bits of the rectangles this gives the bound of Step 2, up to an additive constant that depends only on ℓ . The W of a row is a product of the W_a of its g nodes and of functions that are computed in polynomial time, so one value takes g evaluations.

2.6. Balancing. The running time (7) is set by the largest number of bits on an edge. After grid and merge the edges are not uniform: they carry from $\frac{2}{3}g^2\lambda$ to about $g^2\lambda$ bits, and Step 2

bounds only their average, by about $\frac{11}{12}g^2\lambda$. Padding every string to the length of the longest one would cancel the saving.

So we first merge the g nodes into K groups (Figure 3). An edge between two groups combines about $(g/K)^2$ old edges, so its number of bits is close to the average. Lemma 1 finds a grouping in which every group has at most $2g/K$ nodes and every two groups share at most L bits, where $\binom{K}{2}L$ exceeds the number of bits on the edges between the g nodes by at most about S/K . In a random grouping, Chebyshev's inequality and a union bound show that both properties hold with positive probability. This argument is what requires g to be as large as $3K^8$. Randomness appears only in this analysis: the algorithm is deterministic and tries all K^g groupings, a constant number.

The factor $2^{5S/K_{\ell-1}}$ multiplies $T_{\ell-1}$ and is not added to it, for the following reason. A step of the recursive call can be an evaluation of the W of a group G at one argument. By Steps 2 and 3, this value is a sum with one summand for each setting of z_G , the strings $z_a(b)$ of all nodes a in G , and each summand is a product of values of the W_a of the given instance and of functions that are computed in polynomial time. For a rectangle whose two rows (or columns, if the columns were merged) both lie in G , the factor A_i is not written as a sum over θ . It is evaluated directly, as the coefficient of τ^i in $\prod_{j \leq t} (R_j + \tau C_j)$, where R_j and C_j are the two summands of Equation (19) for block j , the row choice and the column choice. A group has at most $(2g/K) \cdot g$ nodes, and each of them has fewer than g^2 strings of λ bits, so z_G has fewer than $2g^4\lambda/K < 5S/K$ bits. The work outside the recursive calls is polynomial for each call, so there is no further additive term.

Where the bits inside a merged part enter. Merging removes no bits. The bits on the edges inside a group enter the running time only through the factor $2^{5S/K_{\ell-1}}$, and the bits on the edges between groups enter only through $T_{\ell-1}(S')$. The exponent $5S/K_{\ell-1}$ bounds the number of bits of z_G for a single group G . It is a maximum over the groups and not a sum, because each evaluation concerns a single group. This gives the right column of Table 1, and the term $5/K_{\ell-1}$ in (9) that is not multiplied by $e_{\ell-1}$.

Appendix A balances with Hoeffding's inequality in place of Chebyshev's (Lemma 8), so that g exceeds K only by a factor polylogarithmic in K ; this is one of four changes behind the [Main Theorem](#) (§2.7).

2.7. The stronger bounds. For Theorem 2 it is enough that $e \rightarrow 0$ along some sequence of constants K . For the bound in m of the [Main Theorem](#), three things change. First, K grows with the input, up to a small power of m , so no cost may be hidden in a constant that depends on K (Appendix A). What matters then is how fast e falls as K grows. Above, ℓ levels give $e = \Theta(\rho^\ell)$, where $\rho = \frac{11}{12}$, and $\log K = \Theta(16^\ell)$, that is, $e = \Theta((\log K)^{-\alpha})$ with $\alpha = \frac{1}{4} \log(1/\rho) \approx 0.03$. Second, with the better balancing of Appendix A one level squares K , up to a polylogarithmic factor, so $\log K = \Theta(2^\ell)$ and $\alpha = \log(1/\rho) \approx 0.125$. Third, Appendix B replaces the functions A_i of Lemma 2 by $O(\lambda^2)$ functions with $\rho = 1/\sqrt{2}$, built from operators on a polynomial ring; the numbers of bits that they cost are logarithms of sums of partition numbers, estimated by the Hardy–Ramanujan formula [HR18], by which an integer w has $2^{\Theta(\sqrt{w})}$ partitions. Together with the better balancing this gives $\alpha = \frac{1}{2}$, that is, $e = \Theta(1/\sqrt{\log K})$, and a k -CNF is solved in time $2^{O(ekm)}$. So neither the better balancing nor the new functions are the main change: each alone would give $\alpha \approx 0.125$.

For the bound in n , a fourth change is needed. Sparsification has to be applied with a parameter ε that tends to 0; in Calabro's form [Cal09] it then leaves formulas in which a variable may occur in $\Delta = (k/\varepsilon)^{O(k)}$ clauses, too many for the graph above, whose number of bits on the edges grows with m . Appendix D uses a graph whose nodes hold variables instead of clauses. The condition that no clause is violated is expanded by inclusion–exclusion over

sets of violated clauses, cut off at about $n/\log \Delta$ clauses, and the nodes need to agree only on which clauses are in the set; assignments that violate more clauses are eliminated by $O(n)$ sparse parity checks. This gives $O(kn \log \Delta)$ bits on the edges. For $k \leq (\log n)^{1/4}$ we have $\log \Delta = O(k \log \log n)$, which gives the factors k^2 and $\log \log n$ in the bound in n , and for larger k that bound is so large that trying all assignments is within it; we do not know whether the factor $\log \log n$ can be removed.

3. NETWORKS

Conventions. $\log = \log_2$. $[x = u]$ is 1 if the statement $x = u$ is true and 0 otherwise. For a positive integer K , $[K] = \{1, \dots, K\}$; $\oplus$ is addition of bits modulo 2; $\{0, 1\}^0$ consists of the empty string. Time is counted in *operations*: operations on words, which need only be long enough to hold an address, and table queries (Definition 3). For a formula, n is the number of variables and m the number of clauses; a 3-CNF is a set of clauses (§6), so $m = O(n^3)$, and a bound stated in n alone holds for every m .

Definition 1. Let $\mathcal{X}, \mathcal{X}'$ be finite sets and $c \geq 0$ an integer. A function $F : \mathcal{X} \times \mathcal{X}' \rightarrow \mathbb{Z}$ *factors through c bits* if there are functions $f : \mathcal{X} \times \{0, 1\}^c \rightarrow \mathbb{Z}$ and $f' : \mathcal{X}' \times \{0, 1\}^c \rightarrow \mathbb{Z}$, called a *factorization* of F , with

$$(23) \quad F(z, z') = \sum_{\theta \in \{0, 1\}^c} f(z, \theta) f'(z', \theta) \quad \text{for all } z \in \mathcal{X}, z' \in \mathcal{X}'.$$

Equivalently, the matrix F (first index z , second index z') is the product $f \cdot (f')^\top$ of two integer matrices (second index θ) with inner dimension 2^c ; in particular its rank is at most 2^c . Two observations:

- (a) If F_1, F_2 on $\mathcal{X} \times \mathcal{X}'$ factor through c_1, c_2 bits, then the pointwise product $F_1 F_2$ factors through $c_1 + c_2$ bits: take $\theta = (\theta_1, \theta_2)$ and multiply the functions.
- (b) For any $\pi : \mathcal{X} \rightarrow \{0, 1\}^c$ and $\pi' : \mathcal{X}' \rightarrow \{0, 1\}^c$, the function $[\pi(z) = \pi'(z')] = \sum_{\theta \in \{0, 1\}^c} [\pi(z) = \theta][\pi'(z') = \theta]$ factors through c bits.

Definition 2. Let $P, B, \tau \geq 1$ be integers. A *network on J with parameters P, B, τ* consists of a finite set J of *nodes*; for each node a , a finite set S_a of *settings* and a function $W_a : S_a \rightarrow \mathbb{Z}$; and for each unordered pair $\{a, b\}$ of distinct nodes, a function $\Psi_{a,b} : S_a \times S_b \rightarrow \mathbb{Z}$ (with $\Psi_{b,a}(z_b, z_a) = \Psi_{a,b}(z_a, z_b)$), an integer $c_{a,b} = c_{b,a} \geq 0$, and a factorization $(f_{a,b}, f_{b,a})$ of $\Psi_{a,b}$ through $c_{a,b}$ bits. The settings of each node a are encoded as bit strings of one fixed length $P_a \leq P$ (we identify S_a with a subset of $\{0, 1\}^{P_a}$, so $|S_a| \leq 2^{P_a}$), and: (i) $\sum_{b \in J \setminus \{a\}} c_{a,b} \leq P$ for every $a \in J$; (ii) all values of all $W_a, \Psi_{a,b}, f_{a,b}$ are below 2^B in absolute value; (iii) an algorithm receives P, B and the numbers $P_a, c_{a,b}$ as input, and can decide membership in S_a and compute any value of $W_a, \Psi_{a,b}, f_{a,b}$ with at most τ operations.

We write z_a for a setting of node a , and we say that a and b *share $c_{a,b}$ bits*. The *value* of the network is

$$(24) \quad Z = \sum_{z \in \prod_{a \in J} S_a} \prod_{a \in J} W_a(z_a) \prod_{\{a,b\} \subseteq J} \Psi_{a,b}(z_a, z_b),$$

where the sum is over all choices $z = (z_a)_{a \in J}$ of one setting $z_a \in S_a$ for each node a , and the second product is over all unordered pairs of distinct nodes. The number $U = 2 \sum_{\{a,b\} \subseteq J} c_{a,b}$, the *total* of the network, counts every shared bit from both of its ends. As a sum over *ordered* pairs, $U = \sum_{(a,b) \in J^2: a \neq b} c_{a,b}$, so $U \leq |J| P$ by (i).

Definition 3. Let $K \geq 2$ and $\lambda, B \geq 1$ be integers. The (K, λ) -*clique* with *tables* $W_1, \dots, W_K$ is the following network on $[K]$. A setting z_w of node w is a $(K-1)$ -tuple of strings $z_w(w') \in \{0, 1\}^\lambda$,

one for each $w' \in [K] \setminus \{w\}$, and $S_w = (\{0, 1\}^\lambda)^{K-1}$ is the set of all such tuples. For $w \neq w'$ and $\theta \in \{0, 1\}^\lambda$,

$$(25) \quad c_{w,w'} = \lambda, \quad \Psi_{w,w'}(z_w, z_{w'}) = [z_w(w') = z_{w'}(w)], \quad f_{w,w'}(z_w, \theta) = [z_w(w') = \theta].$$

The $W_w : S_w \rightarrow \mathbb{Z}$ are arbitrary functions with $|W_w| < 2^B$. A setting z_w is encoded by concatenating its $K-1$ strings $z_w(w')$, so $P_w = P = (K-1)\lambda$. The tables are not written out: an algorithm *queries* them by naming w and $z_w \in S_w$, and receives $W_w(z_w)$ at the cost of one operation.

The pairs $\{w, w'\}$ of distinct nodes are the *edges* of the clique, and $z_w(w')$ and $z_{w'}(w)$ are the two *copies* of the string on the edge $\{w, w'\}$, held at w and at w' ; the factor $\Psi_{w,w'}$ of the edge checks that they agree. So only the terms in which the two copies on every edge agree contribute, and the value is the sum, over all choices of one string in $\{0, 1\}^\lambda$ per edge, of $\prod_{w \in [K]} W_w$ evaluated at the strings on the edges at w . Here τ may be taken to be a constant multiple of $K\lambda$.

Definition 4. For an integer $K \geq 2$ and a real $e \geq 0$, the pair (K, e) is *achievable* if there are an algorithm and a constant C , both depending only on K and e , such that for all integers $\lambda, B \geq 1$ and all tables with $|W_w| < 2^B$ the algorithm, given λ, B and query access, outputs the value of the (K, λ) -clique within $C 2^{eK^2\lambda}(B + \lambda)^C$ operations.

If (K, e) is achievable, so is (K, e_1) for every $e_1 \geq e$. Define e^* as the infimum of all e such that (K, e) is achievable for infinitely many K . Trying all $2^{\lambda K(K-1)/2}$ choices of one string per edge shows that $(K, \frac{1}{2})$ is achievable for every K , so $e^* \leq \frac{1}{2}$. From now on, $\text{poly}(\cdot)$ denotes a polynomial whose degree and coefficients depend only on K and e .

Lemma 1. *Let (K, e) be achievable and $g = 3K^8$. There is a constant $C_1 = C_1(K, e)$ such that the value of every network on $[g]$ with parameters P, B, τ and total U is computed within $C_1 2^{e(U+gP/K)+2gP/K}(P+B)^{C_1\tau}$ operations.*

Proof (Figure 3, middle and right). A *grouping* is a function $\chi : [g] \rightarrow [K]$; its *groups* are the sets $G_\gamma = \chi^{-1}(\gamma)$, $\gamma \in [K]$. For $\gamma \neq \gamma'$ let $Y_{\gamma,\gamma'} = Y_{\gamma',\gamma}$ be the sum of $c_{a,b}$ over all $a \in G_\gamma$ and $b \in G_{\gamma'}$. Call χ *good* if (G1) every group has at most $2g/K$ nodes, and (G2) $Y_{\gamma,\gamma'} \leq (U + gP/K)/K^2$ for all $\gamma \neq \gamma'$.

A *good grouping exists*. Note: we use the probabilistic method here just to show existence, but the eventual algorithm finds the good grouping via brute force search, deterministically (see line 10 of Algorithm 1 in §8). Choose $\chi(a) \in [K]$ independently and uniformly for each node a . Each ordered pair (a, b) contributes with probability $1/K^2$, so $\mathbb{E}Y_{\gamma,\gamma'} = U/K^2$. The indicators of two ordered pairs have covariance 0 if the pairs are disjoint and at most 1 otherwise, and by (i) the ordered pairs that meet a fixed pair have c -values summing to at most $4P$. Hence $\text{Var} Y_{\gamma,\gamma'} \leq \sum_{(a,b) \in [g]^2: a \neq b} c_{a,b} \cdot 4P = 4PU \leq 4gP^2$, and by Chebyshev $\Pr(Y_{\gamma,\gamma'} > U/K^2 + gP/K^3) \leq 4K^6/g = 4/(3K^2)$. By a union bound over the fewer than $K^2/2$ pairs of groups, some pair violates this bound with probability less than $2/3$. A group has mean size g/K and variance at most g/K , so it exceeds $2g/K$ with probability at most K/g ; by a union bound over the K groups, some group does so with probability at most $K^2/g < 1/3$. By a union bound over these two events, the grouping fails to be good with probability less than 1, so a good grouping exists.

The algorithm. Being good depends only on the numbers $c_{a,b}$. Go through all K^g groupings (a constant) and fix the first good one. Let $L = \max\{1, \lfloor (U + gP/K)/K^2 \rfloor\}$, so that $Y_{\gamma,\gamma'} \leq L$. We define a (K, L) -clique whose nodes are the groups. By Definition 3 only its tables $\widetilde{W}_\gamma$ need to be defined. For each pair of groups $\{\gamma, \gamma'\}$ choose an order of the pairs $\{a, b\}$ with $a \in G_\gamma$, $b \in G_{\gamma'}$ (any order will do). In this order, the pair $\{a, b\}$ is assigned a *block* of $c_{a,b}$

consecutive positions of $\{1, \dots, L\}$; the $L - Y_{\gamma, \gamma'}$ positions after the last block hold the *appended bits*. An argument of $\widetilde{W}_\gamma$ is a function $\Theta : [K] \setminus \{\gamma\} \rightarrow \{0, 1\}^L$. For $a \in G_\gamma$ and $b \notin G_\gamma$ let $\theta_{a,b} \in \{0, 1\}^{c_{a,b}}$ consist of the bits of $\Theta(\chi(b))$ in the block of $\{a, b\}$. Put

$$(26) \quad \begin{aligned} \widetilde{W}_\gamma(\Theta) = [\text{all appended bits are 0}] & \sum_{(z_a)_{a \in G_\gamma}} \prod_{a \in G_\gamma} W_a(z_a) \prod_{\{a,b\} \subseteq G_\gamma} \Psi_{a,b}(z_a, z_b) \\ & \prod_{a \in G_\gamma, b \notin G_\gamma} f_{a,b}(z_a, \theta_{a,b}). \end{aligned}$$

Here each z_a ranges over S_a . The value of this clique is by definition a sum over the settings of its nodes γ . The two copies of the string on an edge $\{\gamma, \gamma'\}$ must agree, and both $\widetilde{W}_\gamma$ and $\widetilde{W}_{\gamma'}$ read $\theta_{a,b}$ from the same block of this string, so $\theta_{a,b} = \theta_{b,a}$; the appended bits must be 0. Together with the sums inside the tables, this is a sum over all $z \in \prod_{a \in [g]} S_a$ and over one string $\theta_{a,b} \in \{0, 1\}^{c_{a,b}}$ per pair $\{a, b\}$ of nodes in different groups. Each $\theta_{a,b}$ occurs only in $f_{a,b}$ and $f_{b,a}$, and $\sum_{\theta_{a,b} \in \{0,1\}^{c_{a,b}}} f_{a,b}(z_a, \theta_{a,b}) f_{b,a}(z_b, \theta_{a,b}) = \Psi_{a,b}(z_a, z_b)$, so the value equals Z .

Cost. The sum defining $\widetilde{W}_\gamma(\Theta)$ has at most $(2^P)^{2g/K} = 2^{2gP/K}$ terms, each a product of at most g^2 values. So $|\widetilde{W}_\gamma| < 2^{B'}$ with $B' = \lceil 2gP/K \rceil + g^2B$, and one entry costs $2^{2gP/K} \text{poly}(P+B)\tau$ operations. Run the algorithm for (K, e) with L, B' in the roles of λ, B ; this is legitimate because Definition 4 covers all integer tables. Answer each of its at most $C 2^{eK^2L} (B' + L)^C$ queries by computing the entry. Since $K^2L \leq K^2 + U + gP/K$ and $L \leq 2gP$, the bound follows. $\blacksquare$

4. TWO CROSSING EQUALITIES AS A SUM OF $\lceil \lambda/3 \rceil + 1$ CHEAPER FUNCTIONS

Here $\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v} \in \{0, 1\}^\lambda$ are strings, and $x, y, u, v, p, q, k \in \{0, 1\}^3$ are *blocks* of three bits, on which $\oplus$ acts bit by bit. A function of $(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v})$, or of four blocks (x, y, u, v) , factors through c bits *row-wise* if it does so as a function of $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{u}, \mathbf{v})$, and *column-wise* if it does so as a function of $(\mathbf{x}, \mathbf{u})$ and $(\mathbf{y}, \mathbf{v})$ (Definition 1); §2.5 says “with rows merged” and “with columns merged”. The product $[\mathbf{x} = \mathbf{v}][\mathbf{y} = \mathbf{u}]$ is called a *crossing*: each of its two equalities joins a string of $(\mathbf{x}, \mathbf{y})$ to one of $(\mathbf{u}, \mathbf{v})$, and a string of $(\mathbf{x}, \mathbf{u})$ to one of $(\mathbf{y}, \mathbf{v})$, so both splits separate the two strings of each equality. The starting point is a classical identity, the eight-square identity. For blocks $p = (p_1, p_2, p_3)$ and $q = (q_1, q_2, q_3)$ define the sign

$$(27) \quad \sigma(p, q) = (-1)^{\phi(p, q)}, \quad \phi(p, q) = \sum_{1 \leq i \leq j \leq 3} p_i q_j + q_1 p_2 p_3 + p_1 q_2 p_3 + p_1 p_2 q_3.$$

Eight-square identity (Degen 1818, Graves 1843). *For all blocks p, q, p', q' with $p \oplus q = p' \oplus q'$ and $q \neq q'$,*

$$(28) \quad \sigma(p, q) \sigma(p', q') + \sigma(p, q') \sigma(p', q) = 0.$$

Equivalently, for real variables a_p, b_q , where p, q, k range over the blocks,

$$(29) \quad (\sum_p a_p^2) (\sum_q b_q^2) = \sum_k (\sum_{p, q: p \oplus q = k} \sigma(p, q) a_p b_q)^2.$$

Only (28) is used below; Appendix F proves it by a short computation modulo 2 and shows that the two forms are equivalent. The classical identity writes $(a_1^2 + \dots + a_8^2)(b_1^2 + \dots + b_8^2)$ as a sum of eight squares, each the square of a sum of eight terms $\pm a_i b_j$; Conway and Smith print it in full [CS03, Chapter 6, §6.10]. It says that the norm of the octonions is multiplicative [CS03, §§6.3–6.4]. The second form above is this statement for an orthonormal basis (e_p) of the octonions with $e_p e_q = \sigma(p, q) e_{p \oplus q}$; that the octonions have such a basis is due to Albuquerque and Majid [AM99, Propositions 3.7 and 4.4] (the arXiv version misprints the last term of ϕ). For the history see Baez [Bae02, Section 1].

Let $\psi_k(p, q) = [p \oplus q = k] \sigma(p, q)$, and for blocks x, y, u, v let

$$(30) \quad R(x, y, u, v) = \sum_{k \in \{0,1\}^3} \psi_k(x, y) \psi_k(v, u), \quad C(x, y, u, v) = \sum_{k \in \{0,1\}^3} \psi_k(x, u) \psi_k(v, y).$$

So R factors through 3 bits row-wise, C factors through 3 bits column-wise, both have values in $\{-1, 0, 1\}$, and

$$(31) \quad R + C = 2[x = v][y = u].$$

Indeed, $R = [x \oplus y = v \oplus u] \sigma(x, y) \sigma(v, u)$ and $C = [x \oplus u = v \oplus y] \sigma(x, u) \sigma(v, y)$, and both brackets say $x \oplus y \oplus u \oplus v = 000$. If this fails, all three functions in (31) are 0. If it holds and $y = u$, then $x = v$ and $R = C = \sigma(x, y)^2 = 1$. If it holds and $y \neq u$, then $R + C = 0$ by (28) with $(p, q, p', q') = (x, y, v, u)$.

Definition 5. Let $t = \lceil \lambda/3 \rceil$. Append $3t - \lambda$ zeros to each of $\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}$ and cut the results into blocks $x_s, y_s, u_s, v_s \in \{0, 1\}^3$, $s \in [t]$. For a formal variable T , the functions $A_0, \dots, A_t : (\{0, 1\}^\lambda)^4 \rightarrow \mathbb{Z}$ are the coefficients in

$$(32) \quad \prod_{s=1}^t (R(x_s, y_s, u_s, v_s) + T \cdot C(x_s, y_s, u_s, v_s)) = \sum_{i=0}^t A_i(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}) T^i.$$

Let $c_i^{\text{row}} = \min\{2\lambda, 4t + 3i\}$ and $c_i^{\text{col}} = \min\{2\lambda, 7t - 3i\}$.

Lemma 2. For every $\lambda \geq 1$ and $i \in \{0, \dots, t\}$:

- (a) $A_0 + \dots + A_t = 2^t[\mathbf{x} = \mathbf{v}][\mathbf{y} = \mathbf{u}]$, and A_i has integer values with $|A_i| \leq 2^t$, computable in time $\text{poly}(\lambda)$.
- (b) As a function of $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{u}, \mathbf{v})$, A_i factors through c_i^{row} bits; as a function of $(\mathbf{x}, \mathbf{u})$ and $(\mathbf{y}, \mathbf{v})$, through c_i^{col} bits. The functions of both factorizations have values at most 2^t in absolute value, computable in time $\text{poly}(\lambda)$.
- (c) $c_i^{\text{row}} \leq 2\lambda$, $c_i^{\text{col}} \leq 2\lambda$, and $c_i^{\text{row}} + c_i^{\text{col}} \leq 4\rho\lambda + 8$, where $\rho = 11/12$.

Proof. (a) Set $T = 1$ and use (31); the appended zeros agree. The t factors are polynomials in T of degree at most 1 with coefficients in $\{-1, 0, 1\}$, so multiplying them out one at a time takes time $\text{poly}(\lambda)$, and $|A_i| \leq \binom{t}{i} \leq 2^t$.

(b) Every function $F : \{0, 1\}^c \times \mathcal{X}' \rightarrow \mathbb{Z}$ factors through c bits, by $F(z, z') = \sum_{\theta \in \{0, 1\}^c} [z = \theta] F(\theta, z')$. So A_i factors through 2λ bits, and R and C through 6 bits, both row-wise and column-wise. Now A_i is the sum, over the sets $Q \subseteq [t]$ with $|Q| = i$, of the product that takes C at the blocks $s \in Q$ and R at the others. By (30) and observation (a) after Definition 1, this product has a row-wise factorization (f_Q, f'_Q) through $3(t - i) + 6i$ bits, whose index θ' lists the indices of the blocks $s = 1, \dots, t$ in this order, with values in $\{-1, 0, 1\}$. Let $\theta = (Q, \theta')$, where a set $Q \subseteq [t]$ is written as a string of t bits, and put $f(\cdot, \theta) = f_Q(\cdot, \theta')$ and $f'(\cdot, \theta) = f'_Q(\cdot, \theta')$ if $|Q| = i$, and $f(\cdot, \theta) = f'(\cdot, \theta) = 0$ otherwise. This is a row-wise factorization of A_i through $t + 3(t - i) + 6i = 4t + 3i$ bits. Column-wise the roles of R and C are exchanged, which gives $t + 6(t - i) + 3i = 7t - 3i$ bits. Whenever one of these numbers exceeds 2λ , use the factorization through 2λ bits instead.

(c) The first two bounds hold by definition. Moreover $c_i^{\text{row}} + c_i^{\text{col}} \leq (4t + 3i) + (7t - 3i) = 11t$, and $t \leq (\lambda + 2)/3$, so $11t \leq 11\lambda/3 + 8 = 4\rho\lambda + 8$. $\blacksquare$

Each block costs 3 + 6 bits in the two directions together, plus 1 + 1 bits to name Q : 11 bits, against 6 + 6 without (31). The four-square identity, on blocks of two bits, would give $2 + 4 + 1 + 1 = 4 + 4$, no saving.

5. ONE LEVEL OF THE RECURSION

Let $\rho = 11/12$, as in Lemma 2(c), and for a given λ let $t = \lceil \lambda/3 \rceil$, as in Definition 5.

Proposition 3. *If (K, e) is achievable and $g = 3K^8$, then (g^2, e') is achievable for $e' = (\rho + 2/K)e + 2/K$. Hence $e^* = 0$.*

Proof. Take a (g^2, λ) -clique with value Z and put $N = g^4\lambda$; the goal is cost $2^{e'N} \text{poly}(B + \lambda)$. Identify the nodes with grid points $(a, j) \in [g]^2$ (row a , column j). A *rectangle* q is a choice of two rows $a < a'$ and two columns $j < j'$, adjacent or not; its corners are $(a, j), (a, j'), (a', j), (a', j')$, its *diagonals* are the edges $\{(a, j), (a', j')\}$ and $\{(a, j'), (a', j)\}$, and its four strings are (Figure 3, left)

$$(33) \quad \mathbf{x}_q = z_{(a,j)}((a', j')), \quad \mathbf{y}_q = z_{(a,j')}((a', j)), \quad \mathbf{u}_q = z_{(a',j)}((a, j')), \quad \mathbf{v}_q = z_{(a',j')}((a, j)).$$

So $[\mathbf{x}_q = \mathbf{v}_q][\mathbf{y}_q = \mathbf{u}_q]$ is the product of the clique's factors Ψ on the two diagonals of q . Every edge whose endpoints differ in both row and column is a diagonal of exactly one rectangle, the one given by the rows and columns of its endpoints; call the other edges (inside a row or inside a column) *straight*. The set $\mathcal{R}$ of rectangles has $\binom{g}{2}^2 \leq g^4/4$ elements. Multiply Z by $2^{t|\mathcal{R}|}$, apply Lemma 2(a) to every rectangle, and expand:

$$(34) \quad 2^{t|\mathcal{R}|} Z = \sum_{\iota: \mathcal{R} \rightarrow \{0, \dots, t\}} Z_\iota, \\ Z_\iota = \sum_z \prod_w W_w(z_w) \prod_{\{w, w'\} \text{ straight}} [z_w(w') = z_{w'}(w)] \prod_{q \in \mathcal{R}} A_{\iota(q)}(\mathbf{x}_q, \mathbf{y}_q, \mathbf{u}_q, \mathbf{v}_q).$$

Here $z = (z_w)_{w \in [g]^2}$ ranges over all choices of one setting $z_w \in S_w$ for each grid point w , and the first product is over all $w \in [g]^2$.

Two networks for each ι . Fix ι . The *row network* has node set $[g]$, the rows. A setting of row a is the tuple $(z_{(a,j)})_{j \in [g]}$ of the settings of its g grid points; S_a is the set of these tuples, encoded by concatenation in $g(g^2 - 1)\lambda \leq P$ bits, where $P = g^3\lambda$. W_a is the product of the g table entries of row a and of the factors of the straight edges inside row a . $\Psi_{a,a'}$ is the product of the factors of the g straight edges between rows a and a' and of $A_{\iota(q)}(\mathbf{x}_q, \mathbf{y}_q, \mathbf{u}_q, \mathbf{v}_q)$ over the $g(g-1)/2$ rectangles q with these two rows. Each factor in (34) is used exactly once, so the value is Z_ι . Since $\mathbf{x}_q, \mathbf{y}_q$ belong to row a and $\mathbf{u}_q, \mathbf{v}_q$ to row a' , observations (a), (b) after Definition 1 and the first factorization in Lemma 2(b) show that $\Psi_{a,a'}$ factors through

$$(35) \quad c_{a,a'} = g\lambda + \sum_{q \text{ with rows } a,a'} c_{\iota(q)}^{\text{row}} \leq g\lambda + g(g-1)\lambda = g^2\lambda \quad \text{bits}.$$

The *column network* is defined in the same way with rows and columns exchanged. There $\mathbf{x}_q, \mathbf{u}_q$ and $\mathbf{y}_q, \mathbf{v}_q$ are separated, so the second factorization applies, and (35) holds with c^{col} in place of c^{row} . Both networks satisfy Definition 2 with $P = g^3\lambda$ (so $gP = N$), with $B_1 = gB + g^2\lambda + 1$ in the role of B , and with $\tau = \text{poly}(B + \lambda)$: indeed $\sum_{a' \in [g] \setminus \{a\}} c_{a,a'} \leq g^3\lambda$, $|W_a| < 2^{gB}$, and $\Psi_{a,a'}$ and the functions of its factorization are products of indicators and of $g(g-1)/2$ numbers of absolute value at most $2^t \leq 2^\lambda$.

The smaller total. Let $U_{\text{row}}, U_{\text{col}}$ be the totals U of the two networks. By (35), and because U counts ordered pairs, $U_{\text{row}} = g^2(g-1)\lambda + 2 \sum_{q \in \mathcal{R}} c_{\iota(q)}^{\text{row}}$ and $U_{\text{col}} = g^2(g-1)\lambda + 2 \sum_{q \in \mathcal{R}} c_{\iota(q)}^{\text{col}}$. By Lemma 2(c), $c_i^{\text{row}} + c_i^{\text{col}} \leq 11\lambda/3 + 8$. Hence

$$(36) \quad \min(U_{\text{row}}, U_{\text{col}}) \leq \frac{1}{2}(U_{\text{row}} + U_{\text{col}}) \leq \frac{N}{g} + |\mathcal{R}|(11\lambda/3 + 8) \leq N(\rho + \frac{1}{g}) + 2g^4.$$

Without Lemma 2 each rectangle would contribute 4λ to each total, and both totals would be $g^3(g-1)\lambda \approx N$. Rows or columns are chosen once per ι , for the whole grid; nothing is chosen rectangle by rectangle.

Cost. Apply Lemma 1 to the network with the smaller total. By (36), $gP = N$ and $1/g \leq 1/K$, its exponent satisfies $e(U + N/K) + 2N/K \leq N((\rho + 2/K)e + 2/K) + 2eg^4 = e'N + 2eg^4$, and the other factors are $\text{poly}(B + \lambda)$. There are $(t + 1)^{|\mathcal{R}|} \leq (B + \lambda)^{g^4/4}$ functions ι . This is a polynomial in $B + \lambda$ because g depends only on K , and Definition 4 allows such a factor; 2^{2eg^4} is a constant. Adding the Z_ι and dividing by $2^{t|\mathcal{R}|}$ gives Z . So (g^2, e') is achievable.

$e^* = 0$. Let $e > e^*$; then (K, e) is achievable for infinitely many K . Let $e'' > \rho e$. For all large K we have $(\rho + 2/K)e + 2/K \leq e''$, so $(9K^{16}, e'')$ is achievable for infinitely many K , and therefore $e^* \leq e''$. Letting $e'' \rightarrow \rho e$ and $e \rightarrow e^*$ gives $e^* \leq \rho e^*$. Since $0 \leq e^* \leq \frac{1}{2}$ and $\rho < 1$, $e^* = 0$. That is, for every $\eta > 0$, (K, η) is achievable for infinitely many K . ■

6. PROOF OF THEOREM 2

A *literal* is a variable (positive literal) or its negation (negative literal), a *clause* is a nonempty set of literals, and a *3-CNF* is a finite set of clauses with at most 3 literals each.

Lemma 4 (Sparsification Lemma; Impagliazzo, Paturi and Zane [IPZ01, Corollary 1, p. 523], in the form of Dell et al. [DHMTW14, Lemma A.1 with condition (1')], for clauses of at most 3 literals). *For every $\varepsilon > 0$ there is a constant $M(\varepsilon) \geq 1$ such that for every 3-CNF φ on n variables one can compute deterministically, in time $2^{\varepsilon n}$ times a polynomial in n , at most $2^{\varepsilon n}$ 3-CNFs on the same variables, each with at most $M(\varepsilon) \cdot n$ clauses, such that an assignment satisfies φ if and only if it satisfies at least one of them, and no two of them share a satisfying assignment.*

That no two of the formulas share a satisfying assignment is what condition (1') of Dell et al. adds; they sketch its proof, and Appendix C sketches the analogous change in the algorithm of Calabro [Cal09].

Proof of Theorem 2. Constants and reduction. We may assume $\delta \leq 1$. Let $M = M(\delta/3)$ and $\delta' = \delta/(3M) \leq \frac{1}{3}$. Since $e^* = 0 < \delta'/12$ and achievability is monotone in e (Definition 4), $(K, \delta'/12)$ is achievable for infinitely many K . Fix such a $K \geq 72/\delta'$ and put $g = 3K^8$; all these constants depend on δ only. By Lemma 4 with $\varepsilon = \delta/3$, the number of satisfying assignments of the given formula is the sum of those of the formulas of the lemma, so it suffices to compute the number of satisfying assignments of a 3-CNF Φ on n variables with $m \leq Mn$ clauses, in time $O(2^{\delta' m})$ plus a polynomial in n . If $m = 0$ the number is 2^n ; let $m \geq 1$.

Clusters, occurrences, equality tests (Figure 1; there each clause is drawn as its own node, each bit on a solid line between two clauses is an equality test, and the shaded sets, labeled G_1, G_2, G_3 , are three clusters). Divide the clauses of Φ into g clusters, numbered by $a \in [g]$, of at most $\bar{m} = \lceil m/g \rceil$ clauses each; clusters may be empty. An *occurrence* is a pair $o = (E, \omega)$ of a clause $E \in \Phi$ and a literal $\omega \in E$, that is, one position of one clause. It is an occurrence of the variable of ω , whether ω is positive or negative, and it lies in the cluster of E . For each variable, list all its occurrences in Φ , across all clusters, in some order $o_1, \dots, o_d$ (any order will do); the pairs $\{o_1, o_2\}, \{o_2, o_3\}, \dots, \{o_{d-1}, o_d\}$ of consecutive occurrences are the *equality tests*, or *tests*, of this variable. An occurrence lies in at most two tests. The total number of tests is the number of occurrences minus the number of variables that occur, hence at most $3\bar{m} - 1$. The *network* of Φ is the following network on $[g]$ with $P = 6\bar{m}$ and $B = 1$.

- A setting z_a of node a is a bit string with one bit $z_a[o]$ for each occurrence o in cluster a , and S_a is the set of all these bit strings. Their length is at most $3\bar{m} \leq P$.² The bit $z_a[o]$ is a proposed value of the *variable* of which o is an occurrence, not of the literal: call $o = (E, \omega)$ *true* if ω is positive and $z_a[o] = 1$, or ω is negative and $z_a[o] = 0$.

² P is taken as large as $6\bar{m}$ because $\sum_{b \neq a} c_{a,b}$ can reach $6\bar{m}$.

- $W_a : S_a \rightarrow \{0, 1\}$ is 1 exactly if every clause of cluster a has a true occurrence and $z_a[o] = z_a[o']$ for every test $\{o, o'\}$ with both occurrences in cluster a .
- For $a \neq b$, $c_{a,b}$ is the number of tests with one occurrence in cluster a and the other in cluster b . Choose an order of these tests, and let $z_a|_b \in \{0, 1\}^{c_{a,b}}$ list the bits of z_a at their occurrences in cluster a ; likewise $z_b|_a$. Put $\Psi_{a,b}(z_a, z_b) = [z_a|_b = z_b|_a]$, factorized by $f_{a,b}(z_a, \theta) = [z_a|_b = \theta]$ and $f_{b,a}(z_b, \theta) = [z_b|_a = \theta]$, $\theta \in \{0, 1\}^{c_{a,b}}$.

Definition 2 holds: $\sum_{b \neq a} c_{a,b} \leq 2 \cdot 3\bar{m} = P$, all values are 0 or 1, and τ is polynomial in m , after a preparation in time polynomial in n . Every term of the value Z is 0 or 1. Each test is checked exactly once, by a W_a or a $\Psi_{a,b}$, so a term is 1 exactly if all occurrences of each variable carry the same bit and every clause has a true occurrence. Hence Z is the number of assignments of the variables occurring in Φ that satisfy Φ , and Φ has $Z \cdot 2^{n'}$ satisfying assignments, where n' of the n variables do not occur in Φ .

Cost. U counts each test between two clusters twice, so $U \leq 2(3m - 1) < 6m$; and $gP = 6g[m/g] \leq 6(m + g)$. With $e = \delta'/12$, $K \geq 72$ and $1/K \leq \delta'/72$, the exponent in Lemma 1 is

$$(37) \quad \begin{aligned} eU + e \frac{gP}{K} + \frac{2gP}{K} &< \frac{1}{2} \delta' m + \frac{1}{144} \delta' (m + g) + \frac{1}{6} \delta' (m + g) \\ &= \frac{97}{144} \delta' m + \frac{25}{144} \delta' g \leq \frac{3}{4} \delta' m + \delta' g. \end{aligned}$$

The remaining factor $C_1 2^{\delta'g} (P + B)^{C_1} \tau$ is polynomial in m , hence $O(2^{\delta'm/4})$. So, with the preparation, Z is computed in time $O(2^{\delta'm})$ plus a polynomial in n , and $\delta'm \leq \delta' Mn = \delta n/3$. Lemma 4 yields at most $2^{\delta n/3}$ such formulas, in time $2^{\delta n/3}$ times a polynomial in n , and multiplying by $2^{n'}$ and adding up takes time polynomial in n for each of them, so the total time is $O(2^{\delta n})$. ■

7. CONSEQUENCES

Definition 6 (s_k , s_∞ and ETH [IP01, p. 368]; SETH [CIP09, p. 77]). For an integer $k \geq 1$, a k -CNF is a finite set of clauses with at most k literals each, and k -SAT is the problem of deciding whether a given k -CNF is satisfiable. Let s_k be the infimum of all $\delta \geq 0$ such that some deterministic algorithm decides k -SAT on n variables in time $O(2^{\delta n})$, and let $s_\infty = \sup_{k \geq 3} s_k$. The *Exponential Time Hypothesis* (ETH) is the statement that $s_k > 0$ for all $k \geq 3$. The *Strong Exponential Time Hypothesis* (SETH) is the statement $s_\infty = 1$.

Trying all assignments gives $s_k \leq 1$, so $s_\infty \leq 1$. A k -CNF is also a $(k+1)$ -CNF, so $s_k \leq s_{k+1}$. Hence $s_\infty = \lim_{k \rightarrow \infty} s_k$, and ETH is the same as $s_3 > 0$. Impagliazzo and Paturi [IP01, p. 374] pose $s_\infty = 1$ as an open problem, and Calabro, Impagliazzo and Paturi [CIP09] give it its name. Impagliazzo and Paturi [IP01] do not say whether the algorithms in the definition of s_k are deterministic or randomized, and Calabro, Impagliazzo and Paturi [CIP09] allow randomized ones. A deterministic algorithm is a randomized algorithm that uses no random bits, so allowing randomness can only make s_k smaller, and Corollary 5 holds under each of these conventions.

Corollary 5. $s_k = 0$ for every $k \geq 3$. Hence $s_\infty = 0$, and both ETH and SETH are false.

Proof. A 3-CNF is satisfiable if and only if its number of satisfying assignments is positive, so Theorem 2 gives $s_3 = 0$, which is the negation of ETH; for this, the proof of Theorem 2 needs Lemma 4 only in the form of Impagliazzo, Paturi and Zane [IPZ01, Corollary 1, p. 523], without the condition that no two of the formulas share a satisfying assignment: without it the sum may count an assignment more than once, but it is still positive exactly if the given formula is satisfiable. By a theorem of Impagliazzo, Paturi and Zane [IPZ01, Theorem 3, p. 525], which rests on their Sparsification Lemma, $s_3 = 0$ implies $s_k = 0$ for every $k \geq 3$; the reduction (sparsify, then split every long clause into clauses of 3 literals with new variables) uses no

randomness (Calabro, Impagliazzo and Paturi [CIP06, Lemma 1] state the Sparsification Lemma with a deterministic algorithm). In terms of the numbers s_k this is stated by Impagliazzo and Paturi [IP01, Theorem 1, p. 368]: the statements “For all $k \geq 3$, $s_k > 0$ ”, “For some k , $s_k > 0$ ” and “ $s_3 > 0$ ” are equivalent. Hence $s_\infty = \lim_k s_k = 0 \neq 1$. Put differently: $s_\infty = 1$ gives $s_k > 0$ for some k , hence $s_3 > 0$; so SETH implies ETH, and refuting ETH refutes SETH. ■

The nondeterministic variant NSETH [CGIMPS16], which says that for every $\varepsilon > 0$ there is a k such that the tautologies among the k -DNF formulas on n variables are not recognized in nondeterministic time $2^{(1-\varepsilon)n}$, is false as well: a k -DNF is a tautology exactly if its negation, a k -CNF, is unsatisfiable, and a deterministic algorithm is in particular a nondeterministic one.

Circuit lower bounds. By a theorem of Williams [Wil13, Theorem 6.1], if ETH is false then E^{NP} , the class of languages decided in time $2^{O(n)}$ with an oracle for NP, does not have linear-size circuits (see also Williams [Wil14b, Theorem 5.8] and Jahanjou, Miles and Viola [JMV15, Corollary 2, item (1), with Corollary 3, item (1')]). So, by Corollary 5, there is a language in E^{NP} that is not decided by (non-uniform) Boolean circuits of size $O(n)$.

Corollary 6. *For every $k \geq 3$ and $\delta > 0$ there is a deterministic algorithm that computes the number of satisfying assignments of a k -CNF on n variables in time $O(2^{\delta n})$. For $k = 3$ this says that the counting version $\#ETH$ of ETH [DHMTW14, p. 21:5] is false.*

Proof. For $k = 3$ this is Theorem 2. By the work of Dell et al. [DHMTW14, Lemma A.1 with condition (1'), and the proof of Theorem 1.1, pp. 21:24–21:25] this case suffices. First, their form of the Sparsification Lemma, which Lemma 4 states for $k = 3$, turns a k -CNF on n variables, for any $\varepsilon > 0$, into at most $2^{\varepsilon n}$ k -CNFs on the same variables, each with $O(n)$ clauses, no two of which share a satisfying assignment, so that their numbers of satisfying assignments add up. Second, each of these is turned into a 3-CNF with $O(n)$ clauses and $O(n)$ variables and the same number of satisfying assignments, by the standard width reduction, which introduces a constant number of new variables for each clause. The number is preserved if each new variable is forced, by three clauses of at most 3 literals, to equal the disjunction of the two literals that it replaces in a long clause; Dell et al. [DHMTW14] do not spell this out. The constants in $O(n)$ depend on k and ε , so choosing $\varepsilon = \delta/3$ first and then applying Theorem 2 with a small enough value in place of δ gives time $O(2^{\delta n})$. ■

Corollary 7. *For all $\delta > 0$ and $k \geq 3$ there are deterministic algorithms that, given a graph on n vertices, with any number of edges, and an integer s , compute in time $O(2^{\delta n})$ the number of its independent sets of size s , and likewise of its cliques, vertex covers and dominating sets of size s and of its proper colorings with k colors. In particular, they decide Independent Set, Clique, Vertex Cover, Dominating Set and k -Coloring, since subsets of independent sets and of cliques, and supersets of vertex covers and of dominating sets, are again such sets.*

Proof. For deciding Independent Set, Clique, Vertex Cover and k -Coloring, the corollary follows from prior work: these problems, with the number n of vertices as the parameter, reduce to 3-SAT by subexponential reduction families [IPZ01, Theorems 3 to 5, pp. 525–526], which carry over Theorem 2’s time $O(2^{\delta n})$ for every $\delta > 0$ (see also Flum and Grohe [FG04, Corollary 7.1]). These results cover neither Dominating Set nor the counts, for which we argue as follows.

Let the variable x_v say whether the vertex v is in the set S . As in the work of Impagliazzo, Paturi and Zane [IPZ01, proof of Theorem 5], a circuit with $O(n)$ gates of two inputs tests whether $|S| = s$; a new variable for each gate, clauses of at most 3 literals that force it to equal the value of the gate, and a clause that makes the output true give a 3-CNF Γ with $O(n)$ variables and $O(n)$ clauses. Each S of size s extends to exactly one satisfying assignment of Γ , and no other S to any. Add the clause $\{\neg x_u, \neg x_v\}$ for each edge uv ; Corollary 6 then counts the independent sets of size s . The cliques of size s are the independent sets of size s of the

complement graph, and the vertex covers of size s are the complements of the independent sets of size $n - s$. The proper colorings correspond one to one to the satisfying assignments of a k -CNF on kn variables, which says that each vertex has exactly one color and that the ends of each edge have different colors. For dominating sets add to Γ , for each vertex v , the clause that consists of x_v and of x_u for all neighbors u of v , and, for an integer $w \geq 3$, shorten these clauses as Calabro, Impagliazzo and Paturi do [CIP06, §3]: while a clause has more than w literals, pick w of them and branch, either deleting its other literals or setting the w literals to false. Each satisfying assignment survives in exactly one branch. A path to a leaf takes the first branch at most n times and the second, which sets w more of the variables x_v , at most n/w times, so there are $2^{O(n \log w/w)}$ leaves, each a w -CNF on the variables not yet set. Choose w so large that the number of leaves is at most $2^{\delta n/2}$, apply Corollary 6 to each leaf, and add up. All these formulas have $O(n)$ variables, so Corollary 6, applied with a small enough value in place of δ , gives time $O(2^{\delta n})$. ■

8. THE ALGORITHM IN PSEUDOCODE

The proofs go from small K to large K ; Algorithm 1 runs the other way. For $\ell \geq 0$ let

$$(38) \quad K_0 = 64, \quad e_0 = \frac{1}{2}, \quad g_\ell = 3K_\ell^8, \quad K_{\ell+1} = g_\ell^2 = 9K_\ell^{16}, \quad e_{\ell+1} = (\rho + 2/K_\ell)e_\ell + 2/K_\ell.$$

By brute force and Proposition 3, (K_ℓ, e_ℓ) is achievable for every ℓ , and $e_\ell \rightarrow 0$, because $\rho < 1$ and $K_\ell \rightarrow \infty$.

The number of nodes drops from K_ℓ to $K_{\ell-1} \approx K_\ell^{1/16}$ per level, so the depth of the recursion is ℓ : a constant for fixed δ . From one level to the next the total $K^2\lambda$ shrinks by a factor of

Algorithm 1. Counting the satisfying assignments of a 3-CNF. The numbers K_ℓ, g_ℓ, e_ℓ are defined in (38). For Theorem 2, $\varepsilon = \delta/3$, δ' is as in §6, and ℓ is the first level with $K_\ell \geq 72/\delta'$ and $e_\ell \leq \delta'/12$. The algorithm of Theorem A has the same structure; it differs in the choice of ε and ℓ , in the numbers K_ℓ, g_ℓ , which are those of (40) (but $3K_\ell^8$ clusters in line 8), in the groupings and the number L (Lemma 8) and, for other pieces, in lines 16 and 19; for the decision statement, Theorem A(d), whose list of formulas is Calabro's [Cal09], the value returned in line 4 is positive exactly if φ is satisfiable; see Appendix A.

```

1 procedure #SAT( $\varphi$ )  $\triangleright \varphi$  is a 3-CNF on  $n$  variables
2   for each  $\Phi$  in the list that Lemma 4 computes from  $\varphi$  and  $\varepsilon$  do
3      $N_\Phi \leftarrow 2^{n'} \cdot \text{COUNT}(\Phi)$   $\triangleright n'$  of the  $n$  variables do not occur in  $\Phi$ 
4   return  $\sum_\Phi N_\Phi$   $\triangleright$  the number of satisfying assignments of  $\varphi$ 
5 procedure COUNT( $\Phi$ )  $\triangleright \Phi$  is a 3-CNF with  $m$  clauses; §6
6   if  $m = 0$  then return 1
7   choose the level  $\ell$   $\triangleright$  see above
8   return NETWORK( $\ell$ , the network of  $\Phi$  on  $g_\ell$  clusters)
9 procedure NETWORK( $\ell$ , a network on  $g_\ell$  nodes)  $\triangleright$  returns its value; Lemma 1,  $K = K_\ell$ 
10  try all groupings into  $K_\ell$  groups until one is good  $\triangleright$  good: (G1) and (G2) in the proof of Lemma 1
11  let  $L$  and the tables  $\widetilde{W}_\gamma$  be as in the proof of Lemma 1  $\triangleright$  not stored; computed when queried
12  return CLIQUE( $\ell$ ,  $L$ ,  $(\widetilde{W}_\gamma)_\gamma$ )
13 procedure CLIQUE( $\ell$ ,  $\lambda$ , the tables  $(W_w)_w$  of a  $(K_\ell, \lambda)$ -clique)  $\triangleright$  its value; Proposition 3,  $K = K_{\ell-1}$ 
14  if  $\ell = 0$  then return the sum of  $\prod_{w \in [K_0]} W_w$  over all choices of one string per edge (brute
    force)
15  put the  $K_\ell = g^2$  nodes on a  $g \times g$  grid, where  $g = g_{\ell-1}$ ;  $sum \leftarrow 0$ 
16  for each function  $\iota : \mathcal{R} \rightarrow \{0, \dots, \lceil \lambda/3 \rceil\}$  do  $\triangleright \mathcal{R}$  is the set of rectangles
17    form the row network and the column network of  $\iota$   $\triangleright g$  nodes each; both have the value  $Z_\iota$ 
18     $sum \leftarrow sum + \text{NETWORK}(\ell - 1, \text{the one of the two with the smaller total } U)$ 
19  return  $sum / 2^{\lceil \lambda/3 \rceil |\mathcal{R}|}$   $\triangleright$  by (34)
```

about ρ or more, and brute force at the bottom costs about $2^{K_0^2\lambda/2}$; this is where $e_\ell \approx \frac{1}{2}\rho^\ell$ comes from (precisely, unrolling (38) gives $\frac{1}{2}\rho^\ell \leq e_\ell \leq 0.56\rho^\ell$; the excess comes from the two terms $2/K_\ell$ in (38)). A table passed to CLIQUE is not an array but a procedure: a query to $\widetilde{W}_\gamma$ is answered by evaluating the sum in Lemma 1, which in turn queries the tables one level up.

REFERENCES

- [AGHP92] N. Alon, O. Goldreich, J. Håstad, R. Peralta. Simple constructions of almost k -wise independent random variables. *Random Structures & Algorithms* 3(3):289–304, 1992.
- [AKK22] P. Austrin, P. Kaski, K. Kubjas. Tensor network complexity of multilinear maps. *Theory of Computing* 18(16):1–54, 2022. Conference version: ITCS 2019. arXiv:1712.09630.
- [AM99] H. Albuquerque, S. Majid. Quasialgebra structure of the octonions. *Journal of Algebra* 220(1):188–224, 1999. doi:10.1006/jabr.1998.7850. arXiv:math/9802116 (numbering cited from this version).
- [Bae02] J. C. Baez. The octonions. *Bulletin of the American Mathematical Society* 39(2):145–205, 2002. Erratum: 42:213, 2005. arXiv:math/0105155.
- [BCKLOSW26] C. Brand, R. Curticapean, P. Kaski, B. Li, I. Orzel, T. Seppelt, J. Wang. Beyond bilinear complexity: what works and what breaks with many modes? In *CCC 2026*, LIPIcs 383, pages 11:1–11:23. doi:10.4230/LIPIcs.CCC.2026.11. arXiv:2602.11975.
- [BCKN12] H. L. Bodlaender, M. Cygan, S. Kratsch, J. Nederlof. Solving weighted and counting variants of connectivity problems parameterized by treewidth deterministically in single exponential time. arXiv:1211.1505, 2012.
- [BI18] A. Backurs, P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM Journal on Computing* 47(3):1087–1097, 2018. doi:10.1137/15M1053128. Conference version: STOC 2015. arXiv:1412.0348.
- [BK24] A. Björklund, P. Kaski. The asymptotic rank conjecture and the set cover conjecture are not both true. In *STOC 2024*, pages 859–870. doi:10.1145/3618260.3649656. arXiv:2310.11926.
- [Cal09] C. Calabro. *The Exponential Complexity of Satisfiability Problems*. Ph.D. dissertation, University of California, San Diego, 2009.
- [CC17] J.-Y. Cai, X. Chen. Complexity of counting CSP with complex weights. *Journal of the ACM* 64(3), Article 19, 2017. doi:10.1145/2822891. arXiv:1111.2384.
- [CDLMNOPSW16] M. Cygan, H. Dell, D. Lokshitanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, M. Wahlström. On problems as hard as CNF-SAT. *ACM Transactions on Algorithms* 12(3), Article 41, 2016. doi:10.1145/2925416. Conference version: CCC 2012. arXiv:1112.2275.
- [CFKLMPPS15] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshitanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- [CGIMPS16] M. L. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, S. Schneider. Nondeterministic extensions of the Strong Exponential Time Hypothesis and consequences for non-reducibility. In *ITCS 2016*, pages 261–270. doi:10.1145/2840728.2840746. ECCC TR15-148.
- [CHKX06] J. Chen, X. Huang, I. A. Kanj, G. Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences* 72(8):1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007. Conference version: STOC 2004.
- [CHR24] L. Chen, S. Hirahara, H. Ren. Symmetric exponential time requires near-maximum circuit size. In *STOC 2024*, pages 1990–1999. arXiv:2309.12912.
- [CIP06] C. Calabro, R. Impagliazzo, R. Paturi. A duality between clause width and clause density for SAT. In *CCC 2006*, pages 252–260.
- [CIP09] C. Calabro, R. Impagliazzo, R. Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Computation (IWPEC 2009)*, Lecture Notes in Computer Science 5917, pages 75–85. Springer, 2009. doi:10.1007/978-3-642-11269-0_6.
- [CKN12] M. Cygan, S. Kratsch, J. Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. arXiv:1211.1506, 2012.
- [Coo06] S. Cook. The P versus NP problem. In J. Carlson, A. Jaffe, A. Wiles (eds.), *The Millennium Prize Problems*. Clay Mathematics Institute and American Mathematical Society, 2006.
- [Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *STOC 1971*, pages 151–158. doi:10.1145/800157.805047.
- [CS03] J. H. Conway, D. A. Smith. *On Quaternions and Octonions: Their Geometry, Arithmetic, and Symmetry*. A K Peters, 2003.

- [CVZ19] M. Christandl, P. Vrana, J. Zuiddam. Asymptotic tensor rank of graph tensors: beyond matrix multiplication. *computational complexity* 28(1):57–111, 2019. doi:10.1007/s00037-018-0172-8. arXiv:1609.07476.
- [CW16] T. M. Chan, R. Williams. Deterministic APSP, Orthogonal Vectors, and more: quickly derandomizing Razborov–Smolensky. In *SODA 2016*, pages 1246–1255. Journal version: *ACM Transactions on Algorithms* 17(1), Article 2, 2020. doi:10.1145/3402926.
- [CX15] R. Curticapean, M. Xia. Parameterizing the permanent: genus, apices, minors, evaluation mod 2^k . In *FOCS 2015*. arXiv:1511.02321.
- [CXZ21] H. Chu, M. Xiao, Z. Zhang. An improved upper bound for SAT. In *AAAI 2021*, pages 3707–3714. doi:10.1609/aaai.v35i5.16487.
- [DGH+02] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, U. Schöning. A deterministic $(2 - 2/(k + 1))^n$ algorithm for k -SAT based on local search. *Theoretical Computer Science* 289(1):69–83, 2002.
- [DGJ09] M. Dyer, L. A. Goldberg, M. Jerrum. The complexity of weighted Boolean #CSP. *SIAM Journal on Computing* 38(5):1970–1986, 2009. doi:10.1137/070690201. arXiv:0704.3683.
- [DHMTW14] H. Dell, T. Husfeldt, D. Marx, N. Taslaman, M. Wahlén. Exponential time complexity of the permanent and the Tutte polynomial. *ACM Transactions on Algorithms* 10(4), Article 21, 2014. doi:10.1145/2635812. arXiv:1206.1775.
- [DJW05] V. Dahllöf, P. Jonsson, M. Wahlström. Counting models for 2SAT and 3SAT formulae. *Theoretical Computer Science* 332(1–3):265–291, 2005. doi:10.1016/j.tcs.2004.10.037.
- [FG04] J. Flum, M. Grohe. Parameterized complexity and subexponential time. *Bulletin of the EATCS* 84:71–100, 2004.
- [Fre02] E. Frenkel. Vertex algebras and algebraic curves. Séminaire Bourbaki 1999/2000, exposé 875. *Astérisque* 276:299–339, 2002. arXiv:math/0007054 (numbering cited from this version).
- [GL99] A. Galluccio, M. Loeb. On the theory of Pfaffian orientations. I. Perfect matchings and permanents. *Electronic Journal of Combinatorics* 6, #R6, 1999.
- [Her14] T. Hertli. 3-SAT faster and simpler—Unique-SAT bounds for PPSZ hold in general. *SIAM Journal on Computing* 43(2):718–729, 2014. Conference version: FOCS 2011. arXiv:1103.2165.
- [Hir00] E. A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning* 24(4):397–420, 2000. doi:10.1023/A:1006340920104.
- [HKZZ19] T. D. Hansen, H. Kaplan, O. Zamir, U. Zwick. Faster k -SAT algorithms using biased-PPSZ. In *STOC 2019*, pages 578–589. doi:10.1145/3313276.3316359.
- [Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association* 58(301):13–30, 1963.
- [HR18] G. H. Hardy, S. Ramanujan. Asymptotic formulae in combinatory analysis. *Proceedings of the London Mathematical Society* (2) 17:75–115, 1918.
- [Hru24] P. Hrubeš. A subquadratic upper bound on sum-of-squares composition formulas. In *CCC 2024*, LIPIcs 300, pages 12:1–12:11. doi:10.4230/LIPIcs.CCC.2024.12. ECCC TR24-026, revision 1 (numbering cited from this version).
- [Hru24b] P. Hrubeš. A subquadratic upper bound on Hurwitz’s problem and related non-commutative polynomials. ECCC TR24-099, 2024.
- [Hur98] A. Hurwitz. Über die Composition der quadratischen Formen von beliebig vielen Variabeln. *Nachrichten von der Königlichen Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, pages 309–316, 1898.
- [HWY11] P. Hrubeš, A. Wigderson, A. Yehudayoff. Non-commutative circuits and the sum-of-squares problem. *Journal of the American Mathematical Society* 24(3):871–898, 2011. doi:10.1090/S0894-0347-2011-00694-2. Conference version: STOC 2010, pages 667–676. ECCC TR10-021 (numbering cited from this version).
- [IMP12] R. Impagliazzo, W. Matthews, R. Paturi. A satisfiability algorithm for AC^0 . In *SODA 2012*, pages 961–972. arXiv:1107.3127.
- [IP01] R. Impagliazzo, R. Paturi. On the complexity of k -SAT. *Journal of Computer and System Sciences* 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- [IPZ01] R. Impagliazzo, R. Paturi, F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences* 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
- [JMV15] H. Jahanjou, E. Miles, E. Viola. Local reductions. In *ICALP 2015*, pages 749–760. arXiv:1311.3171 (numbering cited from this version).

- [Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher (eds.), *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. doi:10.1007/978-1-4684-2001-2_9.
- [Koi06] M. Koivisto. Optimal 2-constraint satisfaction via sum-product algorithms. *Information Processing Letters* 98(1):24–28, 2006. doi:10.1016/j.ipl.2005.11.013.
- [Kul99] O. Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science* 223(1–2):1–72, 1999.
- [Kut07] K. Kutzkov. New upper bound for the #3-SAT problem. *Information Processing Letters* 105(1):1–5, 2007. doi:10.1016/j.ipl.2007.06.017.
- [Lev73] L. A. Levin. Universal sequential search problems. *Problemy Peredachi Informatsii* 9(3):115–116, 1973 (in Russian).
- [Li24] Z. Li. Symmetric exponential time requires near-maximum circuit size: simplified, truly uniform. In *STOC 2024*, pages 2000–2007. doi:10.1145/3618260.3649615. arXiv:2310.17762.
- [Liu18] S. Liu. Chain, generalization of covering code, and deterministic algorithm for k -SAT. In *ICALP 2018*, LIPIcs 107, pages 88:1–88:13. doi:10.4230/LIPIcs.ICALP.2018.88. arXiv:1804.07901.
- [LMS11] D. Lokshtanov, D. Marx, S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS* 105:41–71, 2011.
- [LMS18a] D. Lokshtanov, D. Marx, S. Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Transactions on Algorithms* 14(2), Article 13, 2018. doi:10.1145/3170442. Conference version: SODA 2011. arXiv:1007.5450.
- [LMS18b] D. Lokshtanov, D. Marx, S. Saurabh. Slightly superexponential parameterized problems. *SIAM Journal on Computing* 47(3):675–702, 2018. doi:10.1137/16M1104834. Conference version: SODA 2011. arXiv:1902.08723.
- [LY22] J. Li, T. Yang. $3.1n - o(n)$ circuit lower bounds for explicit functions. In *STOC 2022*, pages 1180–1193. doi:10.1145/3519935.3519976.
- [Mar10] D. Marx. Can you beat treewidth? *Theory of Computing* 6(5):85–112, 2010.
- [Mat20] The mathlib Community. The Lean mathematical library. In *CPP 2020*, pages 367–381. doi:10.1145/3372885.3373824. arXiv:1910.09336.
- [MN99] A. Matsuo, K. Nagatomo. *Axioms for a Vertex Algebra and the Locality of Quantum Fields*. MSJ Memoirs 4. Mathematical Society of Japan, 1999. arXiv:hep-th/9706118 (numbering cited from this version).
- [MS08] I. L. Markov, Y. Shi. Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing* 38(3):963–981, 2008. arXiv:quant-ph/0511069.
- [MS11] R. A. Moser, D. Scheder. A full derandomization of Schöning’s k -SAT algorithm. In *STOC 2011*, pages 245–252. arXiv:1008.4067.
- [MS85] B. Monien, E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics* 10(3):287–295, 1985. doi:10.1016/0166-218X(85)90050-2.
- [MU21] L. de Moura, S. Ullrich. The Lean 4 theorem prover and programming language. In *CADE 28*, Lecture Notes in Computer Science 12699, pages 625–635, 2021. doi:10.1007/978-3-030-79876-5_37.
- [MW20] C. D. Murray, R. R. Williams. Circuit lower bounds for nondeterministic quasi-polytime from a new easy witness lemma. *SIAM Journal on Computing* 49(5):STOC18-300–STOC18-322, 2020. doi:10.1137/18M1195887.
- [Nas20] E. Naslund. The partition rank of a tensor and k -right corners in $\mathbb{F}_q^n$. *Journal of Combinatorial Theory, Series A* 174:105190, 2020. arXiv:1701.04475 (numbering cited from v3).
- [NN93] J. Naor, M. Naor. Small-bias probability spaces: efficient constructions and applications. *SIAM Journal on Computing* 22(4):838–856, 1993.
- [NP85] J. Nešetřil, S. Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae* 26(2):415–419, 1985.
- [Ope26] OpenAI. The Exponential Time Hypothesis is false. Preprint, October 26, 2026. <https://github.com/openai/math/blob/main/preprints/The-Exponential-Time-Hypothesis-is-False-October-25-2026/paper.pdf>.
- [Oru14] R. Orús. A practical introduction to tensor networks: matrix product states and projected entangled pair states. *Annals of Physics* 349:117–158, 2014. doi:10.1016/j.aop.2014.06.013. arXiv:1306.2164.
- [PPSZ05] R. Paturi, P. Pudlák, M. E. Saks, F. Zane. An improved exponential-time algorithm for k -SAT. *Journal of the ACM* 52(3):337–364, 2005. Conference version: FOCS 1998.

- [PPZ99] R. Paturi, P. Pudlák, F. Zane. Satisfiability coding lemma. *Chicago Journal of Theoretical Computer Science*, Volume 1999, Article 11. Conference version: FOCS 1997, pages 566–574.
- [Pra24] K. Pratt. A stronger connection between the asymptotic rank conjecture and the set cover conjecture. In *STOC 2024*, pages 871–874. doi:10.1145/3618260.3649620. arXiv:2311.02774.
- [PW10] M. Pătraşcu, R. Williams. On the possibility of faster SAT algorithms. In *SODA 2010*, pages 1065–1075. doi:10.1137/1.9781611973075.86.
- [PX23] J. Peng, M. Xiao. Further improvements for SAT in terms of formula length. *Information and Computation* 294, Article 105085, 2023. doi:10.1016/j.ic.2023.105085. arXiv:2105.06131.
- [RS19] E. Robeva, A. Seigal. Duality of graphical models and tensor networks. *Information and Inference* 8(2):273–288, 2019. doi:10.1093/imaiai/iay009. arXiv:1710.01437.
- [RZ25] B. Rossman, D. Zhu. Multiquadratic sum-of-squares lower bounds imply $VNC^1 \neq VNP$. arXiv:2512.01227, 2025.
- [Sch21] D. Scheder. PPSZ is better than you think. In *FOCS 2021*, pages 205–216. Journal version, without the case $k = 3$: *TheoretCS* 3, Article 5, 2024. arXiv:2207.11071.
- [Sch99] U. Schöning. A probabilistic algorithm for k -SAT and constraint satisfaction problems. In *FOCS 1999*, pages 410–414. Journal version: *Algorithmica* 32(4):615–623, 2002. doi:10.1007/s00453-001-0094-7.
- [SH90] R. E. Stearns, H. B. Hunt III. Power indices and easier hard problems. *Mathematical Systems Theory* 23(4):209–225, 1990. doi:10.1007/BF02090776.
- [SS17] D. Scheder, J. P. Steinberger. PPSZ for general k -SAT—making Hertli’s analysis simpler and 3-SAT faster. In *CCC 2017*, LIPIcs 79, pages 9:1–9:15. doi:10.4230/LIPIcs.CCC.2017.9.
- [Sta12] R. P. Stanley. *Enumerative Combinatorics, Volume 1*. Second edition. Cambridge University Press, 2012.
- [Vad12] S. P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science* 7(1–3):1–336, 2012.
- [Vas18] V. Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians (ICM 2018)*, pages 3447–3487. World Scientific, 2018. doi:10.1142/9789813272880_0188.
- [VW21] N. Vyas, R. Williams. On Super Strong ETH. *Journal of Artificial Intelligence Research* 70:473–495, 2021. doi:10.1613/jair.1.11859. Conference version: SAT 2019.
- [Wil05] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science* 348(2–3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.
- [Wil13] R. Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing* 42(3):1218–1244, 2013. doi:10.1137/10080703X. Conference version: STOC 2010.
- [Wil14] R. Williams. Nonuniform ACC circuit lower bounds. *Journal of the ACM* 61(1), Article 2, 2014. doi:10.1145/2559903. Conference version: CCC 2011.
- [Wil14b] R. Williams. Algorithms for circuits and circuits for algorithms. In *CCC 2014*, pages 248–261. doi:10.1109/CCC.2014.33.
- [Wil16] R. R. Williams. Strong ETH breaks with Merlin and Arthur: short non-interactive proofs of batch evaluation. In *CCC 2016*, LIPIcs 50, pages 2:1–2:17. doi:10.4230/LIPIcs.CCC.2016.2. arXiv:1601.04743.
- [Wil19] R. R. Williams. Some estimated likelihoods for computational complexity. In *Computing and Software Science: State of the Art and Perspectives*, Lecture Notes in Computer Science 10000, pages 9–26. Springer, 2019. doi:10.1007/978-3-319-91908-9_2.
- [WJ08] M. J. Wainwright, M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning* 1(1–2):1–305, 2008. doi:10.1561/22000000001.
- [Yam05] M. Yamamoto. An improved $O(1.234^m)$ -time deterministic algorithm for SAT. In *ISAAC 2005*, Lecture Notes in Computer Science 3827, pages 644–653. Springer, 2005.

APPENDIX A. THE RUNNING TIME AS A FUNCTION OF m AND n

Here the number K of nodes grows with the input, so no factor may be hidden in a constant that depends on K . Theorem A turns any decomposition of the crossing into pieces, with the three properties that Proposition 3 uses, into a bound for cliques (part (b) of Theorem A). From the proofs of Lemma 1 and Proposition 3 we take over constructions and identities only; all costs are counted again, because $\text{poly}(\cdot)$ there hides constants that depend on K and e . In

this appendix $\ln$ is the natural logarithm, C_1 is an absolute constant, and C_0, C_2, C_3, C_4, C_5 depend only on the constant c of the following definition. The numbers K_ℓ, g_ℓ, e_ℓ of (40) are not those of (38), and ρ , which is $11/12$ in the main text, is a parameter here.

Definition 7. Let $\frac{1}{2} \leq \rho < 1$ be real and $c \geq 1$ an integer. *Pieces with parameters* (ρ, c) consist, for every $\lambda \geq 1$, of an integer $d \geq 1$, a finite set $\mathcal{I}$ of integers and functions $A_i : (\{0, 1\}^\lambda)^4 \rightarrow \mathbb{Z}$, $i \in \mathcal{I}$, such that:

- (a) $\sum_{i \in \mathcal{I}} A_i = d [\mathbf{x} = \mathbf{v}][\mathbf{y} = \mathbf{u}]$;
- (b) A_i factors through c_i^{row} bits as a function of $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{u}, \mathbf{v})$, and through c_i^{col} bits as a function of $(\mathbf{x}, \mathbf{u})$ and $(\mathbf{y}, \mathbf{v})$, where $c_i^{\text{row}} + c_i^{\text{col}} \leq 4\rho\lambda + c \log(2\lambda)$;
- (c) $|\mathcal{I}| \leq (2\lambda)^c$; the number d , the elements of $\mathcal{I}$ and all values of the A_i and of the functions of these factorizations are below $2^{(2\lambda)^c}$ in absolute value; and one algorithm computes each of them, and $\mathcal{I}$, c_i^{row} , c_i^{col} , from λ , i and the arguments within $2^{c\lambda}$ operations.

By Lemma 2, the functions $A_0, \dots, A_t$ of Definition 5 are pieces with parameters $(11/12, c)$, with $\mathcal{I} = \{0, \dots, t\}$, $d = 2^t$ and the numbers $c_i^{\text{row}}, c_i^{\text{col}}$ defined there, for which $c_i^{\text{row}} + c_i^{\text{col}} \leq 11\lambda/3 + 8$. This holds for every integer $c \geq 8$ such that $2^{c\lambda}$ exceeds the polynomial running times in Lemma 2 for all $\lambda \geq 1$. We may and do assume $c_i^{\text{row}}, c_i^{\text{col}} \leq 2\lambda$: any $F : \{0, 1\}^{2\lambda} \times \mathcal{X}' \rightarrow \mathbb{Z}$ factors through 2λ bits, as $F(z, z') = \sum_{\theta \in \{0, 1\}^{2\lambda}} [z = \theta] F(\theta, z')$, and a longer factorization is replaced by this one (and c by $c + 2$, which pays for computing c_i^{row} or c_i^{col} first).

Lemma 8. *Let $K, g \geq 2$ be integers and $0 < \zeta \leq 1$. Take a network on $[g]$ with parameters P, B, τ and total U , and assume that (i) $g = 3K^8$ and $\zeta = 1/(2K)$, or (ii) K divides g , $c_{a,b} \leq P/g$ for all $a \neq b$, and $2\zeta^2 g \geq K \ln(2gK)$. Put $L = \lceil (U + 2\zeta g P)/K^2 \rceil$ and $B' = \lceil 2gP/K \rceil + g^2 B$. If an algorithm outputs the value of every (K, L) -clique with $|W_w| < 2^{B'}$ within $\mathcal{T}$ operations, then the value of the network is computed within $(K^g + \mathcal{T} \cdot 2^{2gP/K} \tau)(g(P + B))^{C_1}$ operations, by one procedure, the same for all K, g, ζ and all networks, which calls that algorithm.*

Proof. Groupings χ , their groups G_γ and the numbers $Y_{\gamma, \gamma'}$ are as in the proof of Lemma 1. Some grouping has at most $2g/K$ nodes in every group and $Y_{\gamma, \gamma'} \leq L$ for all $\gamma \neq \gamma'$. In case (i) this is shown in that proof, which uses nothing about cliques at this point. In case (ii), let χ be uniformly random among the groupings in which every group has exactly g/K nodes; then each G_γ is a uniformly random set of g/K nodes. By Hoeffding's inequality for sampling without replacement [Hoe63, Theorem 2 and Section 6], the sum S of a uniformly random set of j entries of a list of numbers in $[0, R]$ satisfies $\Pr(S \geq \mathbb{E}S + \zeta j R) \leq \exp(-2j\zeta^2)$, which for $j = g/K$ is at most $1/(2gK)$. Let $c_{a,a} = 0$ and $c_a = \sum_{b \in [g]} c_{a,b} \leq P$, so that $U = \sum_{a \in [g]} c_a$. Apply the inequality, for each γ , to $\sum_{a \in G_\gamma} c_a$, with $R = P$ and expectation U/K ; and, for each node a and each γ' , to $\sum_{b \in G_{\gamma'}} c_{a,b}$, with $R = P/g$ and expectation c_a/K . These are $(g+1)K < 2gK$ events, so for some χ none of them occurs. For this χ we have $\sum_{a \in G_\gamma} c_a < (U + \zeta g P)/K$ and $\sum_{b \in G_{\gamma'}} c_{a,b} < (c_a + \zeta P)/K$ for all γ, γ' and a . Summing the second bound over the g/K nodes $a \in G_\gamma$, where $\gamma \neq \gamma'$, and inserting the first gives

$$(39) \quad Y_{\gamma, \gamma'} < \frac{1}{K} \sum_{a \in G_\gamma} c_a + \frac{g}{K} \cdot \frac{\zeta P}{K} < \frac{U + 2\zeta g P}{K^2}.$$

The algorithm is that of Lemma 1: find the first such grouping among all K^g , define the (K, L) -clique with the tables $\widetilde{W}_\gamma$, whose value is that of the network, run the given algorithm on it, and answer each query by computing the entry. That construction works for all g and K , and uses only these two properties of the grouping. As shown there, $|\widetilde{W}_\gamma| < 2^{B'}$, and an entry is a sum of at most $2^{2gP/K}$ terms, each a product of at most g^2 values. Testing a grouping, finding the blocks in an argument Θ , which has fewer than KL bits, and the arithmetic for one

term, on integers of at most B' bits, take time polynomial in $g(P + B)$, because $K \leq g$ in both cases and $L \leq gP + 1$. $\blacksquare$

Theorem A. Suppose that there are pieces with parameters (ρ, c) , and let $\alpha' = \log(1/\rho)$. Let $x_0 \geq 18$ be an integer with $2^{x_0} \geq c + 5$, and for $\ell \geq 0$ let

$$(40) \quad \begin{aligned} K_0 &= 2^{x_0}, & \zeta_\ell &= 2^{-\ell-5}, & g_\ell &= 2^{3\ell+x_0}K_\ell, & K_{\ell+1} &= g_\ell^2, \\ e_0 &= \frac{1}{2}, & e_{\ell+1} &= (\rho + 3\zeta_\ell)e_\ell + 3/K_\ell. \end{aligned}$$

(a) $\log K_\ell = (3x_0 + 6)2^\ell - 6\ell - 2x_0 - 6$, so $K_\ell < K_{\ell+1} \leq K_\ell^{10}$; and $\frac{1}{2}\rho^\ell \leq e_\ell \leq 0.85\rho^\ell$, where $\rho^\ell \leq ((3x_0 + 6)/\log K_\ell)^{\alpha'}$.

(b) One deterministic algorithm, given $\ell \geq 0$, $\lambda, B \geq 1$ and query access to tables with $|W_w| < 2^B$, outputs the value of the (K_ℓ, λ) -clique within $2^{e_\ell K_\ell^2 \lambda} (K_\ell(\lambda + B))^{C_0 K_\ell^2}$ operations.

(c) One deterministic algorithm counts the satisfying assignments of a k -CNF with $m \geq 2$ clauses in which n variables occur, as assignments of these n variables, in time $2^{O(km/(\log m)^{\alpha'})}$, where the O depends on ρ and c but not on k . For fixed k and $n \geq 4$ variables the time is $2^{O(n \log \log n/(\log n)^{\alpha'})}$, provided that a number between $0.85\rho^\ell$ and ρ^ℓ can be computed from ℓ in time polynomial in K_ℓ , as is the case for $\rho = 11/12$ and for $\rho = 1/\sqrt{2}$.

(d) One deterministic algorithm decides 3-SAT on $n \geq 2$ variables in time $2^{O(n/(\log n)^{\alpha'/10})}$.

Proof. (a) The numbers $x_\ell = \log K_\ell$ satisfy $x_{\ell+1} = 2x_\ell + 6\ell + 2x_0$, which gives the formula by induction. Since $x_\ell \geq 2^\ell x_0 \geq \max\{\ell, x_0\}$, we get $x_{\ell+1} \leq 10x_\ell$. Clearly $e_\ell \geq \frac{1}{2}\rho^\ell$. By $\rho \geq \frac{1}{2}$, the numbers $y_\ell = e_\ell \rho^{-\ell}$ satisfy $y_{\ell+1} \leq (1 + 2^{-\ell-2})y_\ell + 6 \cdot 2^\ell/K_\ell$, and $K_\ell \geq 2^{18(\ell+1)}$. So $y_\ell \leq \exp(\sum_{j \geq 0} 2^{-j-2}) (\frac{1}{2} + \sum_{j \geq 0} 6 \cdot 2^{j-18(j+1)}) \leq e^{1/2} \cdot 0.51 < 0.85$. Finally $\rho^\ell = (2^\ell)^{-\alpha'}$ and $2^\ell \geq x_\ell/(3x_0 + 6)$.

(b) By induction on ℓ , with $C_0 = 2(c + C_4 + 1) + C_5$ for the constants C_4, C_5 that occur below; they depend only on c . For $\ell = 0$, brute force sums $2^{\lambda K_0(K_0-1)/2} \leq 2^{e_0 K_0^2 \lambda}$ terms, each at a cost of at most $(K_0(\lambda + B))^{C_5}$. For the step from ℓ to $\ell + 1$ write K, e, g, ζ for $K_\ell, e_\ell, g_\ell, \zeta_\ell$. Take a (g^2, λ) -clique with $|W_w| < 2^B$, and put $N = g^4 \lambda$ and $\Xi = g^2(\lambda + B)$. Follow the proof of Proposition 3, with this g , with the A_i , $i \in \mathcal{I}$, in place of $A_0, \dots, A_t$ and with d in place of 2^t , up to (36); this part of that proof works for every $g \geq 2$. Thus $d^{|\mathcal{R}|}Z = \sum_i Z_i$ over the $|\mathcal{I}|^{|\mathcal{R}|} \leq \Xi c g^4/4$ functions $\iota : \mathcal{R} \rightarrow \mathcal{I}$, and each Z_i is the value of the row network and of the column network. In (35) we have $c_{\iota(q)}^{\text{row}} \leq 2\lambda$ as before, so again $P = g^3 \lambda$, $gP = N$, and moreover $c_{a,a'} \leq g^2 \lambda = P/g$. By Definition 7(c), Definition 2 holds with $B_1 = gB + g^2(2\lambda)^c$ in the role of B and with $\tau = 2^{c\lambda} \Xi^{C_2}$. By Definition 7(b), (36) becomes

$$(41) \quad U = \min(U_{\text{row}}, U_{\text{col}}) \leq \frac{N}{g} + |\mathcal{R}|(4\rho\lambda + c \log(2\lambda)) \leq N(\rho + \frac{1}{g}) + \frac{c}{4} g^4 \log(2\lambda).$$

Case (ii) of Lemma 8 applies to the network with the smaller total (with B_1 in the role of B): K divides g and $2\zeta^2 g = 2^{\ell+x_0-9}K$, while $2gK \leq g^2 = K_{\ell+1}$ gives $\ln(2gK) \leq 0.7 \log K_{\ell+1} \leq 1.4(3x_0 + 6)2^\ell$, and $1.4(3x_0 + 6) \leq 2^{x_0-9}$ for $x_0 \geq 18$. Also $1/g \leq \zeta$, since $\zeta g = 2^{2\ell+x_0-5}K$. So the lemma's L satisfies

$$(42) \quad K^2 L \leq U + 2\zeta N + K^2 \leq (\rho + 3\zeta)N + \frac{c}{4} g^4 \log(2\lambda) + K^2.$$

Crude bounds give $L \leq N + 1$, $B' \leq N + g^2 B_1$, $g(P + B_1) \leq \Xi^{c+4}$ and $K(L + B') \leq \Xi^{c+5}$. By the induction hypothesis and $e \leq 1$, the lemma's $\mathcal{T}$ may be taken as

$$(43) \quad \mathcal{T} = 2^{eK^2 L} (K(L + B'))^{C_0 K^2} \leq 2^{e(\rho+3\zeta)N} \Xi^{c g^4/4 + K^2 + (c+5)C_0 K^2}.$$

(The number L depends on ι ; the bound holds for every ι .) The set $\mathcal{I}$ and the numbers $d, c_i^{\text{row}}, c_i^{\text{col}}$ are computed once. All integers that occur have a number of bits polynomial in

Ξ , so each of the following takes at most Ξ^{C_3} operations: finding the numbers $c_{a,a'}$ and the totals of both networks for one ι ; adding one Z_ι ; computing $d^{|\mathcal{R}|}$ and dividing by it. A sum of numbers that are at least 2 is at most their product, $K^g \leq \Xi^g$, $(2|\mathcal{I}| + 2)2^{c\lambda} \leq \Xi^{c+1}\tau$, and $c\lambda \leq N/K$. So the total number of operations is at most

$$(44) \quad |\mathcal{I}|^{|\mathcal{R}|} \left((K^g + \mathcal{T} 2^{2N/K} \tau) \Xi^{(c+4)C_1} + 2\Xi^{C_3} \right) + (2|\mathcal{I}| + 2) 2^{c\lambda} + \Xi^{C_3} \\ \leq \Xi^{cg^4/4+g+C_4} \mathcal{T} 2^{2N/K+c\lambda} \leq 2^{e_{\ell+1}N} \Xi^{cg^4/2+g+K^2+C_4+(c+5)C_0K^2}.$$

Since $(c+5)K^2 \leq 2^{x_0}K^2 \leq g^4/2$, the exponent of Ξ is at most $(c+C_4+1)g^4 + \frac{1}{2}C_0g^4 \leq C_0g^4 = C_0K_{\ell+1}^2$ by the choice of C_0 . The algorithm uses ℓ , the pieces, the totals $U_{\text{row}}, U_{\text{col}}$ and the constant c (through x_0 and B_1), but neither ρ nor e_ℓ .

(c) Define the network of a k -CNF as in §6. A cluster now has at most $k\lceil m/g \rceil$ occurrences, each in at most two tests, and there are at most $km-1$ tests. So $P = 2k\lceil m/g \rceil$, $B = 1$, $U < 2km$, $gP \leq 2k(m+g)$, and τ is polynomial in km after a preparation that lists the occurrences and the tests. The preparation takes time polynomial in km , because the variables are named $x_1, \dots, x_n$ (§2), so that the formula has length $O(km \log n)$, and $n \leq km$; the term $O(\log(km))$ below covers it. Let $m \geq K_0^9$ (for smaller m , try all assignments), let $K = K_\ell$, $e = e_\ell$ for the largest ℓ with $K_\ell \leq m^{1/9}$, and $g = 3K^8 \leq m$. By (a), $K \geq m^{1/90}$ and $e = O((\log m)^{-\alpha'})$. By case (i) of Lemma 8, with $\mathcal{T}$ from (b), the logarithm of the number of operations is at most

$$(45) \quad eK^2L + C_0K^2 \log(K(L+B')) + g \log K + 2gP/K + O(\log(km)) \\ \leq 2ekm + O(km/K + K^8 \log K + K^2 \log(km)),$$

and every term after $2ekm$ is $O(km^{1-1/90})$. The second statement follows from Theorem D(a), whose conditions $n \geq K_0^{17}$ and $k \leq (\log n)^{1/4}$ hold for all but finitely many n when k is fixed; for the finitely many other n the algorithm tries all assignments, and a formula with $k \leq 2$ is read as a 3-CNF. Its hypothesis holds by (a) and (b), with the larger of C_0 and 10: $e_\ell \geq \frac{1}{2}\rho^\ell \geq 2^{-\ell-1} \geq 1/\log K_\ell$, and a number between $0.85\rho^\ell$ and ρ^ℓ lies in $[e_\ell, 2e_\ell]$.

(d) We use a form of Lemma 4 in which the formulas may share satisfying assignments, but which is otherwise stronger [Cal09, Theorem 5.15; outlined in CIP06, §6]: one deterministic algorithm takes $\varepsilon \in (0, 1]$ as an input, runs in time $2^{\varepsilon n}$ times a polynomial in n that does not depend on ε , and achieves $M(\varepsilon) = O(\varepsilon^{-9})$. Each formula that it produces has $m \leq M(\varepsilon)n$ clauses. The function $m/(\log m)^{\alpha'}$ increases with $m \geq 3$, because $\ln 3 > 1 \geq \alpha'$, and $\log(M(\varepsilon)n) \geq \log n$ because $M(\varepsilon) \geq 1$. So by (c) each formula (one with fewer than 3 clauses is decided at once; in the others the variables that occur are first renamed to fit the convention of §2, in time polynomial in $m \log n$) takes time $2^{O(M(\varepsilon)n/(\log n)^{\alpha'})}$, and the total time is $2^{\varepsilon n}$ times a polynomial in n times $2^{O(M(\varepsilon)n/(\log n)^{\alpha'})}$. With $M(\varepsilon) = O(\varepsilon^{-9})$, the choice $\varepsilon = (\log n)^{-\alpha'/10}$ makes both exponents $O(n/(\log n)^{\alpha'/10})$. This gives (d). ■

With the pieces of Lemma 2, $\alpha' = \log(12/11) = 0.125\dots$; with those of Appendix B, $\alpha' = \frac{1}{2}$. In Appendix C, α denotes this last value, $\frac{1}{2}$. If the constant c of the pieces is not known explicitly, the theorem only says that such algorithms exist.

The bound is purely asymptotic. For a k -CNF with $m = rn$ clauses the main term $2e_\ell km$ of the exponent is below n , the exponent of trying all assignments, only if $e_\ell < 1/(2kr)$. Since $e_\ell \geq \frac{1}{2}\rho^\ell$, this requires $\rho^\ell < 1/(kr)$, which for $\rho = 1/\sqrt{2}$ means $2^\ell > (kr)^2$. The algorithm uses the largest ℓ with $K_\ell \leq m^{1/9}$, so $\log m \geq 9 \log K_\ell$, and $\log K_\ell \geq 60 \cdot 2^\ell - 6\ell - 42$ by Theorem A(a), because $x_0 \geq 18$. For $k = 3$ and $r = 1$ this gives $\ell \geq 4$ and $\log m > 8000$; for $r = 5$ it gives $\ell \geq 8$ and $\log m > 1.3 \cdot 10^5$. For smaller m the bound proved here is no better than that of trying all assignments, so it says nothing about inputs of any realistic size.

APPENDIX B. PIECES WITH $\rho = 1/\sqrt{2}$

Theorem B. *There are pieces with parameters $(1/\sqrt{2}, c)$, for an absolute constant c .*

With Theorem A this gives the exponent $\log(1/\rho) = \frac{1}{2}$ of $\log m$, that is, the bound in m of the Main Theorem. The pieces are matrix entries of operators on a polynomial ring, and the number of shared bits of a piece is the logarithm of a number of partitions.

The ring and its operators. For integers k and $j \geq 0$ let $\binom{k}{j} = k(k-1) \cdots (k-j+1)/j!$, and $\binom{k}{j} = 0$ for $j < 0$; Pascal's rule and Vandermonde's identity $\sum_j \binom{k}{i-j} \binom{k'}{j} = \binom{k+k'}{i}$ hold for all integers k, k' . A *partition* μ of E is a multiset of positive integers, its *parts*, with sum $|\mu| = E$. We list the parts as $\mu_1, \mu_2, \dots$ in some order; sums and products over i run over this list, so equal parts are distinguished. Let $\text{mult}_k(\mu)$ be the number of parts equal to k , and $\mathbf{z}_\mu = \prod_k k^{\text{mult}_k(\mu)} \text{mult}_k(\mu)!$. Let $p(E)$ be the number of partitions of E , with $p(0) = 1$ and $p(E) = 0$ for $E < 0$. Adding a part 1 shows $p(E) \leq p(E+1)$, and by the asymptotic formula $p(E) \sim \exp(\pi\sqrt{2E/3})/(4E\sqrt{3})$ of Hardy and Ramanujan [HR18, (1.41)] there is a constant C_{HR} such that, with $c_{\text{HR}} = \pi\sqrt{2/3}/\ln 2$,

$$(46) \quad c_{\text{HR}}\sqrt{E} - \log E - C_{\text{HR}} \leq \log p(E) \quad (E \geq 1), \quad \log p(E) \leq c_{\text{HR}}\sqrt{E} + C_{\text{HR}} \quad (E \geq 0).$$

Let $V = \mathbb{Q}[t_1, t_2, \dots]$, with the basis of monomials $t^\mu = \prod_i t_{\mu_i}$; $[t^\mu]f$ is the coefficient of t^μ in f . Let V_E be the span of the t^μ with $|\mu| = E$, the polynomials of *weight* E ; so $\dim V_E = p(E)$, $V_0 = \mathbb{Q}$, and $V_E = 0$ for $E < 0$. For $k \geq 1$ let $\nabla_{-k}f = t_k f$ and $\nabla_k f = k \partial f / \partial t_k$; so ∇_k maps V_E to V_{E-k} for every $k \neq 0$. For a monomial $a = t^\mu$ and $r \in \mathbb{Z}$ define the linear map

$$(47) \quad a[r] = \sum_i \prod_{k_i} \binom{-k_i - 1}{\mu_i - 1} \cdot \prod_{i: k_i < 0} \nabla_{k_i} \cdot \prod_{i: k_i > 0} \nabla_{k_i} : V \rightarrow V,$$

summed over all choices of one nonzero integer k_i for each part μ_i with the given sum, and extend it linearly to all $a \in V$. In each term the differentiations are applied first. On $f \in V_E$ a term vanishes if its positive k_i add up to more than E , so only finitely many terms are nonzero. Clearly:

(O1) $a[r]$ maps V_E to V_{E+r} , so $a[r]f = 0$ if $f \in V_E$ and $E+r < 0$; it maps integer polynomials to integer polynomials if a is one; and $1[r] = [r=0] \text{Id}$.

(O2) Sorting the terms by the integer chosen for one part n : if $a = t_n a'$ with a monomial a' , then $a[r] = \sum_{l \geq n} \binom{l-1}{n-1} \nabla_{-l} a'[r-l] + \sum_{k \geq 1} \binom{-k-1}{n-1} a'[r+k] \nabla_k$.

(O3) If $b \in V_h$, then $a[-h]b \in V_0$ is a number, and only terms with all k_i positive contribute to it: if some k_i is negative, the positive ones add up to more than h .

Now let $h \geq 1$, $b \in V_h$, and $a = \sum_{h'=0}^h \bar{a}_{h'}$ with $\bar{a}_{h'} \in V_{h'}$. Put $a^{(i)} = \sum_{h'} \binom{h'+h}{i} \bar{a}_{h'}$ and, for $s \in \mathbb{Z}$,

$$(48) \quad Q_s(a, b) = \sum_{i=0}^{2h} (-1)^i a^{(i)} [s-h+i] b[h-s-i].$$

Lemma 9. *On V_E , for every $E \geq 0$: (a) $Q_s(a, b)$ does not change if the two factors are exchanged in all products of (48) at once; (b) $Q_s(a, b) = 0$ for $|s| > E+h$; (c) $\sum_{|s| \leq E+h} Q_s(a, b) = a[-h]b \cdot \text{Id}$.*

Proof. All three claims are linear in a and b , so let $a = t^\mu$ with $|\mu| = h' \leq h$ and $b = t^\nu$; then $a^{(i)} = \binom{H}{i} a$ with $H = h' + h$.

Series. Let $\mathcal{L} = \mathbb{Q}[\omega_1, \omega_1^{-1}, \omega_2, \omega_2^{-1}]$, let $\mathcal{L}_+$ be the ring of formal series $\sum_{j \geq j_0} q_j(\omega_1) \omega_2^j$ with $q_j \in \mathbb{Q}[\omega_1, \omega_1^{-1}]$, and $\mathcal{L}_-$ the same with ω_1 and ω_2 exchanged. Both contain $\mathcal{L}$ and are closed

under termwise differentiation, which satisfies the product rule, and $\omega_1 - \omega_2$ has the inverse $\varepsilon_+ = \sum_{j \geq 0} \omega_1^{-j-1} \omega_2^j$ in $\mathcal{L}_+$ and $\varepsilon_- = -\sum_{j \geq 0} \omega_2^{-j-1} \omega_1^j$ in $\mathcal{L}_-$. Differentiating $(\omega_1 - \omega_2) \varepsilon_{\pm} = 1$ gives $\partial_{\omega_1} \varepsilon_{\pm} = -\varepsilon_{\pm}^2$ and $\partial_{\omega_2} \varepsilon_{\pm} = \varepsilon_{\pm}^2$. So for $j, n \geq 1$ there is a number $\Gamma_{j,n}$, the same for both signs, with

$$(49) \quad \frac{1}{(j-1)!(n-1)!} \partial_{\omega_1}^{j-1} \partial_{\omega_2}^{n-1} \varepsilon_{\pm}^2 = \Gamma_{j,n} \varepsilon_{\pm}^{j+n}.$$

Generating functions. For $n \geq 1$ put $J_n^+(\omega_1) = \sum_{l \geq 1} \binom{l-1}{n-1} \nabla_{-l} \omega_1^{l-n}$ and $J_n^-(\omega_1) = \sum_{k \geq 1} \binom{-k-1}{n-1} \nabla_k \omega_1^{-k-n}$. By (47), $Y(a, \omega_1) := \sum_r a[r] \omega_1^{r-h'} = \sum_S \prod_{i \in S} J_{\mu_i}^+(\omega_1) \prod_{i \notin S} J_{\mu_i}^-(\omega_1)$, summed over all sets S of parts of μ . Take further variables x_n, x'_n ($n \geq 1$), write $x^\mu = \prod_i x_{\mu_i}$ and $\mu! = \prod_k \text{mult}_k(\mu)!$, and put $A^\pm = \sum_n x_n J_n^\pm(\omega_1)$, $B^\pm = \sum_n x'_n J_n^\pm(\omega_2)$. Exponentials of these are formal power series in the x_n, x'_n , and an equation between products of them means: apply both sides to a polynomial, and take the coefficient of a monomial in the t_k and of a monomial in the x_n, x'_n . In all products below, e^{A^+} stands to the left of e^{A^-} , and e^{B^+} to the left of e^{B^-} . If moreover e^{B^-} does not stand to the left of e^{A^+} , the result lies in $\mathcal{L}_+$: the differentiations from B^- are then applied before all multiplications, so their indices add up to at most the weight of the polynomial; a power of ω_2 then leaves finitely many choices for the multiplications from B^+ ; and the weights bound the rest. Likewise it lies in $\mathcal{L}_-$ if e^{A^-} does not stand to the left of e^{B^+} . Expanding the exponentials gives $e^{A^+} e^{A^-} = \sum_\mu x^\mu Y(t^\mu, \omega_1)/\mu!$, and likewise $e^{B^+} e^{B^-} = \sum_\nu x'^\nu Y(t^\nu, \omega_2)/\nu!$. Now B^+ is multiplication by $\sum_k B_k t_k$ with $B_k = \sum_n x'_n \binom{k-1}{n-1} \omega_2^{k-n}$, and $A^- = \sum_k A_k \nabla_k$ with $A_k = \sum_j x_j \binom{-k-1}{j-1} \omega_1^{-k-j}$. By Taylor's formula e^{A^-} is the ring homomorphism that maps every t_k to $t_k + k A_k$, applied to the coefficient of each power of ω_2 . It maps $\sum_k B_k t_k$ to $\sum_k B_k t_k + \kappa$, so $e^{A^-} e^{B^+} = e^\kappa e^{B^+} e^{A^-}$ in $\mathcal{L}_+$, with

$$(50) \quad \kappa = \sum_k k A_k B_k = \sum_{j,n} x_j x'_n \sum_{k \geq 1} k \binom{-k-1}{j-1} \binom{k-1}{n-1} \omega_1^{-k-j} \omega_2^{k-n} = \sum_{j,n} x_j x'_n \Gamma_{j,n} \varepsilon_+^{j+n},$$

because the sum over k is the left side of (49) for $\varepsilon_+^2 = \partial_{\omega_2} \varepsilon_+ = \sum_k k \omega_1^{-k-1} \omega_2^{k-1}$. In the same way $e^{B^-} e^{A^+} = e^{\kappa'} e^{A^+} e^{B^-}$ in $\mathcal{L}_-$, with $\kappa' = \sum_{j,n} x_j x'_n \Gamma_{j,n} \varepsilon_-^{j+n}$, because $\varepsilon_-^2 = -\partial_{\omega_1} \varepsilon_- = \sum_k k \omega_2^{-k-1} \omega_1^{k-1}$. Multiplications commute with each other, and so do differentiations; and these identities may be used inside a product of four factors, because the sums over the monomials in between are finite. So, with $\mathcal{N} = e^{A^+} e^{B^+} e^{A^-} e^{B^-}$,

$$(51) \quad e^{A^+} e^{A^-} e^{B^+} e^{B^-} = e^\kappa \mathcal{N} \text{ in } \mathcal{L}_+, \quad e^{B^+} e^{B^-} e^{A^+} e^{A^-} = e^{\kappa'} \mathcal{N} \text{ in } \mathcal{L}_-.$$

One Laurent polynomial. Fix partitions σ, σ' of E and write $\langle X \rangle = [t^\sigma] X t^{\sigma'}$. Only the ∇_k with $|k| \leq E$ contribute to $\langle \mathcal{N} \rangle$, so the coefficient $n_{\mu'', \nu''}$ of $x^{\mu''} x'^{\nu''}$ in $\langle \mathcal{N} \rangle$ lies in $\mathcal{L}$; and $n_{\emptyset, \emptyset} = [\sigma = \sigma']$. The coefficient of $x^{\mu'} x'^{\nu'}$ in e^κ is $\Gamma_{\mu', \nu'} \varepsilon_+^{|\mu'|+|\nu'|}$ for a number $\Gamma_{\mu', \nu'}$, and in $e^{\kappa'}$ it is $\Gamma_{\mu', \nu'} \varepsilon_-^{|\mu'|+|\nu'|}$. Let $f_+ = \langle Y(a, \omega_1) Y(b, \omega_2) \rangle \in \mathcal{L}_+$ and $f_- = \langle Y(b, \omega_2) Y(a, \omega_1) \rangle \in \mathcal{L}_-$. Comparing the coefficients of $x^\mu x'^\nu$ in (51) gives $f_\pm = \mu! \nu! \sum \Gamma_{\mu', \nu'} \varepsilon_\pm^{|\mu'|+|\nu'|} n_{\mu'', \nu''}$, summed over all pairs of partitions μ', μ'' with union μ and ν', ν'' with union ν . Therefore $(\omega_1 - \omega_2)^H f_+$ and $(\omega_1 - \omega_2)^H f_-$ are the same element $F = \mu! \nu! \sum \Gamma_{\mu', \nu'} (\omega_1 - \omega_2)^{H-|\mu'|+|\nu'|} n_{\mu'', \nu''}$ of $\mathcal{L}$, and $F(1, 1) = \mu! \nu! \Gamma_{\mu, \nu} [\sigma = \sigma']$.

Reading off. By (O1), $f_+ = \sum_r \langle a[r] b[-r] \rangle \omega_1^{r-h'} \omega_2^{-r-h}$, with no terms for $r > E$, and $f_- = \sum_r \langle b[-r] a[r] \rangle \omega_1^{r-h'} \omega_2^{-r-h}$, with no terms for $r < -E$. Multiplying by $(\omega_1 - \omega_2)^H = \sum_i (-1)^i \binom{H}{i} \omega_1^{H-i} \omega_2^i$ shows that $F = \sum_s F_s \omega_1^s \omega_2^{-s}$, where $F_s = \sum_i (-1)^i \binom{H}{i} \langle a[s-h+i] b[h-s-i] \rangle = \langle Q_s(a, b) \rangle$, and also $F_s = \sum_i (-1)^i \binom{H}{i} \langle b[h-s-i] a[s-h+i] \rangle$. This is (a). For $s > E+h$ all terms of the first sum vanish, and for $s < -E-h$ all terms of the second; this is (b). And $\sum_s \langle Q_s(a, b) \rangle = F(1, 1) = \mu! \nu! \Gamma_{\mu, \nu} [\sigma = \sigma']$.

The constant. The number $\Gamma_{\mu,\nu}$ does not depend on E . For $E = 0$ we have $\langle \mathcal{N} \rangle = 1$, so $f_+ = \mu! \nu! \Gamma_{\mu,\nu} \varepsilon_+^H$, in which ω_2^0 has the coefficient $\mu! \nu! \Gamma_{\mu,\nu} \omega_1^{-H}$. In $f_+ = \sum_r \langle a[r] b[-r] \rangle \omega_1^{r-h'} \omega_2^{-r-h}$ it has the coefficient $a[-h] (b[h] 1) \omega_1^{-H}$, and $b[h] 1 = b$: in (47) only terms with all $k_i = -l_i$ negative act on 1, and $\binom{l_i-1}{\nu_i-1} = 0$ for $l_i < \nu_i$, so $\sum_i l_i = h$ forces $l_i = \nu_i$. Hence $\mu! \nu! \Gamma_{\mu,\nu} = a[-h] b$, which proves (c). ■

Lemma 10. *Let $y_n = \sum_{j=1}^n \binom{n}{j} t_j$, and for a partition μ of h let $a^\mu = \prod_i (-1)^{\mu_i-1} y_{\mu_i}$ and $b^\mu = (h!/\mathbf{z}_\mu) t^\mu$. Then $a^\mu[-h] b^\nu = h! [\mu = \nu]$ for all partitions μ, ν of h .*

Here $h!/\mathbf{z}_\mu$ is an integer, the number of permutations of $[h]$ whose cycle lengths are the parts of μ [Sta12, Proposition 1.3.2].

Proof. Let $\vartheta_n(k) = \sum_{j=1}^n \binom{n}{j} \binom{-k-1}{j-1}$. Multiplying out $\prod_i y_{\mu_i}$ and using (47) and (O3) for each monomial gives, for $b \in V_h$,

$$(52) \quad (\prod_i y_{\mu_i})[-h] b = \sum_{\sum_i k_i = h, k_i \geq 1} \prod_i \vartheta_{\mu_i}(k_i) \cdot \prod_i \nabla_{k_i} b.$$

By Vandermonde's identity $\vartheta_n(k) = \binom{n-k-1}{n-1}$, which is 0 for $1 \leq k < n$ and $(-1)^{n-1}$ for $k = n$. So a nonzero term has $k_i \geq \mu_i$ for all i , hence $k_i = \mu_i$, because both add up to h . Therefore $a^\mu[-h] b = \prod_i \nabla_{\mu_i} b$. For $b = t^\nu$ this is 0 if $\text{mult}_k(\mu) > \text{mult}_k(\nu)$ for some k ; otherwise $\mu = \nu$, because $|\mu| = |\nu|$, and it is $\mathbf{z}_\mu$. ■

Proof of Theorem B. Given $\lambda \geq 1$, let $h \geq 2$ be the least integer with $p(h) \geq 2^\lambda$, and let $\mu(\mathbf{x})$ be the partition of h with number $\mathbf{x} \in \{0, 1\}^\lambda$ (read as an integer, counting from 0) in the lexicographic list of the partitions of h , written as nonincreasing sequences. Let $d = h!$ and $\mathcal{I} = \{-2h, \dots, 2h\}$. The pieces are

$$(53) \quad A_s(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}) = [t^{\mu(\mathbf{u})}] Q_s(a^{\mu(\mathbf{x})}, b^{\mu(\mathbf{v})}) t^{\mu(\mathbf{y})} \quad (s \in \mathcal{I}).$$

Here $a^{\mu(\mathbf{x})}$ is a combination of monomials of weight at most h , as (48) requires. By Lemma 9(b), (c) with $E = h$ and Lemma 10, $\sum_s A_s = h! [\mathbf{x} = \mathbf{v}] \cdot [t^{\mu(\mathbf{u})}] t^{\mu(\mathbf{y})} = h! [\mathbf{x} = \mathbf{v}] [\mathbf{y} = \mathbf{u}]$.

Factorizations. Write $a = a^{\mu(\mathbf{x})}$, $b = b^{\mu(\mathbf{v})}$. In (48), $b[h-s-i] t^{\mu(\mathbf{y})}$ lies in V_{2h-s-i} ; with the factors exchanged, $a^{(i)}[s-h+i] t^{\mu(\mathbf{y})}$ lies in V_{s+i} . Expanding these polynomials in monomials t^σ gives

$$(54) \quad A_s = \sum_{i=0}^{2h} \sum_{|\sigma|=s+i} (-1)^i [t^\sigma] a^{(i)}[s-h+i] t^{\mu(\mathbf{y})} \cdot [t^{\mu(\mathbf{u})}] b[h-s-i] t^\sigma$$

$$(55) \quad = \sum_{i=0}^{2h} \sum_{|\sigma|=2h-s-i} (-1)^i [t^{\mu(\mathbf{u})}] a^{(i)}[s-h+i] t^\sigma \cdot [t^\sigma] b[h-s-i] t^{\mu(\mathbf{y})}.$$

All factors are integers by (O1). In (54) the first factor depends on $(\mathbf{x}, \mathbf{y})$ and the second on $(\mathbf{u}, \mathbf{v})$; in (55), on $(\mathbf{x}, \mathbf{u})$ and on $(\mathbf{y}, \mathbf{v})$. The numbers of terms are $R_s^{\text{row}} = \sum_{i=0}^{2h} p(s+i)$ and $R_s^{\text{col}} = \sum_{i=0}^{2h} p(2h-s-i)$, both at least 1. Number the pairs (i, σ) of each sum from 0, read θ as an integer, put the sign $(-1)^i$ into the first function, and let both functions be 0 if θ is not the number of a pair. This gives factorizations (Definition 1) through $c_s^{\text{row}} = \lceil \log R_s^{\text{row}} \rceil$ and $c_s^{\text{col}} = \lceil \log R_s^{\text{col}} \rceil$ bits.

Numbers of bits. Since $p(h-1) < 2^\lambda$, (46) gives $c_{\text{HR}} \sqrt{h-1} < \lambda + \log h + C_{\text{HR}}$. Hence $h = O(\lambda^2)$, and then $c_{\text{HR}} \sqrt{h} \leq c_{\text{HR}} \sqrt{h-1} + c_{\text{HR}} \leq \lambda + O(\log 2\lambda)$. As p is nondecreasing, $R_s^{\text{row}} \leq (2h+1)p(2h+s)$ and $R_s^{\text{col}} \leq (2h+1)p(2h-s)$. By (46) and $\sqrt{2h+s} + \sqrt{2h-s} \leq 2\sqrt{2h}$,

$$(56) \quad c_s^{\text{row}} + c_s^{\text{col}} \leq 2\sqrt{2} c_{\text{HR}} \sqrt{h} + 2 \log(2h+1) + 2C_{\text{HR}} + 2 \leq 2\sqrt{2} \lambda + O(\log 2\lambda).$$

Sizes and operations. List the partitions of $0, 1, 2, \dots$ until h is found, and then up to $4h$. Their number Δ is at most $(4h+1)p(4h)$, so $\log \Delta \leq 2c_{\text{HR}}\sqrt{h} + O(\log 2\lambda) = O(\lambda)$ by (46), and all partitions in (54) and (55) are among them. The list gives $\mathcal{I}$, d , the numbers $c_s^{\text{row}}, c_s^{\text{col}}$, the partition $\mu(\mathbf{x})$, and the pair (i, σ) with a given number θ . For a monomial a of weight at most h , its *table* consists of the integers $[t^{\sigma'}]a[|\sigma'| - |\sigma|]t^{\sigma}$ for all listed σ', σ ; the table of 1 is $[\sigma' = \sigma]$. Since $\nabla_k t^{\sigma} = k \text{mult}_k(\sigma) t^{\sigma}/t_k$, and the coefficient of $t^{\sigma'}$ in $t_l f$ is that of $t^{\sigma'}/t_l$ in f , by (O2) each entry of the table of $t_n a'$ is a sum of at most $8h$ entries of the table of a' , multiplied by integers of absolute value at most $4h \cdot 2^{5h}$. A monomial of weight at most h has at most h factors. So all entries of all tables are at most $(32h^2 2^{5h})^h \leq 2^{12h^2}$ in absolute value, and the tables, of which there are at most Δ , are computed with $\Delta^{O(1)}$ operations. Each factor in (54) and (55) is a combination of entries with the coefficients of $a^{(i)}$ or b . The coefficients of a^{μ} are found by multiplying in one y_{μ_i} at a time and collecting equal monomials after each step, of which there are never more than Δ . The absolute values of the coefficients add up to at most $2^{2h} \prod_i 2^{\mu_i} = 2^{3h}$ for $a^{(i)}$, and to at most $h!$ for b . And A_s is a sum of at most Δ products of two factors. Hence all these numbers are $2^{O(h^2)} = 2^{O(\lambda^4)}$ in absolute value, and each is computed with $\Delta^{O(1)} = 2^{O(\lambda)}$ operations. So Definition 7 holds with $\rho = 1/\sqrt{2}$ if c is large enough. ■

APPENDIX C. COUNTING SATISFYING ASSIGNMENTS AS A FUNCTION OF n (PROOF SKETCH)

Theorem A(d) only decides, because the formulas produced by the sparsification in its proof may share satisfying assignments. Dell et al. [DHMTW14, Appendix A] change the sparsification algorithm so that they share none (Lemma 4). Their Lemma A.1 gives no explicit bound on the number of clauses, so we make the analogous change in the algorithm of Calabro [Cal09, Figure 5.1 and Theorem 5.15], which the proof of Theorem A(d) uses.

Theorem C. *One deterministic algorithm computes the number of satisfying assignments of a 3-CNF on $n \geq 2$ variables in time $2^{O(n/(\log n)^{\alpha/10})}$.*

Proof sketch. First delete every clause that contains a variable together with its negation; this does not change the satisfying assignments. Given a 3-CNF ψ and $\varepsilon \in (0, 1]$, the algorithm of Calabro [Cal09, Figure 5.1] deletes every clause that properly contains another clause, which does not change the satisfying assignments either, and then either outputs ψ , or selects a nonempty set H of literals and a set $F \subseteq \psi$ of clauses that properly contain H and calls itself on $\psi \cup \{H\}$ and on $\psi \cup Q$, where $Q = \{E \setminus H : E \in F\}$. We carry along a set $\mathcal{N}$ of clauses with one literal each, empty at the start, which the algorithm never reads: the first call receives $\mathcal{N}$, the second $\mathcal{N} \cup \{\{\neg\omega\} : \omega \in H\}$, where $\neg\omega$ is the negation of the literal ω ; and where the algorithm outputs ψ we output $\psi \cup \mathcal{N}$. An assignment of the n variables that satisfies $\psi \cup \mathcal{N}$ satisfies $\psi \cup \{H\} \cup \mathcal{N}$ if it makes some literal of H true; otherwise it satisfies $\psi \cup Q \cup \mathcal{N} \cup \{\{\neg\omega\} : \omega \in H\}$, because each $E \in F$ has a true literal, which lies in $E \setminus H$. No assignment satisfies both, since the second contains $\{\neg\omega\}$ for every $\omega \in H$ and the first contains the clause H . Conversely, each of the two contains $\psi \cup \mathcal{N}$, so an assignment that satisfies either of them satisfies $\psi \cup \mathcal{N}$. By induction over the recursion, each satisfying assignment of the input satisfies exactly one output, and no other assignment satisfies any. Since no choice depends on $\mathcal{N}$, the tree of recursive calls and the formula ψ at each leaf are those of Calabro's algorithm [Cal09]: there are at most $2^{\varepsilon n}$ outputs, produced in time $2^{\varepsilon n}$ times a polynomial in n that does not depend on ε , and each has at most $(M(\varepsilon) + 2)n$ clauses, as there are only $2n$ clauses with one literal. If an output Φ has $m \geq 2$ clauses, Theorem A(c), applied after the v variables that occur in Φ have been renamed to $x_1, \dots, x_v$, computes the number Z of §6, and Φ has $Z \cdot 2^{n-v}$ satisfying assignments; with no clause it has 2^n , and with one clause of j literals $(2^j - 1)2^{n-j}$. Sum over the outputs. The time is bounded as in the proof of Theorem A(d), with $M(\varepsilon) + 2$ in place of $M(\varepsilon)$, and $\varepsilon = (\log n)^{-\alpha/10}$ gives the theorem. ■

Dell et al. [DHMTW14] add the clauses $\{\neg\omega\}$ to ψ itself, where the next reduction deletes the other clauses that contain $\neg\omega$, so the recursion tree can change; keeping them in $\mathcal{N}$ leaves Calabro's analysis [Cal09] untouched. For clauses of 3 literals, a bound in Calabro's proof [Cal09, (5.4), in the proof of Theorem 5.15] gives $M(\varepsilon) = O(\varepsilon^{-2} \log^2(1/\varepsilon))$, and with it the time becomes $2^{O(n(\log \log n)^{2/3}/(\log n)^{\alpha/3})}$.

APPENDIX D. COUNTING SATISFYING ASSIGNMENTS: THE BOUND IN n

In Theorems A(d) and C the exponent of $\log n$ is only a tenth of the exponent of $\log m$ in Theorem A(c), because the total U of the network of §6 grows with the number of clauses, and sparsification with a small ε leaves $O(\varepsilon^{-9}n)$ clauses. Here a different network has $U = O(kn \log \Delta)$ when every variable occurs in at most Δ clauses, so the number of clauses left by sparsification enters only through $\log \Delta = O(k \log(k/\varepsilon))$.

Theorem D. (a) For a constant $C_0 \geq 1$ and all integers $\ell \geq 0$ let $K_\ell \geq 2$ be integers and e_ℓ reals with $K_\ell < K_{\ell+1} \leq K_\ell^{C_0}$ and $(\log K_\ell)^{-C_0} \leq e_\ell \leq 1$. Suppose that K_ℓ and a number in $[e_\ell, 2e_\ell]$ can be computed from ℓ in time polynomial in K_ℓ , and that one deterministic algorithm, given $\ell \geq 0$, $\lambda, B \geq 1$ and query access to tables with $|W_w| < 2^B$, outputs the value of the (K_ℓ, λ) -clique within

$$(57) \quad 2^{e_\ell K_\ell^2 \lambda} \cdot (K_\ell(\lambda + B))^{C_0 K_\ell^2} \text{ operations.}$$

Then one deterministic algorithm computes, for all integers n and k with $n \geq K_0^{17}$ and $3 \leq k \leq (\log n)^{1/4}$, the number of satisfying assignments of a k -CNF on n variables in time $2^{O(k^2 e n \log(k/e))}$, where $e = e_\ell$ for the largest ℓ with $K_\ell \leq n^{1/17}$. In particular, if $e_\ell = O((\log K_\ell)^{-\alpha'})$ for a constant $\alpha' > 0$, the time is $2^{O(k^2 n \log \log n / (\log n)^{\alpha'})}$.

(b) The pairs (K_ℓ, e_ℓ) of (40), for the pieces of Appendix B, satisfy this hypothesis, and $e_\ell = \Theta((\log K_\ell)^{-1/2})$. So there are a deterministic algorithm and an absolute constant C such that, for all integers $k \geq 1$ and $n \geq 4$, the algorithm computes the number of satisfying assignments of a k -CNF on n variables in time $2^{C k^2 n \log \log n / \sqrt{\log n}}$.

The constants in the O in (a) depend only on those of the hypothesis, not on k .

Part (b) gives the bound in n of the Main Theorem. The bound (57) is that of Definition 4 with C made explicit, for one algorithm; that each pair is achievable is not enough, because K grows with n . Step 5 of the proof below takes from Appendix C, which is a proof sketch, the change to the sparsification algorithm and the argument that every satisfying assignment satisfies exactly one output; the other steps are proved in full.

Corollary 11. There are an absolute constant C and a deterministic algorithm that, given a k -CNF, for any $k \geq 1$, with $m \geq 2$ clauses in which $n \geq 4$ variables occur, computes the number of its satisfying assignments, as assignments of these n variables, in time at most

$$(58) \quad \min\{2^{C \cdot km / \sqrt{\log m}}, 2^{C \cdot k^2 n \log \log n / \sqrt{\log n}}\}.$$

Proof. Run the algorithm of Theorem A(c), with the pieces of Appendix B, and the algorithm of Theorem D(b) in alternation, one operation of each at a time, and output the result of the one that stops first. Both compute the same number. This multiplies the number of operations of the faster one by a constant; as $km/\sqrt{\log m}$ and $n \log \log n / \sqrt{\log n}$ are at least 2, this constant goes into C . Computing the two exponents up to a factor 2 and running only the algorithm with the smaller one gives the same bound. ■

This is the Main Theorem, with the ranges of k , m and n made explicit. For a formula on $N \geq n$ variables of which n occur, renaming these to $x_1, \dots, x_n$ takes time polynomial in the

length of the formula, and the number of satisfying assignments of all N variables is 2^{N-n} times the number computed here.

Idea. Let $V(\xi)$ be the set of clauses that the assignment ξ violates; we want the number of ξ with $V(\xi) = \emptyset$. Inclusion–exclusion writes $[V(\xi) = \emptyset]$ as the sum of $(-1)^{|\Pi|}$ over all sets $\Pi \subseteq V(\xi)$ of *marked* clauses. If the nodes of a network hold the variables, they then have to share only the marks, at about $\log \Delta$ bits each. We cut inclusion–exclusion off at $\bar{n} \approx n/\log \Delta$ marks, which cost $O(n)$ bits. This changes nothing for the ξ with $|V(\xi)| \leq \bar{n}$. The ξ with $|V(\xi)| > \bar{n}$ are killed by parities: $r = O(n)$ sparse sets Λ_j of clauses (*checks*) such that each such $V(\xi)$ meets some Λ_j in an odd number of clauses. In the expansion $[q \text{ even}] = \sum_{i=0}^q \sigma_i \binom{q}{i}$ the coefficient σ_i is $\pm 2^{i-1}$ for $i \geq 1$, so modulo 2^{n+1} all terms with more than $n+r$ marks *in all checks together* can be dropped; the $O(n)$ marks that remain cost $O(\log \Delta)$ bits each. The checks are found one at a time, by exhaustive search in an explicit family of polynomial size in which every set V of more than $\bar{n}$ clauses has odd intersection with at least a quarter of the members. The integer $X = \sum_{\xi \in \{0,1\}^n} \binom{|V(\xi)|}{\bar{n}+1} [\text{all parities so far are even}]$, which a network of the same kind computes, stays below $(\frac{3}{4})^r 2^{O(n)}$ after r checks, and $X = 0$ certifies the checks.

Setting and parameters. In Steps 1–4, $3 \leq k \leq (\log n)^{1/4}$, and Φ is a k -CNF with $m \geq 1$ clauses on n variables in which every variable occurs in at most Δ clauses, where $2 \leq \Delta \leq (\log n)^{4kC_0}$ for the constant C_0 of Theorem D(a); so $m \leq \Delta n$. Assignments $\xi \in \{0,1\}^n$ are assignments of all n variables; ξ *violates* a clause if it makes all its literals false, and $V(\xi) \subseteq \Phi$ is the set of clauses that ξ violates. For integers $t \geq 0$, $0 \leq \mu \leq \mu'$ and $\mu' \geq 1$,

$$(59) \quad \log \binom{t}{\mu} \leq \mu' \max\{1, \log(3t/\mu')\},$$

since $\binom{t}{\mu} \leq (3t/\mu')^{\mu'}$ if $\mu' \leq t$, and $\binom{t}{\mu} \leq 2^t \leq 2^{\mu'}$ otherwise. Let $n \geq 2 \log(2\Delta)$, let $\bar{n} \geq 2$ be the power of two with $n/(2 \log(2\Delta)) < \bar{n} \leq n/\log(2\Delta)$, and, if $m > \bar{n}$, let

$$(60) \quad \begin{aligned} \beta &= n + \lceil \log \binom{m}{\bar{n}+1} \rceil + 1, & \bar{r} &= \min\{j \geq 0 : 3^j 2^\beta < 4^j\}, & M &= \bar{r} + \beta - 1, \\ \bar{s} &= 16m/\bar{n}, & p &= 16/(15\bar{n}), & \bar{t} &= \min\{c \geq 0 : 2^c \geq m(1+p)^{\bar{r}}\}. \end{aligned}$$

Here $2^{\beta-1}$ will bound X , $\bar{r}$ the number of checks, M the number of their marks, and $\bar{s}$ the size of a check; p will bound the fraction of the candidate checks that contain a given clause, and $\bar{t}+2$ the number of chosen checks that contain it. Since $m/\bar{n} < 2\Delta \log(2\Delta)$, we have $\bar{s} < 32\Delta \log(2\Delta)$ and $3m/(\bar{n}+1) < 6\Delta \log(2\Delta) \leq (2\Delta)^3$, so by (59) $\beta \leq n + 3(\bar{n}+1) \log(2\Delta) + 2 \leq 4n + 3 \log(2\Delta) + 2$; and $\bar{r} \leq 2.41\beta + 1$ because $2.41 \log \frac{4}{3} > 1$. The constants in $O(\cdot)$ and $o(\cdot)$ below do not depend on k . As $\log \Delta \leq 4kC_0 \log \log n \leq 4C_0(\log n)^{1/4} \log \log n$, there is an n_0 that depends only on C_0 such that for $n \geq n_0$, which we assume from now on, $\Delta \leq n^{1/(40C_0)}$, $n \geq 2 \log(2\Delta)$ and

$$(61) \quad \bar{r} \leq 10n, \quad M \leq 14n.$$

Then $p\bar{r} < 22 \log(2\Delta)$, and with $\log(1+p) \leq \frac{3}{2}p$ also $\bar{t} \leq \log m + 33 \log(2\Delta) + 1$.

Step 1: parity checks, and a cut-off modulo a power of 2. A *check* is a set of clauses of Φ . For a list $\mathbf{\Lambda} = (\Lambda_1, \dots, \Lambda_r)$ of checks, which may overlap or repeat, let $V_j(\xi) = \Lambda_j \cap V(\xi)$, let $\eta(\xi)$ be the sum of $(-1)^{|\Pi|}$ over the sets $\Pi \subseteq V(\xi)$ with $|\Pi| \leq \bar{n}$, and

$$(62) \quad F(\mathbf{\Lambda}) = \sum_{\xi \in \{0,1\}^n} \eta(\xi) \prod_{j=1}^r [|V_j(\xi)| \text{ even}], \quad X(\mathbf{\Lambda}) = \sum_{\xi \in \{0,1\}^n} \binom{|V(\xi)|}{\bar{n}+1} \prod_{j=1}^r [|V_j(\xi)| \text{ even}].$$

$$(63) \quad \text{If } X(\mathbf{\Lambda}) = 0, \text{ then } F(\mathbf{\Lambda}) \text{ is the number of satisfying assignments of } \Phi.$$

Indeed, the terms of $X(\mathbf{\Lambda})$ are nonnegative; so for every ξ with $|V(\xi)| > \bar{n}$ the product over j is 0, and it does not matter that $\eta(\xi)$ is then an uncontrolled integer. If $1 \leq |V(\xi)| \leq \bar{n}$, then

$\eta(\xi)$ runs over all subsets of $V(\xi)$ and is 0. If $V(\xi) = \emptyset$, all factors are 1. If $m \leq \bar{n}$, the empty list has $X = 0$, and Step 2 is skipped.

Let $\sigma_0 = 1$ and $\sigma_i = (-1)^i 2^{i-1}$ for $i \geq 1$; then $[q \text{ even}] = \frac{1}{2}(1 + (1 - 2)^q) = \sum_{i=0}^q \sigma_i \binom{q}{i}$ for all integers $q \geq 0$.

Lemma 12 (Cut-off). *Let Λ be a list of r checks, $\varphi : \{0, 1\}^n \rightarrow \mathbb{Z}$ a function, and $\nu \geq 1$ and $M' \geq \nu - 1 + r$ integers. Then*

$$(64) \quad \sum_{\xi \in \{0,1\}^n} \varphi(\xi) \prod_{j=1}^r [|V_j(\xi)| \text{ even}] \equiv \sum_{\xi \in \{0,1\}^n} \varphi(\xi) \sum_{(\Pi_1, \dots, \Pi_r)} \prod_{j=1}^r \sigma_{|\Pi_j|} \pmod{2^\nu},$$

where the inner sum is over the tuples with $\Pi_j \subseteq V_j(\xi)$ for all j and $\sum_{j=1}^r |\Pi_j| \leq M'$.

Proof. Without the last condition the two sides are equal, since $\binom{|V_j(\xi)|}{i}$ counts the sets $\Pi_j \subseteq V_j(\xi)$ of size i . The term of a tuple is $\pm \varphi(\xi) 2^y$ with $y = \sum_{j=1}^r \max\{|\Pi_j| - 1, 0\} \geq \sum_{j=1}^r |\Pi_j| - r$, and $y \geq \nu$ for an omitted tuple. ■

In Lemma 12, for a list of $r \leq \bar{r}$ checks, take $M' = M$, and $(\varphi, \nu) = (\binom{|V|}{\bar{n}+1}, \beta)$ or $(\varphi, \nu) = (\eta, n+1)$, and write $\varphi(\xi)$ as a sum over sets $\Pi_0 \subseteq V(\xi)$:

$$(65) \quad \begin{aligned} X(\Lambda) &\equiv \sum_{\xi \in \{0,1\}^n} \sum_{(\Pi_0, \dots, \Pi_r): |\Pi_0| = \bar{n}+1} \prod_{j=1}^r \sigma_{|\Pi_j|} \pmod{2^\beta}, \\ F(\Lambda) &\equiv \sum_{\xi \in \{0,1\}^n} \sum_{(\Pi_0, \dots, \Pi_r): |\Pi_0| \leq \bar{n}} (-1)^{|\Pi_0|} \prod_{j=1}^r \sigma_{|\Pi_j|} \pmod{2^{n+1}}, \end{aligned}$$

where the tuples have $\Pi_0 \subseteq V(\xi)$, $\Pi_j \subseteq V_j(\xi)$ for $1 \leq j \leq r$, and $|\Pi_1| + \dots + |\Pi_r| \leq M$. The residues determine the numbers: $0 \leq X(\Lambda) \leq 2^n \binom{m}{\bar{n}+1} \leq 2^{\beta-1}$, and $0 \leq F(\Lambda) \leq 2^n$ if $X(\Lambda) = 0$.

Step 2: finding the checks. Let $m > \bar{n}$.

Lemma 13 (Family of checks). *Let v be a power of two with $2 \leq v \leq m$. In time polynomial in m one can list sets $\Gamma_u \subseteq \Phi$, $u \in \Omega$, with $|\Omega| < (32m)^4$, such that (a) $|\Gamma_u| \leq 16m/v$ for all $u \in \Omega$; (b) every clause lies in Γ_u for at most $\frac{16}{15v}|\Omega|$ indices u ; (c) for every set $V \subseteq \Phi$ of more than v clauses, $|\Gamma_u \cap V|$ is odd for at least $\frac{1}{4}|\Omega|$ indices u .*

Proof. We subsample pairwise independently at rate $2/v$ and then take a small-bias parity. Let $\mathbb{F} = \text{GF}(2^s)$ with $16m \leq 2^s < 32m$ (an irreducible polynomial of degree s is found by search), with elements written as s -bit strings, and let $\langle \cdot, \cdot \rangle$ be the inner product of such strings modulo 2. Give the clauses $E \in \Phi$ distinct elements $\vartheta_E \in \mathbb{F}$ and distinct numbers $\iota(E) \in \{0, \dots, m-1\}$. For $u = (u_1, u_2, u_3, u_4) \in \mathbb{F}^4$ let Γ_u consist of the clauses E such that the first $\log v - 1$ bits of $u_1 \vartheta_E + u_2$ are 0 (event H_E) and $\langle u_3^{\iota(E)}, u_4 \rangle = 1$, where $u_3^0 = 1$ also for $u_3 = 0$. Let u be uniformly distributed in $\mathbb{F}^4$. For distinct $\vartheta, \vartheta' \in \mathbb{F}$ the pair $(u_1 \vartheta + u_2, u_1 \vartheta' + u_2)$ is uniformly distributed in $\mathbb{F}^2$ [Vad12, §3.5], so the events H_E have probability $2/v$ and are pairwise independent. Hence for $|V| > v$ the number Y of $E \in V$ with H_E has a mean $y > 2$ and variance at most y , and by the Cauchy–Schwarz inequality $\Pr(Y > 0) \geq (\mathbb{E}Y)^2 / \mathbb{E}(Y^2) \geq y^2 / (y^2 + y) > \frac{2}{3}$. For every nonempty set V' of clauses, $\sum_{E \in V'} \langle u_3^{\iota(E)}, u_4 \rangle$ is odd with probability at least $\frac{1}{2}(1 - \frac{m-1}{2^s}) \geq \frac{15}{32}$: the sum is $\langle w, u_4 \rangle$ modulo 2 with $w = \sum_{E \in V'} u_3^{\iota(E)}$, a nonzero polynomial of degree at most $m-1$ in u_3 , so $w = 0$ for at most $m-1$ values of u_3 , and for $w \neq 0$ the probability is $\frac{1}{2}$. This is the powering construction of small-bias sample spaces [AGHP92] (see also [NN93]). As (u_3, u_4) is independent of (u_1, u_2) , $|\Gamma_u \cap V|$ is odd with probability at least $\frac{2}{3} \cdot \frac{15}{32} = \frac{5}{16}$. A clause lies in

Γ_u with probability at most $\frac{1}{v}$, so $|\Gamma_u| > 16m/v$ with probability at most $\frac{1}{16}$. Let Ω be the set of the other u ; then $|\Omega| \geq \frac{15}{16}|\mathbb{F}|^4$, which gives (b), and (c) holds because $\frac{5}{16} - \frac{1}{16} = \frac{1}{4}$. ■

Take $v = \bar{n}$ in Lemma 13; then part (a) says $|\Gamma_u| \leq \bar{s}$, and the bound in part (b) is $p|\Omega|$. For a list $\mathbf{\Lambda}$ of $r \leq \bar{r}$ checks let $\text{load}(E)$ be the number of its checks that contain the clause E , let $(\mathbf{\Lambda}, \Gamma)$ be $\mathbf{\Lambda}$ with the check Γ appended, and

$$(66) \quad \Upsilon(\mathbf{\Lambda}) = \left(\frac{4}{3}\right)^r \frac{X(\mathbf{\Lambda})}{2^{\beta-1}} + (1+p)^{\bar{r}-r} 2^{-\bar{t}} \sum_{E \in \Phi} 2^{\text{load}(E)}.$$

Algorithm: start with the empty list; while $X(\mathbf{\Lambda}) > 0$, replace $\mathbf{\Lambda}$ by the list $(\mathbf{\Lambda}, \Gamma_u)$, $u \in \Omega$, with the smallest Υ ; then output $F(\mathbf{\Lambda})$. Every list that is reached has $\Upsilon(\mathbf{\Lambda}) \leq 2$. For the empty list both terms are at most 1, by $X \leq 2^{\beta-1}$ and the choice of $\bar{t}$. If $\mathbf{\Lambda}$ is reached and $X(\mathbf{\Lambda}) > 0$: a nonzero term ξ of $X(\mathbf{\Lambda})$ has $|V(\xi)| > \bar{n}$, and appending Γ_u multiplies it by $[|\Gamma_u \cap V(\xi)| \text{ even}]$, so by part (c) the average of $X(\mathbf{\Lambda}, \Gamma_u)$ over $u \in \Omega$ is at most $\frac{3}{4}X(\mathbf{\Lambda})$; by part (b) the average of $2^{[E \in \Gamma_u]}$ is at most $1+p$. So the average of $\Upsilon(\mathbf{\Lambda}, \Gamma_u)$ is at most $\Upsilon(\mathbf{\Lambda})$, and so is the minimum. Hence $X(\mathbf{\Lambda}) \leq (\frac{3}{4})^r 2^\beta$, which for $r = \bar{r}$ is below 1: *the loop ends after at most $\bar{r}$ rounds with $X(\mathbf{\Lambda}) = 0$, and the output is right by (63).* Also $2^{\text{load}(E)-\bar{t}} \leq 2$ for every clause E . So every list whose X or F is needed (a reached list, possibly with one more Γ_u) has

$$(67) \quad r \leq \bar{r} \text{ checks,} \quad |\Lambda_j| \leq \bar{s} \text{ for all } j, \quad \text{load}(E) \leq \bar{t} + 2 \text{ for all } E \in \Phi,$$

and there are at most $\bar{r}(32m)^4 + 2$ such lists. The number Υ is a rational with polynomially many bits, so the comparisons are exact.

Step 3: a network with two kinds of marks. We compute the right sides of (65) for a list with (67). Let $2 \leq g \leq n$ (in Step 4, $g = 3K^8$). Distribute the variables over the nodes $a \in [g]$, at most $\lceil n/g \rceil$ to each, and give check j the *home* $h(j) \in [g]$ with $h(j) \equiv j \pmod{g}$. *Objects* are the clauses (kind 0) and the *incidences* (E, j) with $E \in \Lambda_j$ (kind 1); there are $I \leq r\bar{s}$ incidences. A clause *touches* the nodes that hold its variables; (E, j) touches these nodes and $h(j)$. If an object touches $a_1 < \dots < a_t$, its *owner* is a_1 and its *tests* are the pairs $\{a_i, a_{i+1}\}$, $1 \leq i < t$; an object of kind $\kappa \in \{0, 1\}$ has at most $k - 1 + \kappa$ tests. A setting z_a has a value bit for each variable at a and a *mark* bit for each object that touches a . Let $W_a(z_a) = 0$ if a marked object has a true literal at a ; otherwise $W_a(z_a) = \prod_{j: h(j)=a} \sigma_{q_j}$, where q_j is the number of marked incidences of check j , times, for F only, $(-1)^q$, where q is the number of marked clauses owned by a . Let $\Psi_{a,b}(z_a, z_b) = 1$ if z_a and z_b give the same mark to every object with the test $\{a, b\}$, and 0 otherwise. In a nonzero term the marks of an object agree at all nodes that it touches, because its tests form a path. So the term is given by ξ and the sets Π_0 and Π_j of the clauses E for which E , respectively (E, j) , is marked; all literals of a marked clause are false, so $\Pi_0 \subseteq V(\xi)$ and $\Pi_j \subseteq V_j(\xi)$; and $h(j)$ sees $|\Pi_j|$. Hence the nonzero terms are the tuples $(\xi, \Pi_0, \dots, \Pi_r)$, each once, with the values in (65).

Profiles. A *profile* consists of integers $\mu_{a,\kappa} \geq 0$ and $\mu_{a,b,\kappa} = \mu_{b,a,\kappa} \geq 0$ for $a \neq b$ in $[g]$ and $\kappa \in \{0, 1\}$. Its network has S_a restricted to the settings in which exactly $\mu_{a,\kappa}$ objects of kind κ owned by a , and for each $b \neq a$ exactly $\mu_{a,b,\kappa}$ of the $t_{a,b,\kappa}$ objects of kind κ with the test $\{a, b\}$, are marked. Every nonzero term belongs to exactly one profile, which has $\sum_{a \in [g]} \mu_{a,0} = |\Pi_0|$, $\sum_{a \in [g]} \mu_{a,1} = \sum_{j=1}^r |\Pi_j|$, $\mu_{a,b,\kappa} \leq t_{a,b,\kappa}$ and $\sum_{\{a,b\} \subseteq [g]} \mu_{a,b,\kappa} \leq (k-1+\kappa) \sum_{a \in [g]} \mu_{a,\kappa}$. Call a profile with the last two properties *admissible* if $\sum_{a \in [g]} \mu_{a,0} = \bar{n} + 1$ (for F : $\leq \bar{n}$) and $\sum_{a \in [g]} \mu_{a,1} \leq M$. Each right side of (65) is the sum of the values of the networks of the admissible profiles, of which there are at most $(I + m + 1)^{2(g^2+g)}$.

The total. Within a profile, replace the string of the marks of the $t_{a,b,\kappa}$ objects, in a fixed order, by its position in the list of all strings of this length with $\mu_{a,b,\kappa}$ ones. By observation (b) after Definition 1, $\Psi_{a,b}$ factors through $c_{a,b} = \sum_{\kappa \in \{0,1\}} \lceil \log \binom{t_{a,b,\kappa}}{\mu_{a,b,\kappa}} \rceil$ bits. A product of binomial

coefficients is at most the binomial coefficient of the sums, so $\sum_{\{a,b\} \subseteq [g]} \log \binom{t_{a,b,\kappa}}{\mu_{a,b,\kappa}} \leq \log \binom{T_\kappa}{\mu_\kappa}$, where $\mu_\kappa = \sum_{\{a,b\} \subseteq [g]} \mu_{a,b,\kappa}$ and $T_\kappa = \sum_{\{a,b\} \subseteq [g]} t_{a,b,\kappa}$ is the number of tests of the objects of kind κ , so $T_0 \leq (k-1)m$ and $T_1 \leq kr\bar{s}$. By (59) with $\mu' = (k-1)(\bar{n}+1)$, kind 0 costs at most $(k-1)(\bar{n}+1) \cdot 3 \log(2\Delta) \leq 3(k-1)(n + \log(2\Delta))$ bits; this is why $\bar{n} \approx n/\log \Delta$. With $\mu' = kM$ and $3T_1/\mu' \leq 3\bar{s}$ (as $r \leq M$), kind 1 costs at most $kM \log(3\bar{s})$ bits. Adding two roundings per pair, by (61) the total $U = 2 \sum_{\{a,b\} \subseteq [g]} c_{a,b}$ satisfies

$$(68) \quad U \leq 6k(n + \log(2\Delta)) + 28kn \log(96\Delta \log(2\Delta)) + 2g^2 = O(kn \log \Delta) + 2g^2.$$

The parameters. At most $\Delta \lceil n/g \rceil$ clauses have a variable at a ; each gives z_a one mark of kind 0 and $\text{load}(E) \leq \bar{t} + 2$ marks of kind 1; the checks with home a give at most $\lceil \bar{r}/g \rceil \bar{s}$ more. So z_a has at most $P_0 = \lceil n/g \rceil (1 + \Delta(\bar{t} + 3)) + \lceil \bar{r}/g \rceil \bar{s}$ bits. An object has at most two tests at a , and $\lceil \log \binom{t}{\mu} \rceil \leq t$, so condition (i) of Definition 2 holds with $P = \lceil 2P_0 \rceil$, and

$$(69) \quad gP \leq 2(n + g)(1 + \Delta(\bar{t} + 3)) + 2(10n + g)\bar{s} + g = O(n\Delta(\log \Delta + \log n)).$$

This is what the second term of Υ is for: otherwise one clause could lie in all checks, and $P \geq r$ is more than the term $2gP/K$ in (70) below can bear. A marked incidence that touches a is owned by a or has a test at a , so a setting of an admissible profile has at most $(k+1)M$ of them; as $|\sigma_i| \leq 2^i$, condition (ii) holds with $B = (k+1)M + 1$. Counting marks, multiplying powers of 2 and finding the position of a string among those with as many ones take $\tau = n^{O(1)}$ operations, as condition (iii) requires. If $m \leq \bar{n}$, the list is empty and there are only clauses: then $F(\Lambda)$ equals the right side of (65), every test costs at most one bit, so $U \leq 2(k-1)\bar{n} \leq 2kn$, and $P = 2\lceil n/g \rceil (1 + \Delta)$ and $B = 1$ will do.

Step 4: cost. Assume the hypothesis of Theorem D(a). Let $n \geq \max\{K_0^{17}, n_0\}$, let $K = K_\ell$ and $e = e_\ell$ for the largest ℓ with $K_\ell \leq n^{1/17}$, and let $g = 3K^8$. Then $K^{C_0} \geq K_{\ell+1} > n^{1/17}$, so $n^{1/(17C_0)} < K \leq n^{1/17}$ and $e \geq (\log n)^{-C_0}$; moreover $g^2 \leq 9n^{16/17}$ and $g \leq n$, and K and a number in $[e, 2e]$ are found in time polynomial in n . Take a network on $[g]$ with parameters P, B, τ and total U , and let $L = \lceil (U + gP/K)/K^2 \rceil$ and $B' = \lceil 2gP/K \rceil + g^2B$. By case (i) of Lemma 8, with $\mathcal{T}$ taken from (57) for L, B' in the roles of λ, B , and by $K^2L \leq U + gP/K + K^2$, its value is computed with a number of operations whose logarithm is at most

$$(70) \quad e(U + gP/K + K^2) + 2gP/K + C_0K^2 \log(K(L + B')) + g \log K + O(\log(gPB\tau)).$$

For the networks of Step 3, B grows with n ; this is legitimate because (57) covers all integer tables. The numbers $L \leq gP + 1$ and B' are polynomial in n , so the third term of (70) is $O(K^2 \log n)$ and the last is $O(\log n)$; by (69) and $\Delta \leq n^{1/(40C_0)}$, $gP/K = O(n^{1-1/(17C_0)+1/(40C_0)} \log n)$; and $eg^2, eK^2, g \log K$ and $K^2 \log n$ are $O(n^{16/17} \log n)$. All these are $o(en)$. So by (68) one network costs $2^{O(ekn \log \Delta)}$ operations. There are $2^{O(n^{16/17} \log n)}$ profiles per sum and polynomially many sums; the values of the networks are computed as exact integers, and only their sums are reduced modulo 2^β or 2^{n+1} . Hence the number of satisfying assignments of Φ is computed in time $2^{O(ekn \log \Delta)}$.

Step 5: sparsification. The change made in Appendix C to Calabro's sparsification algorithm [Cal09, Figure 5.1] uses nothing about clauses of 3 literals. Applied to a k -CNF on n variables and $\varepsilon \in (0, 1]$, the changed algorithm produces at most $2^{\varepsilon n}$ outputs in time $2^{\varepsilon n} n^{O(k)}$ with an absolute constant in the O : its tree of recursive calls, which is that of the unchanged algorithm, is a full binary tree with at most $2^{\varepsilon n}$ leaves [Cal09, Lemma 5.13], so it has fewer than $2^{\varepsilon n+1}$ nodes; the formula ψ at a node, without $\mathcal{N}$, has no more clauses than the input [Cal09, Lemma 5.3], that is, fewer than $(2n+1)^k \leq n^{2k}$; and the work at a node (the reduction, the choice of H among the at most 2^k subsets of each clause, and carrying $\mathcal{N}$, which has at most $2n$ clauses) is polynomial in n , 2^k and the number of clauses, with an absolute exponent, so at most $n^{O(k)}$, as $2^k \leq n^k$. Each satisfying assignment of the input satisfies exactly one

output, and no other assignment satisfies any. Each output is $\psi \cup \mathcal{N}$, where ψ is an output of the unchanged algorithm, in which every literal occurs at most $2 + 4\alpha(32\alpha)^{k-2}$ times, where $\alpha = 2(k-1)^2\epsilon^{-1} \log(32(k-1)^2/\epsilon)$ [Cal09, (5.4), from which Theorem 5.15, item 3(a), is derived], and every literal occurs at most once in $\mathcal{N}$. With $x = 32(k-1)^2/\epsilon$ we have $32\alpha = 2x \log x \leq 2x^{3/2} \leq (8k/\epsilon)^3$, so $2 + 4\alpha(32\alpha)^{k-2} = 2 + \frac{1}{8}(32\alpha)^{k-1} \leq (8k/\epsilon)^{3k}$, and a literal occurs at most $(8k/\epsilon)^{3k}$ times in ψ . So every variable occurs in at most $2(8k/\epsilon)^{3k} + 2$ clauses of each output. The algorithm takes for k the largest number of literals in a clause of its input, or 3 if that is larger; the bound increases with k . Take $\epsilon \leq 1$ between e and $2e$, for the e of Step 4. An output without clauses counts 2^n . Every other output is a formula Φ as above, with Δ the larger of 2 and the largest number of its clauses in which one variable occurs; so $\log \Delta \leq 3k \log(8k/e) + 2 = O(k \log(k/e))$. As $e \geq (\log n)^{-C_0}$ and $k \leq (\log n)^{1/4}$, this is at most $3k(3 + (C_0 + \frac{1}{4}) \log \log n) + 2$, which is at most $4kC_0 \log \log n$ for large n , because $C_0 \geq 1$. Run Steps 2–4 on each of the at most 2^{2en} outputs and add the results. The time is $2^{2en} n^{O(k)} 2^{O(ekn \log \Delta)} = 2^{O(k^2 en \log(k/e))}$, because $k \log n \leq en$ for large n . Here n is large if it exceeds a bound that depends only on the constants of the hypothesis. For the finitely many smaller $n \geq K_0^{17}$, and the $k \leq (\log n)^{1/4}$ that go with them, trying all assignments is within the bound. If $e_\ell = O((\log K_\ell)^{-\alpha'})$, then $e = O((\log n)^{-\alpha'})$ because $\log K > (\log n)/(17C_0)$, and $\log(k/e) \leq (C_0 + \frac{1}{4}) \log \log n$. This proves Theorem D(a).

Step 6: the pairs of (40). By Theorem B, Theorem A applies with $\rho = 1/\sqrt{2}$ and $\alpha' = \frac{1}{2}$. That its pairs satisfy the hypothesis of Theorem D(a) is checked in the proof of Theorem A(c). By Theorem A(a) and its proof, $\frac{1}{2}\rho^\ell \leq e_\ell \leq 0.85\rho^\ell$ and $x_0 2^\ell \leq \log K_\ell \leq (3x_0 + 6)2^\ell$, so $e_\ell = \Theta((\log K_\ell)^{-1/2})$. Here the constants of the hypothesis are absolute, because the constant c of Theorem B is. So for $n \geq K_0^{17}$ and $3 \leq k \leq (\log n)^{1/4}$ the algorithm of Theorem D(a) takes time $2^{O(k^2 n \log \log n / \sqrt{\log n})}$ with an absolute constant. As in Step 5, k is the largest number of literals in a clause, or 3 if that is larger, which multiplies k^2 by at most 9. In the other cases, that is, if $n < K_0^{17}$ or this k exceeds $(\log n)^{1/4}$, the algorithm tries all assignments; as there are fewer than $(2n+1)^k \leq n^{2k}$ clauses, this takes time $2^n n^{O(k)}$, reading the input included. It is within the bound: $\log n \leq n/\sqrt{\log n} \leq n \log \log n / \sqrt{\log n}$ for $n \geq 4$, so $k \log n \leq k^2 n \log \log n / \sqrt{\log n}$; if $k > (\log n)^{1/4}$, then $k^2 n \log \log n / \sqrt{\log n} \geq n \log \log n \geq n$; and if $n < K_0^{17}$, then $n \log \log n / \sqrt{\log n} \geq n/\sqrt{17 \log K_0}$. This proves Theorem D(b). ■

Remarks. The algorithm is deterministic and exact: probabilities occur only in the proofs of Lemmas 1, 8 and 13, as fractions of finite sets that the algorithm searches. Time is counted in operations, as in §3. Of the two factors k , one comes from the up to k tests of a mark and the other from $\log \Delta = O(k \log(k/e))$. The bound in (b) is $2^{o(n)}$ as long as $k = o((\log n)^{1/4} / \sqrt{\log \log n})$. In (a) the bound $(\log n)^{1/4}$ on k is a convenient choice: with the bound on Δ in the Setting replaced by $\Delta \leq n^{1/(40C_0)}$, the proof works as long as $k \log(k/e)$ is at most a small enough multiple, depending on C_0 , of $\log n$. This proof needs some restriction: for k near $\log n$ and e near $(\log n)^{-C_0}$ with $C_0 > 2$, the number Δ is no longer $n^{o(1)}$, while $k^2 e \log(k/e) < 1$, so that exhaustive search does not give the bound either. Only $\log \Delta$ enters, so any form of the Sparsification Lemma with $2^{O(\epsilon n)}$ outputs, found in time $2^{O(\epsilon n)} n^{O(k)}$, in which the number of occurrences is polynomial in $1/\epsilon$ and no two outputs share a satisfying assignment gives the same bound for fixed k . The factor $\log \Delta = O(k \log(k/e))$ is paid in (68): up to $14n$ marks are placed among up to $\bar{r}\bar{s} = O(n\Delta \log \Delta)$ incidences, and the shared strings say which.

This bound, too, is purely asymptotic. For $k = 3$ the main term $28ekn \log(96\Delta \log(2\Delta))$ of the exponent, from (68), is below n only if $84e \log(96\Delta \log(2\Delta)) < 1$. For an arbitrary formula, Step 5 bounds Δ by $2(2 + 4\alpha(32\alpha)^{k-2}) + 2$, with the α defined there, which for $k = 3$ and

$\varepsilon \leq 2e$ is at least $4096 e^{-2} \log^2(64/e)$; with this value the condition requires $e < 2.1 \cdot 10^{-4}$. Since $e = e_\ell \geq \frac{1}{2} \cdot 2^{-\ell/2}$, this means $\ell \geq 23$, and $\log n \geq 17 \log K_\ell$ with $\log K_\ell \geq 60 \cdot 2^\ell - 6\ell - 42$, as at the end of Appendix A, gives $\log n > 8.5 \cdot 10^9$. In particular, the bound proved here is no better than that of trying all assignments unless $n > 10^{10^8}$.

APPENDIX E. THE MAIN RESULTS IN LEAN

This appendix states two results of this paper in the language of the proof assistant Lean 4 [MU21], on top of its mathematical library Mathlib [Mat20]: that the Exponential Time Hypothesis is false, which Theorem 2 implies, and Theorem 1, the Main Theorem; what differs from the printed statements is listed below. The listing is a complete Lean file, with its own comments, and exactly the file that was checked. It imports only Mathlib and contains definitions only, no proofs. Two of its definitions are the claims:

- (a) **ETHIsFalse**: $s_3 = 0$, where s_3 is the infimum of all $\delta \geq 0$ such that some deterministic machine, in Mathlib’s multi-stack model of Turing machines and with finite stack alphabets, decides 3-SAT on n variables within $c \cdot 2^{\delta n}$ steps, for some c and all n .
- (b) **MainTheorem**: there exist one such machine and one constant C , both chosen before k and the formula, such that for every $k \geq 1$, every $n \geq 4$ and every k -CNF on variables $x_0, \dots, x_{n-1}$, each of which occurs, with $m \geq 2$ clauses, the machine computes the number of satisfying assignments within $2^C \min\{km/\sqrt{\log m}, k^2 n \log n/\sqrt{\log n}\}$ steps.

What has been checked. Both propositions have been proved in Lean 4 (version 4.33.1) over Mathlib (commit 0df444a360eaa60ab8c11dca51a86af692955474), with time measured in steps of Mathlib’s deterministic multi-stack Turing machine. The first is derived from the second, by an argument that applies to every machine with the bound of the second. Apart from Mathlib, every result the proofs use is proved in the development, and they rest on no axiom beyond Lean’s three standard ones (`propext`, `Classical.choice`, `Quot.sound`). They are accepted by Lean’s kernel, by Lean’s replay checker `leanchecker`, module by module, and, through Comparator, by the independently implemented kernel `nanoda`; `leanchecker` has also checked again, starting from an empty environment, every declaration on which they depend, Mathlib’s included, on an earlier copy of the development whose Lean files have the same code, token for token, and differ from the published ones only in comments and white space. All 1,050 Lean modules of the development have been built together from nothing, on Mathlib fetched compiled. Whether the two definitions say what they are meant to say, that the Exponential Time Hypothesis is false and the Main Theorem, no machine can check; that is for the reader of the listing to judge. The Lean development was written by AI models and reviewed by other instances of the same models; at the time of writing no person is known to have checked it; its constant is not explicit.

The full Lean proof, with instructions for building and checking it, is at <https://github.com/anthropics/formal-math/tree/main/eth>; the listing is its file `MainResults.lean`.

```
/-
```

```
Copyright (c) 2026 Anthropic, PBC. All rights reserved.
Released under Apache 2.0 license as described in the file LICENSE.
SPDX-License-Identifier: Apache-2.0
```

```
-/
```

```
import Mathlib
```

```
/-!
```

```
# The two main results of the accompanying paper, as Lean statements
```

```
This file has definitions only: no theorem, no proof. It imports Mathlib and nothing else, so it
can be read, and compiled, on its own. Two of its definitions are the claims:
```

* `ETHIsFalse`: the Exponential Time Hypothesis, `ETH`, is false.
 * `MainTheorem`: one machine counts the satisfying assignments of every k -CNF (no clause written twice) with $k \geq 1$ and $m \geq 2$ clauses on $n \geq 4$ variables $x_0 \dots x_{n-1}$ (the paper's $x_1 \dots x_n$), each of which occurs, within $2^{\{C \cdot \min\{km/\sqrt{\log m}, k^2 \cdot n \cdot \log \log n / \sqrt{\log n}\}\}}$ steps ($\log$ is $\log_2$), for one constant C that does not depend on k .

A proof is a separate file that imports this one and states theorems whose types are exactly these two propositions.

****The machine.**** An algorithm is a machine of Mathlib's kind `Turing.FinTM2`: a deterministic machine with finitely many stacks in place of tapes. Its program is a finite set of labelled blocks; it has finitely many internal states; on a stack it can push a symbol, pop the top symbol, or read it. `Turing.TM2ComputableAux Bool Bool` adds a bijection from the alphabet of the input stack to $\text{Bool} = \{\text{false}, \text{true}\}$, and one from the alphabet of the output stack; bits are written through them. Time is the number of steps. One step runs the current block until it jumps to a block (possibly itself) or halts; for a fixed machine this is a bounded number of stack operations. `Turing.TM2OutputsInTime M.tm x (some y) t` says that the machine, started with x on its input stack and all other stacks empty, halts after at most t steps with y on its output stack, all other stacks empty, and its internal state back at its initial value. Of the stack alphabets, Mathlib requires only that of the input stack to be finite; the condition ` $\forall i, \text{Finite } (M.\text{tm}.\Gamma i)$ `, used below, requires every stack alphabet to be finite, as for a textbook machine.

****Why this machine suffices.**** A random-access machine simulates such a machine with a bounded number of operations per step, and it can rewrite a formula on the paper's variables $x_1 \dots x_n$ into the input words used here (with $x_0 \dots x_{n-1}$) in time polynomial in n and the length of the formula. In the paper a formula is a set of clauses, so no clause is repeated in it. For ETH: a 3-CNF on n variables then has length polynomial in n . So, for $\delta > 0$, a machine here that runs in time $O(2^{\delta n})$ gives an algorithm in the usual sense that runs in time $O(2^{\delta n})$. Hence s_3 (defined below) for algorithms in the usual sense is at most s_3 as defined here, and $s_3 = 0$ here gives $s_3 = 0$ there.

For the Main Theorem: every variable occurs and no clause is repeated, so $n \leq km$ and $m \leq (2n+1)^k$; then the formula has $O(km \cdot \log n)$ bits, and every polynomial in that length is at most $2^{O(\min\{km/\sqrt{\log m}, k^2 \cdot n \cdot \log \log n / \sqrt{\log n}\})}$. So a machine as in `MainTheorem` gives an algorithm in the usual sense within the same bound, with another constant. The simulation and the rewriting are standard facts; they are not part of the Lean statements or proofs.

-/

namespace MainResults

open Turing

open scoped NNReal ENNReal

/-! ### Formulas -/

/-- A literal on $x_0 \dots x_{n-1}$ is a pair (i, b) : it is x_i if b is true and $\neg x_i$ if b is false. -/
 abbrev Lit (n : ℕ) : Type := Fin n × Bool

/-- A clause is the OR of a list of literals; a formula is the AND of a list of clauses. -/
 abbrev CNF (n : ℕ) : Type := List (List (Lit n))

/-- An instance of k -SAT: every clause has at most k literals, and no clause is written twice. A clause is a list: it may be empty (then it is false), may repeat a literal or hold both x and $\neg x$, and the same literals in another order make a different clause. So a k -CNF on n variables has at most $(2n+1)^k$ clauses. (Without this, no time bound in n alone could hold: before halting, a machine must pop nearly all of its input, a bounded number of symbols per step.) -/
 def IsKCNF (k : ℕ) {n : ℕ} (φ : CNF n) : Prop := (∀ C ∈ φ, C.length ≤ k) ∧ φ.Nodup

/-- The number written for one position of a clause: 0 for an empty position, $2i+1$ for $\neg x_i$, $2i+2$ for x_i . -/
 def slot {n : ℕ} : Option (Lit n) → ℕ

| none => 0
 | some l => 2 * l.1.val + 1 + l.2.toNat

/-! ### The Exponential Time Hypothesis -/

```

/-- Some assignment makes a literal of every clause true. -/
def Satisfiable {n : ℕ} (φ : CNF n) : Prop := ∃ a : Fin n → Bool, ∀ C ∈ φ, ∃ l ∈ C, a l.1 = 1.2

/-- The input word for k-SAT with k fixed, n in unary: 1n 0, then for each clause k positions of
`Nat.log2 (2 * n) + 1` bits each ( $\lceil \log_2 (2n) \rceil + 1$  bits; 1 bit when n = 0), each holding a `slot`
number, lowest bit first. A clause with fewer than k literals is padded with empty positions (0). -/
def encClauses (k : ℕ) {n : ℕ} (φ : CNF n) : List Bool :=
  List.replicate n true ++ false :: φ.flatMap fun C => (List.range k).flatMap fun j =>
    (List.range (Nat.log2 (2 * n) + 1)).map (slot C[j]?).testBit

/-- `M` decides k-SAT within t(n) steps: started on `encClauses k φ`, for any n and any k-CNF φ on
x0 ... x{n-1} with any number of clauses, it halts after at most t(n) steps with one bit as output,
which is 1 if and only if φ is satisfiable. Here k is fixed and is not a part of the input. -/
def DecidesKSat (k : ℕ) (M : TM2ComputableAux Bool Bool) (t : ℕ → ℕ) : Prop :=
  ∀ (n : ℕ) (φ : CNF n), IsKNF k φ → ∃ b : Bool, (b = true ↔ Satisfiable φ) ∧
    Nonempty (TM2OutputsInTime M.tm ((encClauses k φ).map M.inputAlphabet.invFun)
      (some ([b].map M.outputAlphabet.invFun)) (t n))

/-- "Some deterministic algorithm decides k-SAT on n variables in time O(2δn)": one machine,
finite alphabets, one constant c; the machine decides k-SAT within c·2δn steps, rounded down,
for every n. The machine and c may depend on k and on δ. -/
def InTime (k : ℕ) (δ : ℝ ≥ 0) : Prop :=
  ∃ (M : TM2ComputableAux Bool Bool) (c : ℕ), (∀ i, Finite (M.tm.Γ i)) ∧
    DecidesKSat k M fun n => ⌊(c : ℝ) * (2 : ℝ) ^ ((δ : ℝ) * n)⌋+

/-- sk: the infimum of all δ ≥ 0 such that some deterministic algorithm decides k-SAT on n
variables in time O(2δn). It is a number in [0, ∞], and ∞ if there is no such δ. Only s3 is
used below. -/
noncomputable def s (k : ℕ) : ℝ ≥ 0 := ⓧ (δ : ℝ ≥ 0) ( _ : InTime k δ), (δ : ℝ ≥ 0 ∞)

/-- **The Exponential Time Hypothesis** (Impagliazzo and Paturi, "On the complexity of k-SAT",
J. Comput. Syst. Sci. 62 (2001)), in the form s3 > 0: for some δ > 0 no deterministic algorithm
decides 3-SAT on n variables in time O(2δn). The paper that this file accompanies defines ETH as
"sk > 0 for all k ≥ 3" and notes that this is the same as s3 > 0. The form here follows from that
one: take k = 3, then use that s3 for algorithms in the usual sense is at most s3 as defined here
(see the head comment); so its negation refutes both. (A remark outside Lean: a deterministic
algorithm is a randomized algorithm that uses no random bits, so allowing randomness could only make
s3 smaller, and refuting this deterministic form also refutes the randomized form.) -/
def ETH : Prop := 0 < s 3

/-- **The Exponential Time Hypothesis is false** : not s3 > 0. As s3 ≥ 0, this says s3 = 0. It is
equivalent to: for every δ > 0 some deterministic algorithm decides 3-SAT on n variables in time
O(2δn) (InTime 3 δ, which gets weaker as δ grows). -/
def ETHIsFalse : Prop := ¬ ETH

/-! ### The Main Theorem -/

/-- Every one of the variables x0 ... x{n-1} occurs in φ, positively or negatively. -/
def AllOccur {n : ℕ} (φ : CNF n) : Prop := ∀ i : Fin n, ∃ C ∈ φ, ∃ l ∈ C, l.1 = i

/-- The number of satisfying assignments of φ, among all 2n assignments to x0 ... x{n-1}. -/
def numSat {n : ℕ} (φ : CNF n) : ℕ :=
  (Finset.univ.filter fun a : Fin n → Bool => ∀ C ∈ φ, ∃ l ∈ C, a l.1 = 1.2).card

/-- The lowest n + 1 bits of the number c, lowest bit first. (A count of assignments of n variables
is at most 2n, so nothing of it is lost.) -/
def binary (n c : ℕ) : List Bool := (List.range (n + 1)).map c.testBit

/-- The input word of the Main Theorem, with k as a part of it: 1k 0, then `encClauses k φ`. -/
def encInput (k : ℕ) {n : ℕ} (φ : CNF n) : List Bool :=
  List.replicate k true ++ false :: encClauses k φ

/-- The paper's log: the logarithm to base 2, not rounded, and of x itself, not of x + 1. In the
comments of this file, log always means this. Under the conditions of `MainTheorem` (m ≥ 2, n ≥ 4)
it is applied only to numbers that are at least 2. -/

```

```

noncomputable def lg (x : ℝ) : ℝ := Real.logb 2 x

/-- The exponent in the number m of clauses: k·m/√(log m). -/
noncomputable def expM (k m : ℕ) : ℝ := (k : ℝ) * m / Real.sqrt (lg m)

/-- The exponent in the number n of variables: k²·n·log log n/√(log n). -/
noncomputable def expN (k n : ℕ) : ℝ := (k : ℝ) ^ 2 * n * lg (lg n) / Real.sqrt (lg n)

/-- The time of the Main Theorem, rounded down: 2^{C·min{km/√(log m), k²·n·log log n/√(log n)}}. -/
noncomputable def timeMin (C k m n : ℕ) : ℕ :=
  ⌊(2 : ℝ) ^ ((C : ℝ) * min (expM k m) (expN k n))⌋_+

/-- `M`, started on `encInput k φ`, halts after at most t steps with output the number of
satisfying assignments of φ, written in n + 1 bits, lowest bit first (`binary`). -/
def CountsWithin (M : TM2ComputableAux Bool Bool) (k : ℕ) {n : ℕ} (φ : CNF n) (t : ℕ) : Prop :=
  Nonempty (TM2OutputsInTime M.tm ((encInput k φ).map M.inputAlphabet.invFun)
    (some ((binary n (numSat φ)).map M.outputAlphabet.invFun)) t)

/-- **The Main Theorem.** There is one machine with finite alphabets, and one constant C, both
chosen before k and the formula, such that: for every k ≥ 1, every n ≥ 4, and every formula φ on the
variables x₀ ... x_{n-1}, each of which occurs in it, with `IsKCNF k φ` (clauses of at most k
literals, none written twice) and m ≥ 2 clauses, the machine, started on `encInput k φ`, outputs
the number of satisfying assignments of φ within
2^{C·min{km/√(log m), k²·n·log log n/√(log n)}} steps (rounded down). Where no thresholds are
stated, an O(·) as in the bound 2^{O(·)} speaks only of all large enough m and n; here one C
serves every m ≥ 2 and n ≥ 4, which implies that reading. These are the least values with
log m ≥ 1 and log log n ≥ 1. At n = 2 the factor log log n is 0, so the bound would be one step.
At m = 1 the divisor √(log m) is 0, so km/√(log m) is undefined (Lean's convention x/0 = 0 would
make the bound one step there too). And k ≥ 1 because the paper defines k-CNFs only for k ≥ 1; with
k = 0 every clause is empty, so there is at most one clause and no variable occurs.
Nothing is said about any other input word, including words for k = 0, for formulas with a
repeated clause or with a variable that does not occur, and for formulas with fewer than two
clauses or fewer than four variables. -/
def MainTheorem : Prop :=
  ∃ (M : TM2ComputableAux Bool Bool) (C : ℕ), (∀ i, Finite (M.tm.Γ i)) ∧
  ∀ (k n : ℕ) (φ : CNF n), 1 ≤ k → IsKCNF k φ → 2 ≤ φ.length → 4 ≤ n → All0occur φ →
  CountsWithin M k φ (timeMin C k φ.length n)

end MainResults

```

How to read it. The comments of the listing explain each definition and Mathlib's machine. Three remarks on notation: $\varphi.\text{length}$ is the number m of clauses; $C[j]?$ is the j th literal of the clause C , counted from 0, if there is one; and `Nonempty` turns Mathlib's record of a run into a proposition. The letter C denotes a clause in some definitions and the constant in others. An example of the input word of `MainTheorem`: for $k = n = 2$ and the single clause $x_0 \vee \neg x_1$ it is 110110010110, that is 1^k0 , 1^n0 , and the numbers 2 and 3 in three bits each, lowest bit first; the count is 3 and the output word is 110 (this formula is too small for the hypotheses; it only shows the code).

Differences from the printed statements. Lean names the absolute constant of the $O(\cdot)$: one number C , whose value the proof does not give; and the ranges $k \geq 1$, $m \geq 2$ and $n \geq 4$, which the [Main Theorem](#) leaves to its $O(\cdot)$, are explicit, as in [Corollary 11](#): the bound holds for every such k , m and n , not only for large m and n . The paper measures time in the word RAM model; the Lean statements count steps of Mathlib's machine, the reading of the input included, where one step runs the machine's program as far as its next `goto` or `halt`, a number of stack operations that is bounded for a fixed machine. The bounds are rounded down, which changes nothing for a number of steps. The input is a fixed word: 1^n0 , then for each clause k positions of $\lfloor \log(2n) \rfloor + 1$ bits, each holding $2i + 2$ for x_i , $2i + 1$ for $\neg x_i$, or 0, lowest bit first; for the [Main Theorem](#) it is preceded by 1^k0 . So there k is part of the input; it may exceed the length of the longest clause, and the bound is then the one for that k . The variables are called $x_0, \dots, x_{n-1}$

and not $x_1, \dots, x_n$ (§2). In the paper a clause is a nonempty set of literals and a formula is a set of clauses; in Lean a clause is a list of literals and a formula a list of clauses in which no clause occurs twice: a clause may be empty or repeat a literal, and the same literals in another order count as another clause. ETH is the statement $s_3 > 0$ for s_3 defined with Mathlib’s machine; for the s_3 of Definition 6, ETH as defined there is equivalent to $s_3 > 0$ by the remark after it. ETHIsFalse holds if and only if for every $\delta > 0$ some machine decides 3-SAT within $c \cdot 2^{\delta n}$ steps, the decision version of Theorem 2. Two steps from the Lean statements to the printed ones are not part of the Lean proof: writing the formula as this word, in time polynomial in n and its length, and simulating Mathlib’s machine on a word RAM, with a bounded number of operations for each step. That this polynomial is below the bound of the Main Theorem is proved in Lean: $n \leq km$, as every variable occurs, $m \leq (2n + 1)^k$, as no clause is repeated, and every polynomial in the length of the input is at most $2^{C' \min\{km/\sqrt{\log m}, k^2 n \log \log n/\sqrt{\log n}\}}$ for a C' that depends only on the polynomial. The hypotheses are a promise: nothing is said about inputs that do not meet them.

Differences in the proof. The Lean proof follows the route of the proof of Corollary 11: from one algorithm for cliques it derives Theorem D(a), and from that the bound of Theorem D(b), as in Step 6 of Appendix D; from the same algorithm, the first sentence of Theorem A(c) for the pieces of Appendix B; and from these two, by running the two machines in alternation, Corollary 11. Each step is proved for *every* machine that is built from the machines of the step before as the printed proof builds it, which fixes what a machine outputs and a bound on its steps, not the steps themselves; that such a machine exists is proved separately. The algorithm for cliques is the recursion of Theorem A(b) as a machine with query access to the tables, where an operation is one query or one step, a step may be any computation that takes time polynomial in the size of the machine’s state, and the states are bounded only by the bound on the operations; Lean proves that this machine satisfies the hypothesis of Theorem D(a) with operations counted in this way, so, in time, the exponent $e_\ell K_\ell^2 \lambda$ holds only up to a constant factor, which the O in the exponent of the conclusion of Theorem D(a) absorbs. Main differences that remain: from that hypothesis on, the proof renders T operations as time polynomial in the input and a padding string of T ones; alternation and choice cost a constant power on Mathlib’s machine and not a constant factor; the algorithm for the pieces of Theorem B writes a padding string of about $2^{c\lambda}$ ones and then does all its work in a single step; in Step 5, Lean’s own analysis of Appendix C’s procedure replaces Calabro’s cited results [Cal09], with thresholds we take to be Calabro’s, not checked against the dissertation; the constants depend on the algorithm for cliques, not only on the constants of the hypothesis. AI models, not a mechanical check, compared the Lean proof with the printed one; because of these differences the strictest of them count it as a different proof of the same statement, along the printed route.

APPENDIX F. PROOF OF THE EIGHT-SQUARE IDENTITY

This appendix proves the identity in the form in which §4 states it. Recall that $\sigma(p, q) = (-1)^{\phi(p, q)}$ with $\phi(p, q) = \sum_{1 \leq i \leq j \leq 3} p_i q_j + q_1 p_2 p_3 + p_1 q_2 p_3 + p_1 p_2 q_3$ for blocks $p, q \in \{0, 1\}^3$, and that (28) is claimed for all blocks p, q, p', q' with $p \oplus q = p' \oplus q'$ and $q \neq q'$.

Proof of (28). Let $d = q \oplus q'$; it is not 000 because $q \neq q'$, and it equals $p \oplus p'$ because $p \oplus q = p' \oplus q'$. So $p' = p \oplus d$ and $q' = q \oplus d$. We have to show that $\phi(p, q) + \phi(p \oplus d, q \oplus d) + \phi(p, q \oplus d) + \phi(p \oplus d, q)$ is odd, and compute modulo 2. Every term of $\phi(p, q)$ contains exactly one bit of q , so $\phi(p, q) + \phi(p, q \oplus d) \equiv \phi(p, d)$ for all p , and the sum is $\phi(p, d) + \phi(p \oplus d, d)$. Replacing p by $p \oplus d$ changes $\sum_{1 \leq i \leq j \leq 3} p_i d_j$ by $\sum_{1 \leq i \leq j \leq 3} d_i d_j$, and it changes $d_1 p_2 p_3$ by $d_1(p_2 d_3 + d_2 p_3 + d_2 d_3)$, $p_1 d_2 p_3$ by $d_2(p_1 d_3 + d_1 p_3 + d_1 d_3)$, and $p_1 p_2 d_3$ by $d_3(p_1 d_2 + d_1 p_2 + d_1 d_2)$. In these three changes

each of $p_1d_2d_3$, $p_2d_1d_3$, $p_3d_1d_2$ occurs twice and $d_1d_2d_3$ three times. So $\phi(p, d) + \phi(p \oplus d, d)$ is congruent to the sum of the four changes, in which the terms that occur twice cancel. With $d_i^2 = d_i$ this is $d_1 + d_2 + d_3 + d_1d_2 + d_1d_3 + d_2d_3 + d_1d_2d_3 \equiv 1 + (1 + d_1)(1 + d_2)(1 + d_3)$, which is 1 because some d_i is 1. ■

Proof that the two forms are equivalent. Expand the right side of the second form (29): $a_p^2b_q^2$ has the coefficient $\sigma(p, q)^2 = 1$, and every other monomial that occurs is $a_p a_{p'} b_q b_{q'}$ with p, q, p', q' as in (28), with twice the left side of (28) as its coefficient. So the two sides are equal as polynomials exactly when (28) holds. ■

APPENDIX G. RELATED WORK IN MORE DETAIL

Before this work, the fastest randomized algorithms known ran in time $O(1.308^n)$ ($O(2^{0.3874n})$) for 3-SAT and $2^{(1-\mu_k/(k-1))n+o(n)}$ for k -SAT, with constants μ_k that increase to $\pi^2/6$ [PPSZ05, Her14]; later work has lowered the exponents only minutely [SS17, HKZZ19, Sch21]. The fastest deterministic ones ran in time $O(1.32793^n)$ ($O(2^{0.4092n})$) for 3-SAT and slightly below $(2 - 2/k)^{n+o(n)}$ for k -SAT [MS11, Liu18].

The closest work we know is Austrin, Kaski and Kubjas [AKK22]. Their object contains ours: an index on every vertex of a hypergraph, an arbitrary table on every hyperedge, and the sum, over all values of the indices, of the product of the table entries. For the (K, λ) -clique the vertices are the $\binom{K}{2}$ pairs of nodes (indices in $\{0, 1\}^\lambda$) and the hyperedges are the K stars (all pairs containing a node w ; table W_w). For algorithms that execute *one* tensor network along *one* contraction tree, they prove an upper bound via fast matrix multiplication and an unconditional lower bound of $\Omega((2^\lambda)^{\text{bw}})$ operations. Here bw is the branchwidth of the hypergraph: over all trees with the hyperedges as leaves and inner nodes of degree 3, the minimum of the largest number of vertices shared by the two sides of a tree edge. Every such tree has an edge with about $K/3$ or more stars on each side, and these share about $2K^2/9$ or more pairs (our calculation, not in their paper [AKK22]); so in their model (K, e) is not achievable for any e below about $\frac{2}{9}$, which is $\frac{4}{9}$ of the $e = \frac{1}{2}$ of brute force. The algorithm of this paper is outside that model: it writes Z , times a power of 2, as a signed sum of many networks, and merges rows in some terms and columns in others. A sum of networks can be written as one network with one more index, but then all terms are contracted in the same order.

The steps from a formula to a network and from a network to a clique (§6, Lemma 1) are a variant of the move of Williams [Wil05], after Nešetřil and Poljak [NP85]: grouping the variables of a 2-CSP into three blocks turns counting into a weighted triangle problem. There the saving comes only from fast matrix multiplication, which is not used here. Christandl, Vrana and Zuiddam [CVZ19] bound the asymptotic tensor rank of the complete-graph tensor, which underlies the (K, λ) -clique; a decomposition into rank-one terms needs at least $2^{\lfloor K^2/4 \rfloor \lambda}$ terms (the matrix rank when the nodes are split into two halves), whereas the terms used here are networks.

Low-rank expansions of a crossing, the function $[\mathbf{x} = \mathbf{v}][\mathbf{y} = \mathbf{u}]$, have been studied in connection with lower bounds for non-commutative arithmetic circuits. Let $P(N)$, $N = 2^\lambda$, be the least number of rational-valued functions, each of row-wise or of column-wise rank 1 (the rank of the matrix with first index $(\mathbf{x}, \mathbf{y})$ and second index $(\mathbf{u}, \mathbf{v})$, respectively $(\mathbf{x}, \mathbf{u})$ and $(\mathbf{y}, \mathbf{v})$), with sum $[\mathbf{x} = \mathbf{v}][\mathbf{y} = \mathbf{u}]$. Naslund [Nas20, Problem 12] asks whether $P(N) \geq N^{1+\varepsilon}$ for some $\varepsilon > 0$ and all large N . By Lemma 3.5 of Hrubeš, Wigderson and Yehudayoff [HWY11], $P(N)$ is the width over $\mathbb{Q}$ of $\sum_{i,j \in [N]} x_i y_j x_i y_j$, and by their Proposition 4.1 it lies between s and $4s$, where s is the least number of products in an identity $(\sum_{i \in [N]} x_i^2)(\sum_{j \in [N]} y_j^2) = \sum_{k \in [s]} f_k f'_k$ with bilinear forms f_k, f'_k over $\mathbb{Q}$, a variant of Hurwitz's problem. Hrubeš [Hru24, Theorem 12]

shows $s = O(N^{1.62})$ over every field, sketching the proof as a modification of his Theorem 1, so $P(N) = O(N^{1.62})$. The eight-square identity, from which Lemma 2 starts, is the case $N = s = 8$; from it Hrubeš outlines a tensor product construction with $8^{11\ell/6}$ squares over $\mathbb{C}$ for $N = 8^\ell$ [Hru24, Section 5.2], and Lemma 2 likewise gives $P(N) = O(N^{11/6})$, by taking the A_i with $i < t/2$ row-wise and the others column-wise, and dividing by 2^t . What Proposition 3 uses is, for each A_i , explicit integer factorizations in the two directions through at most $11\lambda/3 + 8$ bits together. A bound $P(N) \leq N^c$ gives two functions whose row-wise and column-wise ranks have product at most N^{2+c} , which for $c = 1.62$ is slightly less than $N^{11/3}$. So with integer factorizations of polynomial bit length that are computable in polynomial time, Hrubeš's bound would give a smaller ρ , about 0.905; his proof of Theorem 12 is only sketched, and we have not checked whether his construction provides such factorizations. In algorithms, the closest precedent for expanding crossings that we are aware of is Curticapean and Xia [CX15], after Galluccio and Loeb1 [GL99]: to count perfect matchings in non-planar graphs, they simulate a gadget that does not exist as a linear combination of $O(1)$ gadgets that do, and then solve every term in the same way. Further afield: the asymptotic rank conjecture and the set cover conjecture cannot both hold [BK24, Pra24], which is conditional, whereas the bound of Lemma 2 is explicit; rank-based dynamic programming [BCKN12, CKN12] factorizes across one fixed family of cuts; and under ETH, treewidth-based algorithms for CSPs are essentially optimal, up to a logarithmic factor in the exponent [Mar10]. We are not aware of the combination used here (an exact signed low-rank expansion of crossings, a separate choice of direction for each term, combined with recursion) appearing in the prior literature.

APPENDIX H. HOW THE SEARCH FOR CHEAPER FUNCTIONS LEADS TO THE EIGHT-SQUARE IDENTITY AND TO $\rho = 1/\sqrt{2}$

No proof in this paper uses this appendix. The algorithm needs a nonzero multiple of the crossing $[x = v][y = u]$ written as a sum of a few functions, each of which, for long strings, factors through fewer bits, row-wise and column-wise together, than the crossing itself (Lemma 2, Definition 7). The first part shows, with hindsight, how two natural guesses lead someone who looks for such a decomposition to a classical problem of Hurwitz, whose extreme solutions are the two-, four- and eight-square identities. The second part gives a hint that leads from there to the pieces of Appendix B. Much of this is known, and we say where. We are not aware of earlier work that counts the bits of a single term in both directions together, or that reads the directions as the two orders of a product.

From the crossing to sums of squares. Let x, y, u, v range over a set of N elements. Row-wise, that is, as a matrix with row index (x, y) and column index (u, v) , and column-wise, with the indices (x, u) and (y, v) (§4), the crossing $[x = v][y = u]$ is a permutation matrix, of rank N^2 . A function that factors through c bits has rank at most 2^c (see after Definition 1), so the crossing needs $2 \log N$ bits in each direction. We look for functions R and C whose sum is a nonzero multiple of the crossing, R of small rank row-wise and C column-wise.

Two guesses. Exchanging y and u turns the row-wise arrangement into the column-wise one, and it does not change the crossing. So the first guess is that the solution has the same symmetry: $C(x, y, u, v) = R(x, u, y, v)$. Then C is column-wise the same matrix as R is row-wise. With row index (x, y) and column index (v, u) the crossing is the identity matrix, so the second guess is that R is symmetric and positive semidefinite, of rank M : $R(x, y, u, v) = \sum_{k=1}^M f_k(x, y) f_k(v, u)$ with real functions f_k . That $R + C$ is twice the crossing now reads

$$(71) \quad \sum_{k=1}^M f_k(x, y) f_k(v, u) + \sum_{k=1}^M f_k(x, u) f_k(v, y) = 2[x = v][y = u].$$

What the equation says. Take real variables a_x and b_y , multiply (71) by $a_x b_y a_v b_u$, and sum over x, y, u, v . With $z_k = \sum_{x,y} f_k(x, y) a_x b_y$, both sums on the left give $\sum_k z_k^2$, so

$$(72) \quad z_1^2 + \cdots + z_M^2 = \left(\sum_x a_x^2 \right) \left(\sum_y b_y^2 \right).$$

Conversely, the coefficient of $a_x a_v b_y b_u$ in the difference of the two sides of (72) is $\frac{1}{2}$, 1 or 2 times the difference of the two sides of (71), so (72) implies (71). Under the two guesses the question is therefore: for which M is the product of two sums of N squares a sum of M squares of bilinear forms? This is a case of Hurwitz's problem [Hur98], [Hru24, Section 1].

The equivalence is known. Hrubeš, Wigderson and Yehudayoff write the crossing as the non-commutative polynomial $\sum_{i,j} x_i y_j x_i y_j$ [HWY11, (1.4)]. They pass from a decomposition of it to an identity like (72) by letting the variables commute, and back, for a sum of squares, by adding to each square the same product with the two y -variables exchanged [HWY11, Proposition 4.1 and its proof]. Applied to the eight-square identity, their construction is (31). Hrubeš writes out its coefficients, which are half the left side of (71) [Hru24b, proof of Lemma 22, (10)]. In terms of matrices, with the indices (x, y) and (v, u) : if Φ is the $M \times N^2$ matrix with rows f_k , then $R = \Phi^\top \Phi$ and C is its partial transpose [RZ25, proof of Lemma 3.15].

The least M . For fixed a with $\sum_x a_x^2 = 1$, the linear map $b \mapsto z$ from $\mathbb{R}^N$ to $\mathbb{R}^M$ preserves the norm by (72), so $M \geq N$ [HWY11, Section 1.2]. Hurwitz proved in 1898 that $M = N$ is possible only for $N \in \{1, 2, 4, 8\}$ [Hur98], [HWY11, Theorem 1.1]; the least M for general N is still open [Hru24, Section 1]. The identities themselves are older: after Euler's four-square identity of 1748 [HWY11, Section 1.2], Degen found the eight-square identity in 1818, and Graves found it again in 1843, the year of Hamilton's quaternions, by showing that the norm of the octonions is multiplicative [Bae02, Section 1]. The functions $f_k = \psi_k$ of §4 are a solution for $N = 8$; their σ is the table of signs of the multiplication of the octonions [Bae02, Section 2, Table 1], with the basis elements renamed. Their restrictions to the blocks of the forms $(p_1, p_2, 0)$ and $(p_1, 0, 0)$ are solutions for $N = 4$ and $N = 2$.

Why eight. For strings of λ bits, Definition 5 uses the identity once for each block of three bits; do the same with blocks of $\log N$ bits. The proof of Lemma 2 counts the bits of a piece A_i in a simple way: it factors R or C in each block, and it names the set of the blocks that take C , with one more bit for each block in each direction. By this count, in the two directions together, a block takes the logarithms of the two ranks of R and 2 more bits, against $4 \log N$ bits for the crossing. Row-wise and column-wise, R has the ranks 2 and 2 for $N = 2$, the ranks 4 and 16 for $N = 4$, and the ranks 8 and 64 for $N = 8$. This makes $1 + 1 + 2 = 4$ of 4 bits, $2 + 4 + 2 = 8$ of 8 bits, and $3 + 6 + 2 = 11$ of 12 bits. So by this count, of the three identities only the eight-square identity leaves a gain, the $\rho = 11/12$ of Lemma 2. Hrubeš tensors the same identity block by block and counts squares rather than bits [Hru24, Section 5.2]; see Appendix G.

The count is not tight. With row index (x, y) and column index (v, u) , the three identities have $R^2 = NR$ and $C = 2 \cdot \text{Id} - R$, so R and C have two common eigenspaces, of dimensions N and $N^2 - N$, and the rank of A_i is the number of its nonzero eigenvalues, counted with multiplicity. For $\lambda = t \log N$, where no zeros are appended, this gives the rank $\sum_{z \leq i} \binom{t}{z} (N^2 - N)^z N^{t-z}$ row-wise for $N = 4, 8$, and the same with $t - i$ in place of i column-wise. For large t , the logarithms of the two ranks of the worst piece, $i = t/2$, add up to about $2 + \log 48 \approx 7.58$ of 8 bits for each block if $N = 4$, and to about $2 + \log 448 \approx 10.81$ of 12 bits if $N = 8$; for $N = 2$ both ranks are $\binom{t}{i} 2^t$, and there is still no gain. So counted by ranks, four squares leave a gain as well. Without the first guess, so do single bits: $[x = v][y = u] = [x = u][y = v] - [x \neq y](-1)^x \cdot [u \neq v](-1)^u$, and the same argument for the pieces made from these two terms in the same way, with eigenspaces of dimensions 1 and 3 in both directions, gives $\log 12 \approx 3.58$ of 4 bits, a slightly

larger gain than that of eight squares. So “eight” is the answer under the two guesses and for the simple count, which is the count that Lemma 2 uses.

From the octonions to operators on a graded space. The hint takes two sentences. *The two directions are the two orders of multiplying two operators, and the number of bits is at most the logarithm of the dimension of the space through which the product passes. So replace the octonions, where both orders pass through one space of dimension 8, by operators on a graded space, where one order goes up in degree and the other goes down.*

The identity as a relation between operators. Let L_x be the 8×8 matrix with entry $\psi_k(x, y)$ in row k and column y , the matrix of multiplication by e_x from the left in the basis (e_p) of §4. By (30),

$$(73) \quad R(x, y, u, v) = \langle e_u, L_v^\top L_x e_y \rangle, \quad C(x, y, u, v) = \langle e_u, L_x^\top L_v e_y \rangle,$$

so (31) says, entry by entry,

$$(74) \quad L_v^\top L_x + L_x^\top L_v = 2[x = v] \cdot \text{Id}.$$

For $M \times N$ matrices, these are the Hurwitz–Radon conditions, the classical form of (72) [Hru24, Section 3, Lemma 3]. Here $L_{000} = \text{Id}$ and $L_x^\top = -L_x$ for $x \neq 000$, so for $x, v \neq 000$ they read $L_x L_v + L_v L_x = -2[x = v] \cdot \text{Id}$, the relations of a Clifford algebra [Bae02, Section 2.3].

Directions are orders. In (73), x and v name operators, y names the basis vector to which they are applied, and u names the coordinate that is read off. In R the operator named by x acts first, and in C the operator named by v . In general, let X_x and Y_v be linear maps between spaces with fixed bases, (e_τ) the basis of the space that contains the vectors $X_x e_y$, and $\langle e_\tau, w \rangle$ the coefficient of e_τ in w . Then

$$(75) \quad \langle e_u, Y_v X_x e_y \rangle = \sum_\tau \langle e_\tau, X_x e_y \rangle \cdot \langle e_u, Y_v e_\tau \rangle$$

is a sum of functions of (x, y) times functions of (u, v) , with one term for each τ . In the other order, $\langle e_u, X_x Y_v e_y \rangle$ is a sum of functions of (y, v) times functions of (x, u) . So we want to write $[x = v] \cdot \text{Id}$ as a combination of products, in the two orders, that pass through small spaces.

What limits the octonions. In (74) each product stands in one order only, and it has full rank in the other direction: R has the ranks 8 and 64, that is, $3 + 6$ of 12 bits. The piece A_0 of Definition 5 takes R in every block, so for $\lambda = 3t$ its ranks are 8^t and 64^t : however one counts, a term that takes R in every block takes $\frac{3}{4}$ of the bits. To get below $\frac{3}{4}$, look for terms to which the argument of “Directions are orders” applies in both directions. Such a term can be written in both orders: it is a combination of products that does not change when the two factors are exchanged in all of them. And the two orders should pass through spaces of different sizes: if $\lambda^{O(1)}$ pieces add up to a nonzero multiple of the crossing of strings of λ bits, one of them needs $2\lambda - O(\log \lambda)$ bits row-wise, which leaves it at most about $(4\rho - 2)\lambda$ column-wise.

A second relation of the same form. In the notation of Appendix B, on the polynomial ring $V = \mathbb{Q}[t_1, t_2, \dots]$ and for $k, l \geq 1$,

$$(76) \quad \nabla_k \nabla_{-l} - \nabla_{-l} \nabla_k = k[k = l] \cdot \text{Id} = (\nabla_k t_l) \cdot \text{Id}.$$

Like (74), this writes a multiple of $[k = l] \cdot \text{Id}$ as a combination of the two orders. The new feature is the grading: ∇_{-l} raises the weight by l , and ∇_k lowers it by k . For $k = l$, on V_E , one order passes through V_{E+k} and the other through V_{E-k} . As it stands, (76) is not enough: each product stands in one order only, and k, l take far fewer values than there are monomials, whereas $\mathbf{x}$ and $\mathbf{y}$ have the same length, so we need as many operators as basis vectors.

What Appendix B supplies. For every polynomial a and every integer r it defines an operator $a[r]$, a sum of products of the ∇_k , that raises the weight by r ; for example, $t_1[r] = \nabla_{-r}$

for $r \neq 0$. For b of weight h and a of weight at most h , the operator $Q_s(a, b)$ of (48) is a signed combination of $2h + 1$ products; product number $i \in \{0, \dots, 2h\}$ consists of an operator that comes from a and raises the weight by $s - h + i$, and one that comes from b and lowers it by as much. By Lemma 9, $Q_s(a, b)$ does not change when the factors are exchanged in all products at once, and $\sum_s Q_s(a, b) = (a[-h]b) \cdot \text{Id}$, which has the shape of (76). Lemma 10 gives polynomials with $a^\mu[-h]b^\nu = h! [\mu = \nu]$ for all partitions μ, ν of h . The piece $A_s(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v})$ is the coefficient of $t^{\mu(\mathbf{u})}$ in $Q_s(a^{\mu(\mathbf{x})}, b^{\mu(\mathbf{v})}) t^{\mu(\mathbf{y})}$, with the four roles as in (73).

The count. Here h is the least integer with $p(h) \geq 2^\lambda$, so that a string $\mathbf{x}$ of λ bits names a partition $\mu(\mathbf{x})$ of h ; then $\lambda \approx c_{\text{HR}} \sqrt{h}$ by (46). On a monomial of weight h , product number i passes through weight $s + i \leq 2h + s$ if the operator from a acts first, and through weight $2h - s - i \leq 2h - s$ if the one from b acts first. So for $|s| \leq 2h$, up to $O(\log \lambda)$ bits, A_s factors through $c_{\text{HR}} \sqrt{2h + s}$ bits row-wise and through $c_{\text{HR}} \sqrt{2h - s}$ bits column-wise, and

$$(77) \quad c_{\text{HR}} \sqrt{2h + s} + c_{\text{HR}} \sqrt{2h - s} \leq 2c_{\text{HR}} \sqrt{2h} \approx 2\sqrt{2} \lambda, \quad \text{in place of } 4\lambda.$$

What one order gains in weight the other loses, and $\log \dim V_E \approx c_{\text{HR}} \sqrt{E}$ grows like a concave function of E . If it grew like E , the sum would be 4λ again; if it grew like E^α with $0 < \alpha \leq 1$, the same calculation would give $\rho = 2^{\alpha-1}$.

The same objects elsewhere. The relations (76) are those of the Heisenberg algebra, and V with the operators ∇_k is its Fock representation [Fre02, §2.3]. Comparing (47) with [Fre02, §2.3, (4)] shows that the $a[r]$ are the coefficients of the fields $Y(a, \omega)$ of the Heisenberg vertex algebra. These fields are local, that is, $(\omega_1 - \omega_2)^H [Y(a, \omega_1), Y(b, \omega_2)] = 0$ for a large enough integer H [Fre02, §2.1 and Proposition 2.4]. Written out for each coefficient [MN99, Definition 1.5.1 and the display after it], with H the sum of the weights of a and b , locality contains Lemma 9(a) for homogeneous a : the lemma is its part about the products $a[r]b[-r]$, which keep the weight. Appendix B proves what it needs without this theory. We are not aware of earlier work that uses these operators to decompose a crossing.