Electronic Colloquium on Computational Complexity, Report No. 347 (2026)

Counting from Geographic Information Retrieval: An Elementary Refutation of ETH

Noam Mazor* Rafael Pass†

November 25, 2026

Abstract

We here provide an alternative elementary and self-contained proof for the refutation of ETH. In more detail, we show how to count satisfiable assignments to a 3SAT formula in subexponential time in the number of clauses. Together with the classical sparsification lemma of Impagliazzo, Paturi, and Zane (JCSS 2001), this gives an alternative proof that ETH is false. At the core of our proof is a new primitive which we call *geographic information retrieval* (GIR): Given a pattern of requests fixed in advance, GIR assigns addresses to readers and database entries so that the entries are well spread along each axis; it then preprocesses the database into a short public hint. With this hint, a reader can recover even a distant requested entry by reading only entries near its own address.

1 Introduction

The goal of this paper is to solve the following problem.

#3SAT with 3-Bounded Occurrences. Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a 3CNF formula over the variables $x_1, \ldots, x_N$, where each variable appears in φ at most 3 times.[1] We are interested in the number of satisfying assignments for φ.

More generally, we will solve the following, more general problem. For any set of predicates $F_1, \ldots, F_m$ such that each predicate depends on at most 3 variables among $x_1, \ldots, x_N$, and each variable appears in at most 3 predicates, we want to compute

$$S = \sum_{x \in \{0,1\}^N} \prod_{i \in [m]} F_i(x_{V(i)}). \tag{1}$$

where $V(i) \subseteq [N]$ is the set of variables F_i depends on.

We prove the following theorem.

Theorem 1 (#3SAT, [Ope26, Cla26])**.** *For every constant $\epsilon > 0$, #3SAT with 3-bounded occurrences can be computed in time $2^{\epsilon N}\mathsf{poly}(N)$.*

*New York University. E-mail: noammaz@gmail.com.

†Cornell Tech, Technion and Tel Aviv University. E-mail: rafael@cs.cornell.edu.

[1] Here, in a 3CNF formula each clause contains at most 3 literals.

ISSN 1433-8092

In Section 6 we show how to use Theorem 1 to solve (unbounded) #3SAT by applying the sparsification lemma of [IPZ01] and a standard reduction. The main part of the paper focuses on solving the above counting problem. As a first attempt, assume we could partition the predicates F_i and variables of φ into g clusters, such that each predicate depends only on variables from its own cluster. Moreover, assume that the clusters are balanced, meaning that each cluster contains at most N/g variables. We call such a clustering a *perfect clustering*. Let $P_s \subseteq [m]$ and $V_s \subseteq [N]$ denote the sets of predicates and variables in cluster s, respectively. Under this assumption, every predicate F_i, for $i \in P_s$, depends only on variables in V_s. Consequently, the product of all predicates can be decomposed as

$$\prod_{i\in[m]} F_i(x_{V(i)}) = \prod_{s=1}^{g} \underbrace{\left(\prod_{i\in P_s} F_i(x_{V(i)}) \right)}_{Q_s(x_{V_s})}.$$

This means that

$$S = \sum_{x\in\{0,1\}^N} \prod_{s=1}^{g} Q_s(x_{V_s}) \quad = \prod_{s=1}^{g} \left(\sum_{y\in\{0,1\}^{V_s}} Q_s(y) \right).$$

This decomposition immediately yields a faster counting algorithm. Instead of enumerating all 2^N assignments to the original variables, we independently evaluate g smaller sums, each involving at most N/g variables. Thus, the total running time is $\approx g \cdot 2^{N/g}$ rather than 2^N.

Unfortunately, such a balanced clustering may not necessarly exist. Thus, we need to generalize the above argument. Roughly speaking, our approach will proceed in two steps:

- We first consider a version of the counting problem above—which we refer to as *geographic counting*—where each predicate and each variable is assigned an "address" in space. We assume that (1) the variables are well spread along each coordinate, and (2) each predicate only touches variables that are close by. We show that geographic counting can still be solved in subexponential time, intuitively by clustering nearby points. The idea is to use different clusterings for different parts of the count: In more detail, each coordinate gives a different way to divide the variables into small groups. We show how to compute each part of the sum using a coordinate for which few of the dependencies in that part cross between groups.

- In the next step, we introduce a new primitive which we call *geographic information retrieval*, or *GIR*. Consider a database and a set of readers whose requests are fixed in advance. Each reader requests at most three entries, and each entry is requested by at most three readers. For any such pattern of requests, GIR assigns addresses to the readers and the entries so that the entries are well spread along each coordinate. It then preprocesses the database into a short public *hint*. Using this hint, each reader can recover every entry it requested by reading only entries near its own address. We show how to construct GIR and then use it to reduce the counting problem above to a slight generalization of geographic counting. Intuitively, even when a requested entry is far away, the reader can recover its value using the public hint and reading only entries near the reader's own address.

- To construct GIR, we first give an elementary construction of a primitive we call *robust information retrieval* (RIR). Here the database entries have binary addresses. Using a short public hint, we can recover an entry at address s by reading only entries whose addresses are close to $\bar{s}$, the address obtained by flipping every bit of s. (Intuitively, the database is robust to s being deleted, as well as everything that is close s.)

  Our construction of GIR from RIR uses many copies of RIR. In each copy, we associate every RIR entry with a reader and a database entry it wants to read, together with their addresses in the space.

  The construction has the following property: whenever two entries in the same RIR copy have different bits in some coordinate of their binary addresses, the database entry associated with the second entry has the same address as the reader associated with the first entry in that coordinate (and vice versa). Consequently, since the RIR decoder reads only entries whose binary addresses are far from the entry being decoded, the corresponding database entries in GIR agree with the reader's address in almost all coordinates. Thus, the GIR decoder reads only database entries that are geographically close to the reader.

  This can be achieved easily: once we fix the addresses of the reader and the database entry associated with one RIR position, we can deduce the addresses associated with all other positions in the same RIR copy. For each coordinate, if another position has the same bit in its binary address, we assign it the same reader and database entry addresses in that coordinate. Otherwise, we swap these two addresses.

  How do we find room for all the readers at once? We use copies of the RIR with all possible pairs of addresses. This gives every pair of entry and reader addresses the same number of places. We choose the addresses so that roughly the same number of requests need each pair.

2 Warm Up: Geographical Counting

In this part we prove a warm-up case of Theorem 1. Fix constants $\delta > 0$ and $g \geq 2$, and $u \geq 1$. We assume that every predicate F_i and variable x_j in the formula has an address $c(F_i), c(x_j) \in [g]^u$. Moreover, we will assume that the addresses have the following nice properties:

1. **Balanced:** For each coordinate $a \in [u]$, the variables are balanced on the a-th coordinate. That is, for each $s \in [g]$, the number of variables $j \in [N]$ such that $c_a(x_j) = s$ is at most $N/g + O(1)$.

2. **Closeness:** Every predicate F_i is close to each one of its variables. That is, for each vertex $i \in [m]$ and every variables $j \in V(i)$ it depends on, $d_H(c(F_i), c(x_j)) \leq \delta u$, where d_H is the Hamming distance.

That is, we assume that there is a "balanced" placement of the predicates and variables in the space in a way that each predicate only depends on variables that are close to it. (Think of placing predicates and variables on a grid, so variables are balanced, but we need to ensure that each predicate is geographically close to all its variables.) Observe that, while there may not be a perfect clustering for the formula, each coordinate of the addresses partitions the variables and predicates to a clustering that is balanced. Moreover, for each pair of predicate and one of its variables (F_i, x_j), F_i and x_j belong to the same cluster according to most of the coordinates.

We will show in this part that under this assumption, we can still count the satisfying assignments in subexponential time $\approx 2^{O(\delta N + N/g)}$. The high level idea is to use each coordinate of the address to compute the part of the sum for which it is good. We next describe the algorithm.[2]

In the following, we call a pair (F_i, x_j) such that F_i depends on x_j an incidence of the formula φ. Let $I = \{(i,j) : i \in [m],\ j \in V(i)\}$ be the set of at most $3N$ incidences.

2.1 Step I: Marking the Variables

We first identify each predicate F_i with the multilinear polynomial (in up to 3 variables) that computes F_i over $\{0,1\}^{V(i)}$. Recall that we want to compute

$$S = \sum_{x \in \{0,1\}^N} \prod_{i \in [m]} F_i(x_{V(i)}).$$

As mentioned above, for every coordinate $a \in [u]$, we would like the variables x_j that F_i depends on to have the same cluster $c_a(x_j)$ as F_i, $c_a(F_i)$, but we will have some misses. To keep track of the misses we introduce u variables $t_1, \ldots, t_u$, and multiply x_j by t_a if $c_a(x_j) \neq c_a(F_i)$.[3] That is, define the set

$$A(i,j) = \{a \in [u] : c_a(F_i) \neq c_a(x_j)\},$$

and let

$$Z_{ij}(t,x) = x_j \prod_{a \in A(i,j)} t_a. \tag{2}$$

Notice that by our assumption we have that $|A(i,j)| = d_H(c(F_i), c(x_j)) \leq \delta u$ for every i and $j \in V(i)$, and thus the t-degree of Z_{ij} is at most δu. Moreover, $Z_{ij}(\mathbf{1}, x) = x_j$. Define the polynomial

$$\Phi(t) = \sum_{x \in \{0,1\}^N} \prod_{i \in [m]} F_i\big((Z_{ij}(t,x))_{j \in V(i)}\big). \tag{3}$$

Then we have that $\Phi(\mathbf{1}) = S$. Since F_i is multilinear and each of the (at most) $3N$ expressions Z_{ij} appears once in Φ, we get that the degree of Φ is at most $3Nu\delta$. Since there are u variables, this implies that for every monomial there is at least one variable t_a with degree at most $3\delta N$. The main idea below is to break the sum into u parts, and to count each part of the sum using the coordinate a for which the degree of t_a is small.

2.2 Step II: Expanding the Product According to One Coordinate

Next, fix $a \in [u]$. For every value $s \in [g]$, let V_s be the cluster of variables x_j with $c_a(x_j) = s$. We next want to split $\prod_{i \in [m]} F_i\big((Z_{ij}(t,x))_{j \in V(i)}\big)$ into a product of expressions, each depending only on variables x_j in a single cluster $c_a(x_j)$. Toward this, and while for now $Z_{ij}(t,x)$ contains only one monomial, it will be useful to write $Z_{ij}(t,x)$ as a sum of two terms: one term containing t_a and one term without t_a. That is,

$$Z_{ij}(t,x) = l_{ij}(t_{-a}; x) + t_a r_{ij}(t_{-a}; x), \tag{4}$$

[2] While there is a simpler presentation for the algorithm given below for this simple case, we present it in a way that will be useful for the general case.

[3] We emphasize that the variables are only introduced for tracking the terms in the proof. In particular, we will only assign these variables the value 1 in the proof.

where both expressions $l_{ij}(t_{-a};x), r_{ij}(t_{-a};x)$ are independent of t_a (and can be 0). Moreover, by the definition of Z_{ij}, l_{ij} depends only on variables $x_{j'}$ that belong to the same cluster as F_i (that is, $c_a(x_{j'}) = c_a(F_i)$), while the variables of r_{ij} belong to the cluster of $c_a(x_j)$ (that is, $c_a(x_{j'}) = c_a(x_j)$). We prove the following claim.

Claim 2.

$$\prod_{i\in[m]} F_i\big((Z_{ij}(t,x))_{j\in V(i)}\big) = \sum_{J\subseteq I} t_a^{|J|} \prod_{i\in[m]} \left(q_{i,J_i}(t_{-a},x) \prod_{j\in J_i} r_{ij}(t_{-a},x) \right)$$

where q_{i,J_i} depends only on variables $x_{j'}$ with $c_a(x_{j'}) = c_a(F_i)$ and r_{ij} depends only on variables $x_{j'}$ with $c_a(x_{j'}) = c_a(x_j)$. Moreover, given a set $J \subseteq I$, the corresponding product can be constructed efficiently.

We will use the following simple fact in the proof.

Fact 3. *Let p be a multilinear polynomial in k variables. Then $p(a_1+b_1,\ldots,a_k+b_k)$ can be written as*

$$\sum_{S\subseteq[k]} c_S \prod_{i\in S} b_i$$

where c_S depends only on $a_1,\ldots,a_k$ and on the coefficients of p.

Proof of Claim 2. For each $i \in [m]$, we expand the expression

$$F_i\big((Z_{ij}(t,x))_{j\in V(i)}\big) = F_i\big((l_{ij}(t_{-a};x) + t_a r_{ij}(t_{-a};x))_{j\in V(i)}\big).$$

Using Fact 3 we get that

$$F_i((l_{ij}+t_a r_{ij})_{j\in V(i)}) = \sum_{B\subseteq V(i)} q_{i,B} \prod_{j\in B} t_a r_{ij} = \sum_{B\subseteq V(i)} t_a^{|B|} q_{i,B} \prod_{j\in B} r_{ij}.$$

where $q_{i,B}$ depends only on l_{ij} and the coefficients of F_i. Note that we can compute the above form efficiently since $F_i(X)$ depends on at most 3 variables. We therefore have

$$\begin{aligned}\prod_{i\in[m]} F_i\big((Z_{ij}(t,x))_{j\in V(i)}\big)) &= \prod_{i\in[m]} \sum_{B_i\subseteq V(i)} t_a^{|B_i|} q_{i,B_i} \prod_{j\in B_i} r_{ij} \\ &= \sum_{\{B_i\subseteq V(i)\}_{i\in[m]}} \prod_{i\in[m]} t_a^{|B_i|} q_{i,B_i} \prod_{j\in B_i} r_{ij}\end{aligned}$$

Recall that I is the set of incidences of φ, and for a set $J \subseteq I$ let $J_i = \{j \in V(i) : (i,j) \in J\}$. Then we have

$$\begin{aligned}\prod_{i\in[m]} F_i\big((Z_{ij}(t,x))_{j\in V(i)}\big)) &= \sum_{J\subseteq I} \prod_{i\in[m]} t_a^{|J_i|} q_{i,J_i} \prod_{j\in J_i} r_{ij} \\ &= \sum_{J\subseteq I} t_a^{|J|} \prod_{i\in[m]} q_{i,J_i} \prod_{j\in J_i} r_{ij}.\end{aligned}$$

□

Claim 2 implies that

$$\Phi(t) = \sum_{x\in\{0,1\}^N} \prod_{i\in[m]} F_i\big((Z_{ij}(t,x))_{j\in V(i)}\big)$$
$$= \sum_{J\subseteq I} t_a^{|J|} \left(\sum_{x\in\{0,1\}^N} \prod_{i\in[m]} q_{i,J_i} \prod_{j\in J_i} r_{ij} \right).$$

Observe that in each product $\prod_{i\in[m]} q_{i,J_i} \prod_{j\in J_i} r_{ij}$ in the above sum, each term depends on variables $x_{j'}$ in a single cluster $c_a(x_{j'})$ (q_{i,J_i} depends only on variables with $c_a(x_{j'}) = c_a(F_i)$, and r_{ij} only on variables with $c_a(x_{j'}) = c_a(x_j)$). Thus we can write

$$\Phi(t) = \sum_{J\subseteq I} t_a^{|J|} \left(\sum_{x\in\{0,1\}^N} \left(\prod_{s=1}^{g} W_s^J(x_{V_s}) \right) \right).$$

where W_s^J depends only on variables $x_{j'}$ with $c_a(x_{j'}) = s$, as desired. However, we sum over 2^{3N} possible subsets J of I, which is too much. Instead, in the following we will consider the partial sum

$$M_a(t) = \sum_{\substack{J\subseteq I \\ |J|\le\kappa}} t_a^{|J|} \left(\sum_{x\in\{0,1\}^N} \left(\prod_{s=1}^{g} W_s^J(x_{V_s}) \right) \right)$$

that contains all of the monomials of Φ with t_a-degree at most $\kappa = 3\delta N$.

Remark 4 (Generalization of the above algorithm)**.** *The above step in the algorithm applies whenever each input of the predicate F_i has the form*

$$Z_{ij} = \ell_{ij} + t_a r_{ij},$$

where ℓ_{ij} depends only on variables $x_{j'}$ with $c_a(x_{j'}) = c_a(F_i)$, and r_{ij} depends only on variables $x_{j'}$ with $c_a(x_{j'}) = c_a(x_j)$.

We may also consider computing a weighted sum

$$\sum_{x\in\{0,1\}^N} \chi(x) \prod_{i\in[m]} F_i\big((Z_{ij}(t,x))_{j\in V(i)}\big). \tag{5}$$

If, for every coordinate $a \in [u]$, we can write $\chi(x)$ as $\chi(x) = \sum_{\gamma\in\Gamma} \prod_{s=1}^{g} \chi_{a,s}^{\gamma}(x)$, where $\chi_{a,s}^{\gamma}(x)$ depends only on variables $x_{j'}$ with $c_a(x_{j'}) = s$, then we can compute this weighted sum using the same approach, with an additional $|\Gamma|$ factor in the running time. We explain more about it in Section 4.

2.3 Step III: Evaluating the Sum

We next compute the coefficients (as polynomials in t_{-a})

$$\left(\sum_{x\in\{0,1\}^N} \left(\prod_{s=1}^{g} W_s^J(x_{V_s}) \right) \right) = \prod_{s=1}^{g} \left(\sum_{y\in\{0,1\}^{|V_s|}} \big(W_s^J(y)\big) \right)$$

for every $|J| \le \kappa$, and simplify $M_a(t)$ to get a polynomial in t with integer coefficients. This polynomial contains all of the monomials of Φ with t_a degree at most κ.

2.4 Step IV: Removing Duplicate Monomials and Reconstructing the Output

We now have for each coordinate a a polynomial M_a with integer coefficients that contains all of the monomials of $\Phi(t)$ with t_a-degree at most κ. Recall that every monomial of $\Phi(t)$ has at least one variable with degree at most κ, thus every monomial appears at least in one polynomial M_a. However some monomials can appear in more than one polynomial. We therefore sum up all of the polynomials M_a, while removing monomials that appear more than once (keeping one copy of each). This sum is equal to $\Phi(t)$. Finally, we simply evaluate $\Phi(1,\ldots,1)$ to get the value of S.

2.5 Running time

First, we bound the number of t-monomials. Notice that in $\Phi(t)$, the t_a-degree of every monomial is at most $3N$, for any a. Thus, there are at most $(3N+1)^u = \mathsf{poly}(N)$ different monomials. Next, for each fixing of J and a coordinate a, the computation of

$$\prod_{s=1}^{g}\left(\sum_{y\in\{0,1\}^{|V_s|}}\left(W_s^J(y)\right)\right)$$

can be done in time $2^{N/g}\mathsf{poly}(N)$. There are u possible values of a, and

$$\sum_{k=0}^{\kappa}\binom{3N}{k} \leq (\kappa+1)(e/\delta)^{3\delta N} = (\kappa+1)2^{3\delta\log(e/\delta)N}.$$

possible values of J. Overall, we get that the running time is at most

$$2^{3\delta\log(e/\delta)N+N/g}\mathsf{poly}(N).$$

We can now get running time $2^{\epsilon N}$ for any constant ϵ by choosing δ to be small enough and g to be large enough.

3 Geographic Information Retrieval

We next move to the general case, where there is no placement of the variables and the predicates in the space in a way that each predicate only depends on variables that are close to it. Instead, we will use the following access structure, that provides a way to assign addresses for each variable and predicate such that such a placement will be possible, at the cost of increasing the number of variables that each predicate depends on, and introducing a public short hint.

Below, we consider bipartite graphs $G = (V = (P, X), E)$, where $P = \{F_1, \ldots, F_m\}$ is the set of predicates (or readers) and $X = \{X_1, \ldots, X_N\}$ is the set of variables (or dataset entries), where each node X_j has a value x_j and there is an edge (F_i, X_j) if F_i depends on x_j. We say that G has degree-d if each vertex of G has at most d edges.

Definition 5 (Geographic Information Retrieval)**.** Let $G = (V = (P, X), E)$ be a bipartite graph. Let $\delta \in (0, 1/2)$ be a constant and $g \geq 2$, $u \geq 1$ be integers. A (δ, g, u)-Geographic Information Retrieval ((δ, g, u)-GIR) for G is a data structure and address placement $c : V \to [g]^u$ with the following properties.

1. **Advice.** There is a linear advice $h(x)$ that its output can be encoded using δN bits.

2. **Decoding.** There is a decoding algorithm Dec such that for each variable x_j and each of the predicates F_i with $(F_i, X_j) \in E$, $\mathsf{Dec}^x(j, i, h(x)) = x_j$. Moreover, for each such i, j, there is a function λ and coefficients $\mu_{j'}$, such that

$$\mathsf{Dec}^x(j, i, h(x)) = \lambda(h(x)) + \sum_{j' \in [N]} \mu_{j'} x_{j'}. \tag{6}$$

3. **Addresses.** The placement function satisfies the following:

   (i) **Balanced.** For each coordinate $a \in [u]$ and value $s \in [g]$, the set $V_{a,s}$ of variables x_j with $c_a(x_j) = s$ has size at most $N/g + O(1)$.

   (ii) **Closeness.** For each edge $(F_i, X_j) \in E$, if the decoding $\mathsf{Dec}^x(j, i, h(x))$ depends on $x_{j'}$ (that is, $\mu_{j'} \neq 0$), then j' is close to v: $d_H(c(F_i), c(x_{j'})) \leq \delta u$.

We say that the GIR is 2-local, if in addition it has the following properties:

1. The advice can be partitioned into r blocks, $h(x) = (h_C(x))_{C \in [r]}$, such that each h_C can be encoded using b bits such that $rb \leq \delta N$. Moreover, for every a and $C \in [r]$, the block $h_C(x)$ depends on variables x_j with at most two different values of $c_a(x_j)$.

2. If for some edge $(F_i, x_j) \in E$ the coefficient $\mu_{j'} \neq 0$ (that is, we read $x_{j'}$ to decode x_j using predicate F_i), then, for every a, $c_a(x_{j'}) \in \{c_a(F_i), c_a(x_j)\}$.

The GIR is computable in polynomial time if given a graph G the advice, decoding coefficients and the addresses can be found in polynomial time in N.

Recall that we want that each predicate F_i to depend only on variables that are close to it. In the following, we will replace each variable x_j in the predicate F_i with its decoding using the data structure (note that the decoder may depend on the predicate F_i, and not only on the variable x_j we want to decode).

Theorem 6. *For every constants δ and g there is a constant u such that there is a 2-local (δ, g, u)-GIR computable in polynomial time for any degree-3 bipartite graph G.*

4 Counting from GIR

In this part we prove Theorem 1. Fix $\delta > 0$ and $g \geq 2$. Let G be the graph of dependencies of the formula φ, where there is a node F_i for every predicate and a node x_i for every variable, and an edge (F_i, x_i) if x_i appears in F_i. Fix a 2-local (δ, g, u)-GIR for G. Let $\{F_i\}_{i \in [m]}$ be the set of multilinear polynomials. We want to compute

$$S = \sum_{x \in \{0,1\}^N} \prod_{i \in [m]} F_i(x_{V(i)}).$$

The algorithm is similar to the toy case, except we decode each variable using the GIR, and need to deal with the advice. We next explain these changes.

4.1 Step I: Fixing the Advice.

We would like to replace each occurrence of x_j in F_i by its decoder. The obstacle is that the decoder also needs the advice $h(x)$. We therefore first sum over each possible value of the advice. Let Γ be the image set of h_C and $\eta = (\eta_C)_{C\in[r]} \in \Gamma^r$ be a full advice. Since for every x there is one valid advice, we have that

$$S = \sum_{\eta\in\Gamma^r} \sum_{x\in\{0,1\}^N} \prod_{i\in[m]} F_i(x_{V(i)}) \prod_{C=1}^{r} \mathbf{1}[h_C(x) = \eta_C].^4 \tag{7}$$

In the following we fix an advice η, and show how to compute the value of

$$S^\eta = \sum_{x\in\{0,1\}^N} \prod_{i\in[m]} F_i(x_{V(i)}) \prod_{C=1}^{r} \mathbf{1}[h_C(x) = \eta_C]. \tag{8}$$

Later, we will sum over all possible η to get $S = \sum_\eta S^\eta$.

4.2 Step II: Decoding and Marking the Variables

Next, we replace each variable with its decoding using the GIR. Recall that by the decoding of the GIR, for every variable x_j in a predicate F_i there are coefficients $\lambda, \{\mu_{j'}\}$ such that for every x with $h(x) = \eta$, we have $x_j = \mathsf{Dec}^x(j, i, h(x)) = \lambda(\eta) + \sum_{j'\in[N]} \mu_{j'} x_{j'}$.

For every coordinate $a \in [u]$, we would like the cluster $c_a(x_{j'})$ of the variables $x_{j'}$ to be the same as $c_a(F_i)$. As in the toy case, we introduce u variables $t_1, \ldots, t_u$, and multiply $x_{j'}$ by t_a if the cluster of F_i and $x_{j'}$ are different. That is, define the set

$$A(i, j') = \{a \in [u] : c_a(x_{j'}) \neq c_a(F_i)\},$$

and let

$$Z^\eta_{ij}(t, x) = \lambda(\eta) + \sum_{j'\in[N]} \mu_{j'} x_{j'} \prod_{a\in A(i,j')} t_a. \tag{9}$$

Notice that by the promise of the GIR we have that $|A(i, j')| = d_H(c(F_i), c(x_{j'})) \leq \delta u$ (unless $\mu_{j'} = 0$), and thus the t-degree of Z^η_{ij} is at most δu. Moreover, by the above, $Z^\eta_{ij}(\mathbf{1}, x) = x_j$ (when $h(x) = \eta$). Define the polynomial

$$\Phi^\eta(t) = \sum_{x\in\{0,1\}^N} \prod_{i\in[m]} F_i\big((Z^\eta_{ij}(t, x))_{j\in V(i)}\big) \prod_{C=1}^{r} \mathbf{1}[h_C(x) = \eta_C]. \tag{10}$$

Then we have that $\Phi^\eta(\mathbf{1}) = S^\eta$. Since F_i is multilinear and each of the $3N$ expression Z^η_{ij} appears once in Φ^η, we get that the degree of Φ^η is at most $3Nu\delta$. Since there are u variables, this implies that for every monomial there is at least one variable t_a with degree at most $3\delta N$.

[4] Importantly we do not assume that $\mathbf{1}[h_C(x) = \eta_C]$ can be written as a polynomial.

4.3 Step III: Fixing a Coordinate and Splitting the Advice

Next, fix $a \in [u]$. For every $s \in [g]$, let V_s be the set of variables x_j with $c_a(x_j) = s$. Fix an advice block C. By the 2-locality of the advice, h_C depends on variables x_j with at most two different values $c_a(x_j)$. Let s_0 and s_1 be these values. For $s \in [g]$, let x_s be x restricted to the variables x_j with $c_a(x_j) = s$, with all other variables set to zero. By linearity of the advice, we have

$$h_C(x) = h_C(x_{s_0}) + h_C(x_{s_1}).$$

Consequently, we can write

$$\mathbf{1}[h_C(x) = \eta_C] = \sum_{\gamma_C \in \Gamma} \mathbf{1}[h_C(x_{s_0}) = \gamma_C]\, \mathbf{1}[h_C(x_{s_1}) = \eta_C - \gamma_C],$$

where each indicator depends on variables x_j with only one value $c_a(x_j)$. We can thus replace the advice check $\prod_{C=1}^{r} \mathbf{1}[h_C(x) = \eta_C]$ with an expression of the form

$$\sum_{\gamma \in \Gamma^r} \left(\prod_{s=1}^{g} \chi_s^{\gamma}(x_{V_s}) \right),$$

where χ_s^{γ} depends only on variables x_j with $c_a(x_j) = s$. We conclude that

$$\Phi^{\eta}(t) = \sum_{\gamma \in \Gamma^r} \sum_{x \in \{0,1\}^N} \left(\prod_{s=1}^{g} \chi_s^{\gamma}(x_{V_s}) \right) \prod_{i \in [m]} F_i\big((Z_{ij}^{\eta}(t,x))_{j \in V(i)}\big)).$$

4.4 Step IV: Expanding the Product According to One Coordinate

This is step II of the toy case, with the generalization explained in Remark 4. We first separate $Z_{ij}^{\eta}(t,x)$ into terms that include t_a and terms without t_a. That is,

$$Z_{ij}^{\eta}(t,x) = l_{ij}^{\eta}(t_{-a};x) + t_a r_{ij}^{\eta}(t_{-a};x), \tag{11}$$

where both expressions $l_{ij}^{\eta}(t_{-a};x), r_{ij}^{\eta}(t_{-a};x)$ are independent of t_a. Observe that, by our construction and the 2-locality of the decoder, l_{ij}^{η} depends only on variables $x_{j'}$ that have the correct value $c_a(F_i) = c_a(x_{j'})$, while r_{ij}^{η} depends only on variables $x_{j'}$ with the same wrong value $c_a(x_{j'}) = c_a(x_j)$.

By Claim 2, we have that

$$\prod_{i \in [m]} F_i\big((Z_{ij}^{\eta}(t,x))_{j \in V(i)}\big) = \sum_{J \subseteq I} t_a^{|J|} \prod_{i \in [m]} \left(q_{i,J_i}^{\eta} \prod_{j \in J_i} r_{ij}^{\eta} \right).$$

where q_{i,J_i} depends only on variables $x_{j'}$ with $c_a(x_{j'}) = c_a(F_i)$ and r_{ij} depends only on variables $x_{j'}$ with $c_a(x_{j'}) = c_a(x_j)$. This implies that

$$\begin{aligned}
\Phi^{\eta}(t) &= \sum_{\gamma \in \Gamma^r} \sum_{x \in \{0,1\}^N} \left(\prod_{s=1}^{g} \chi_s^{\gamma}(x_{V_s}) \right) \prod_{i \in [m]} F_i\big((Z_{ij}^{\eta}(t,x))_{j \in V(i)}\big)) \\
&= \sum_{\gamma \in \Gamma^r} \sum_{J \subseteq I} t_a^{|J|} \left(\sum_{x \in \{0,1\}^N} \left(\prod_{s=1}^{g} \chi_s^{\gamma}(x_{V_s}) \right) \prod_{i \in [m]} q_{i,J_i}^{\eta} \prod_{j \in J_i} r_{ij}^{\eta} \right).
\end{aligned}$$

As in the toy case, in each product in the above sum, each term depends on variables $x_{j'}$ in a single cluster $c_a(x_{j'})$ (q_{i,J_i} depends only on $x_{j'}$ with $c_a(x_{j'}) = c_a(F_i)$, and r^{η}_{ij} only on $x_{j'}$ with $c_a(x_{j'}) = c_a(x_j)$). Thus we can write

$$\Phi^{\eta}(t) = \sum_{\gamma\in\Gamma^r} \sum_{J\subseteq I} t_a^{|J|} \left(\sum_{x\in\{0,1\}^N} \left(\prod_{s=1}^{g} W_s^{J,\gamma}(x_{V_s}) \right) \right).$$

where $W_s^{J,\gamma}$ depends only on variables $x_{j'}$ with $c_a(x_{j'}) = s$, as desired. Lastly, instead of summing over 2^{3N} possible $J \subseteq I$, we consider the partial sum

$$M_a(t) = \sum_{\gamma\in\Gamma^r} \sum_{\substack{J\subseteq I \\ |J|\leq\kappa}} t_a^{|J|} \left(\sum_{x\in\{0,1\}^N} \left(\prod_{s=1}^{g} W_s^{J,\gamma}(x_{V_s}) \right) \right)$$

that contains all of the monomials of Φ^{η} with t_a-degree at most $\kappa = 3\delta N$.

4.5 Step V: Evaluating the Sum and Removing Duplicate Monomials

We next repeat steps III and IV of the toy case. We compute the coefficients (as polynomials in t)

$$\left(\sum_{x\in\{0,1\}^N} \left(\prod_{s=1}^{g} W_s^{J,\gamma}(x_{V_s}) \right) \right) = \prod_{s=1}^{g} \left(\sum_{y\in\{0,1\}^{|V_s|}} \left(W_s^{J,\gamma}(y) \right) \right)$$

for every $|J| \leq \kappa$ and $\gamma \in \Gamma^r$, and simplify $M_a(t)$ to get a polynomial in t with numerical coefficients. This polynomial contains all of the monomials of Φ^{η} with t_a degree at most κ. Since every monomial of $\Phi^{\eta}(t)$ has at least one variable with degree at most κ, every monomial appears at least in one polynomial M_a, and we can reconstruct $\Phi^{\eta}(t)$. Finally, we simply evaluate $\Phi^{\eta}(1,\ldots,1)$ to get the value of S^{η}. We output $S = \sum_{\eta\in\Gamma^r} S^{\eta}$.

4.6 Running time

First, the number of t-monomials is at most $(3N+1)^u = \mathsf{poly}(N)$. Next, for each fixing of η, γ, J and a coordinate a, the computation of $\prod_{s=1}^{g} \left(\sum_{y\in\{0,1\}^{|V_s|}} \left(W_s^{J,\gamma}(y) \right) \right)$ can be done in time $2^{N/g}\mathsf{poly}(N)$. Next, there are at most

$$|\Gamma|^r \leq 2^{rb} \leq 2^{\delta N} \tag{12}$$

possible values of η, and a similar bound holds for γ. There are u possible values of a, and

$$\sum_{k=0}^{\kappa} \binom{3N}{k} \leq (\kappa+1)(e/\delta)^{3\delta N} = (\kappa+1)2^{3\delta\log(e/\delta)N}$$

possible values of J. Overall, we get that the running time is at most $2^{2\delta N + 3\delta\log(e/\delta)N + N/g}\mathsf{poly}(N)$.

Proof of Theorem 1. Given $\epsilon > 0$, choose a rational $\delta \in (0, 1/2)$ sufficiently small that, with $g = \lceil 1/\delta \rceil$,

$$2\delta + 1/g + 3\delta\log(e/\delta) < \epsilon.$$

The proof follows by the above algorithm together with Theorem 6.

□

5 Constructing GIR

To prove Theorem 6, we will need a primitive called Robust Information Retrieval.[5]

Definition 7 (Robust information retrieval)**.** Let $u, L \geq 1$, $0 < \delta < 1$, and let $\sigma\colon [L] \to \{0,1\}^u$ be an address map. For $i \in [L]$, let

$$P_i = \{j \in [L] : d_{\mathrm{H}}(\sigma(i), \sigma(j)) \geq (1-\delta)u\}$$

be the set of positions that are *far* for i. A (u, δ)-*RIR scheme* for the address map σ consists of a hint $h\colon \{0,1\}^L \to \mathbb{Q}^\ell$ that can be encoded using δL bits and decoders $\mathrm{Dec}_i\colon \mathbb{Q}^\ell \times \{0,1\}^{P_i} \to \mathbb{Q}$ for $i \in [L]$, such that

$$\mathrm{Dec}_i\big(h(D),\, D|_{P_i}\big) = D_i \qquad (D \in \{0,1\}^L,\ i \in [L]). \tag{13}$$

The scheme is *linear* if h and Dec_i are linear over $\mathbb{Q}$.

That is, to decode a coordinate D_i the decoder only reads coordinates with address that are far from $\sigma(i)$.

Theorem 8 (Robust information retrieval)**.** *For every $\delta \in (0, 1/2)$, there exist integers $u, L \geq 1$ with an explicit linear (u, δ)-RIR scheme for an address map $\sigma\colon [L] \to \{0,1\}^u$.*

Proof. Fix $\delta \in (0, 1/2)$. Let k be a sufficiently large integer, and $q = \lfloor \delta k \rfloor$. Set $d = k - q - 1$ and $u = 2k$. The database positions are indexed by the subsets of $[2k]$ of size k. Thus,

$$L = \binom{2k}{k}$$

The address of an entry S is its indicator vector $\sigma(S) \in \{0,1\}^{2k}$.

The hint. For every $U \subseteq [2k]$, let

$$h_U(D) = \sum_{\substack{T \in \binom{[2k]}{k} \\ T \cap U = \emptyset}} D_T.$$

The hint contains h_U for every $|U| \leq d$.

Decoding. Fix a position $S \in \binom{[2k]}{k}$. We first claim that

$$D_S = \sum_{U \subseteq S} (-1)^{|U|} h_U(D) = \sum_{U \subseteq S} (-1)^{|U|} \sum_{\substack{T \in \binom{[2k]}{k} \\ T \cap U = \emptyset}} D_T.$$

Indeed, every set $|T| = k$ appears in the above sum once for each $U \subseteq S \setminus T$, with coefficient $(-1)^{|U|}$. Since every non-empty set have the same number of even and odd-sized subsets, D_T is canceled from the sum whenever $S \setminus T$ is non-empty. Since S and T are of the same size, $S \setminus T$ is empty iff $S = T$. In this case D_S appears in the sum once, with coefficient $(-1)^0 = 1$.

[5] The notion of robust information retrieval and its construction are strongly connected to index coding [BYBJK06, Hav18].

Next, to compute $\sum_{U \subseteq S}(-1)^{|U|}h_U(D)$ we need to compute $h_U(D)$ for every $U \subseteq S$. For every such U with $|U| \leq d$, we can read $h_U(D)$ from the advice. We claim that for every $|U| > d$ we can compute h_U using only entries that are far from S.

Indeed, $h_U(D)$ depends only on sets T that do not intersect with U. Thus, $|S \setminus T| > d$. Since both S and T are of size k, we have that

$$d_H(\sigma(S), \sigma(T)) \geq 2(d+1) = 2(k-q) \geq (1-\delta)u.$$

The length of the advice. The number of sets U in the advice is

$$\ell = \sum_{j=0}^{d=k-\delta k-1} \binom{2k}{j} \leq 2^{2k} \cdot e^{-\delta^2 k/2}.$$

where the inequality follows by Chernoff. Since the database values are binary, each hint coordinate takes a value in $\{0, 1, \ldots, L\}$. Therefore, we can encode the advice using $\ell \log(L+1) \leq \ell(2k+1)$ bits. Since $L = \binom{2k}{k} \geq 2^{2k}/(2k+1)$, we have

$$\frac{\ell(2k+1)}{L} \leq (2k+1)^2 e^{-\delta^2 k/2}$$

which is smaller than δ for large enough k. □

5.1 Constructing GIR

Finally, we use the RIR to construct 2-local GIR.

Proof of Theorem 6. Fix $\delta \in (0, 1/2)$ and $g \geq 2$. Let $\mathcal{R}$ be a linear $(u, \delta/4)$-RIR scheme with database length L and addresses in $\{0,1\}^u$, promised by Theorem 8.

Fix a bipartite graph $G = (V = (P, X), E)$, where $P = \{F_1, \ldots, F_m\}$ is the set of predicates and $X = \{X_1, \ldots, X_N\}$ is the set of variables, and where the degree of each vertex in the graph is at most 3.

In the following, we will construct an access structure D with (at most) $3N$ entries, one for each edge $e = (F_i, X_j) \in E$, with value $D_e = x_j$, and we will have a single decoding function for each entry e. Our GIR construction (Theorem 6) follows by setting $\mathsf{Dec}^x(j, i, h(x))$ to be the decoding procedure of the entry D_e in our database below.

The construction. Recall that the graph has at most $3N$ edges. We will take $r = 3N/L + O(1)$ copies of $\mathcal{R}$, and distribute the edges among their database positions. Each edge $e = (F_i, X_j)$ will be stored with the value x_j, and unused positions will contain zero. Each copy C of the RIR has an advice block $h_C(x) = h(D_C(x))$ that can be encoded into $b \leq \delta/4L$ bits. Thus, the total advice length satisfies

$$rb \leq r \cdot L\delta/4 \leq \delta N$$

for a large enough N. Moreover, the RIR decoders immediately give linear decoders for all of the edges. It remains to choose the addresses of the vertices, and to distribute the edges among the entries of the copies of the RIR.

Choosing addresses for the entries of the RIR. Recall that each RIR entry $q \in [L]$ has a binary address $\sigma(q)$, and we consider many copies of the RIR. On the other hand, each edge $e = (F_i, X_j)$ has an address for F_i and an address for X_j in $[g]^u$. We first assign each of the RIR entries a pair of addresses in $[g]^u$ it is *compatible* with.

That is, we associate with each RIR copy a pair of addresses $(\lambda^0, \lambda^1) \in [g]^u$. We say that an edge $e = (F_i, X_j)$ with coloring $(c(F_i), c(X_j))$ is *compatible* with entry q of this RIR copy, if for every coordinate $a \in [u]$, it holds that $c_a(X_j) = \lambda_a^b$ and $c_a(F_i) = \lambda_a^{1-b}$, where $b = \sigma_a(q)$.

Assume that we assign to each RIR entry an edge $e = (F_i, X_j)$ which is compatible with it. Then we claim that the construction fulfills the coloring requirements of the GIR. Indeed, fix two RIR entries q and q' containing edges $e = (F_i, X_j)$ and $e' = (F_{i'}, X_{j'})$, respectively. Assume we want to decode the value of X_j (i.e., the value of entry q in the RIR) using the RIR decoder, and the decoder reads the value of entry q'. By the above construction, it holds that if the a-th bit of the address $\sigma_a(q)$ is different from $\sigma_a(q')$, then $c_a(X_{j'}) = c_a(F_i)$, as desired. On the other hand, if $\sigma_a(q) = \sigma_a(q')$, then it holds that $c_a(X_{j'}) = c_a(X_j)$. By the RIR farness property, the latter happens in at most δu coordinates a, which means that $d_H(c(F_i), c(X_{j'})) \leq \delta u$.

Furthermore, each advice block depends on variables x_j of at most two values of $c_a(X_j)$, one for each half of its copy. Overall, we get that if we are able to assign for each RIR entry an edge (F_i, X_j) which is compatible with it, then we are done.

Providing enough positions. Let $K = g^u$. There are K^2 possible pairs (λ^0, λ^1). If we take one copy of $\mathcal{R}$ for each such pair, then every edge $e = (F_i, X_j)$ with address sequences $(c(F_i), c(X_j)) \in ([g]^u)^2$, has L compatible positions. Indeed, for every entry $q \in [L]$ there is exactly one pair (λ^0, λ^1) for which

$$\lambda_a^{\sigma(q)_a} = \tau_a, \qquad \lambda_a^{1-\sigma(q)_a} = \omega_a$$

holds for every $a \in [u]$, and thus there is one RIR copy in which the q-th entry is compatible with e. Thus, each of the L base positions contributes exactly one compatible position for every pair (ω, τ).

We call these K^2 copies a block. Each block therefore provides exactly L positions for every possible pair of addresses. If we repeat the block ρ times, we obtain ρL compatible positions for each pair, using $r = \rho K^2$ copies in total.

Balancing the Addresses. We now need to assign the addresses $c(F_i)$ and $c(X_j)$ to the vertices of the graph, such that every pair $(c(F_i), c(X_j))$ of addresses will occur on roughly $3N/K^2 + O(1)$ edges. In this case we can take $\rho = 3N/K^2L + O(1)$, which means we need $3N/L + O(1)$ RIR copies to assign every incidence.

Observe that if we independently assign to each vertex a uniformly random address from $[g]^u$, then for every fixed pair (ω, τ) of addresses, the expected number of edges satisfies $c(F_i) = \omega$ and $c(X_j) = \tau$ would be $\frac{|E|}{K^2} \leq \frac{3N}{K^2}$, as desired. One can show using a concentration bound that with high probability the number of incidences with this particular address is close to the the expectation, which implies that a random placement is good with high probability. In the following we show how to get it deterministically.[6]

We start by assuming that K divides $|X| = N$. We first assign addresses to the vertices in X. Assign each address in $[g]^u$ evenly among the vertices X_j, such that N/K vertices assigned to the same sequence.

[6] An alternative way is to use the method of conditional expectations to derandomize this construction.

Next, define the *type* of a vertex in P to be the multiset of the addresses assigned to its neighboring vertices. Since the degree of vertices in P is at most 3 and K is constant, there are only constantly many types. For each type T let n_T be the number of vertices of this type. Assume again that K divides n_T, and assign each address evenly among the n_T vertices, such that n_T/K vertices assigned to the same address. By symmetry, it is clear that in this case the number of edges $e = (F_i, X_j)$ satisfies $c(F_i) = \omega$ and $c(X_j) = \tau$ is at most $\frac{3N}{K^2}$ for every fixed pair (ω, τ).

To remove the assumption that K divides N and n_T, we simply add dummy vertices of each type to make this assumption hold. That is, first we add $< K$ dummy vertices to X to make N divisible by K. Next, for each vertex type T, we add $< K$ dummy vertices of this type to make n_T divisible by K. We assign addresses to the vertices as before, and then throw away the dummy vertices before embedding the edges in the RIR. Since K and the number of types are constant, we only added a constant many edges, making the number of RIR copies we need to grow by a constant. □

Remark 9 (Optimizing the running time)**.** *Our analysis can also yield an explicit subexponential running time by allowing δ and g to depend on N. In particular, choosing $\delta = C\frac{\log\log N}{\sqrt{\log N}}$ and $g = \lceil 1/\delta \rceil$, for a sufficiently large constant C, gives a running time of $2^{O\left(N(\log\log N)^2/\sqrt{\log N}\right)}$, with an explicit accounting of the construction overhead.*

6 Solving SAT

Lemma 10 (3-CNF to bounded-occurrence 3-CNF)**.** *A 3-CNF with $m \geq 1$ clauses can be transformed in polynomial time into a 3-CNF where each variable appears at most 3 times, with $N \leq 3m$ variables and the same number of satisfying assignments.*

Proof. We construct the formula as follows. For each variable x, we replace each occurrence of x with a new variable x_i, and add additional clauses that force the copies of x, $x_1, \ldots, x_o$, to agree. For this, for every variable with $o \geq 2$ occurrences, we add clauses of the form $(\neg x_i \vee x_{i+1})$ for each $i \in [o]$, where $x_{o+1} = x_1$. □

Corollary 11 (Counting by clauses)**.** *For every constant $\epsilon > 0$, the satisfying assignments of a 3-CNF with n variables and m clauses can be counted in time $2^{\epsilon m}\mathsf{poly}(n+m)$.*

Proof. Immediate by Lemma 10 and Theorem 1. □

Corollary 12 (ETH is false, [Ope26, Cla26])**.** *For every fixed $\epsilon > 0$, 3-SAT on n variables and m clauses can be decided in time $2^{\epsilon n}\mathsf{poly}(n+m)$. In particular, ETH is false.*

Proof. Using the sparsification lemma (Lemma 13) we can write the formula as a disjunction of at most $2^{\epsilon n/2}$ 3-CNFs, each with at most $O(n)$ clauses, in time $2^{\epsilon n/2}\mathsf{poly}(n+m)$. By Corollary 11 we can decide each formula in subexponential time. □

AI Statement. OpenAI ChatGPT (Astra) was used extensively to study [Ope26, Cla26], learn about the problem, and explore candidate constructions. While the main ideas underlying the constructions and abstractions are due to the authors, ChatGPT was invaluable in exploring definitions and proof approaches, and in formalizing proof ideas and sketches. ChatGPT was also used to assist with writing and to provide feedback on the manuscript and its presentation.

References

[BYBJK06] Ziv Bar-Yossef, Yitzhak Birk, T. S. Jayram, and Tomer Kol. Index coding with side information. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 197–206, 2006.

[Cla26] Claude (Anthropic). A $2^{o(n)}$ time algorithm for 3-SAT, 2026. Preprint dated October 26, 2026; posted October 27, 2026. Hexagon:2610.00577v1.

[Hav18] Ishay Haviv. On minrank and the Lovász theta function. In *APPROX/RANDOM 2018*, volume 116 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:15, 2018.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[Ope26] OpenAI. The exponential time hypothesis is false, 2026. Preprint, revised October 25, 2026.

A 3SAT Sparsification

For completeness, in this part we prove the sparsification lemma of [IPZ01].

Lemma 13 (Sparsification for 3-SAT, [IPZ01])**.** *The following holds for any constant $\varepsilon > 0$. There exists an algorithm that given a 3-CNF formula φ on N variables, outputs $s \leq 2^{\varepsilon N}$ 3-CNF formulas $\varphi_1, \ldots, \varphi_s$ on the same set of variables in time $2^{\varepsilon N} \operatorname{poly}(N + |\varphi|)$ such that*

$$\varphi \equiv \bigvee_{i=1}^{s} \varphi_i,$$

and each φ_i has $O(N)$ clauses. In particular, φ is satisfiable if and only if some φ_i is satisfiable.

Proof. Fix a 3SAT formula φ. We say that φ is simple if it does not contain clauses which are tautologies, any duplicated clauses, or a clause which contains another clause. Given a 3SAT formula we can simplify it in polynomial time, and thus we can assume without loss of generality that φ is simple (by removing tautologies and duplications, and if a clause is contained in another, larger clause, by removing the larger one).

Fix an integer $t \geq 5$, to be chosen later. During the algorithm, we maintain a set $\mathcal{S}$ of 3SAT formulas, which is initialized to be $\mathcal{S} = \{\varphi\}$. If in some point during the run of the algorithm $\mathcal{S}$ contains only formulas with at most $4t^2N$ clauses, we output $\mathcal{S}$. Otherwise, we do the following for any $\varphi' \in \mathcal{S}$ with more than $4t^2N$ clauses:

1. If φ' contains $r > t$ distinct clauses of the form $(a \vee z_i)$ for literals $a, z_1, \ldots, z_r$, remove φ' from $\mathcal{S}$ and add the following two formulas after simplification: In the first, replace the clauses $(a \vee z_i)$ of φ' with the clause (a). In the second, replace each clause $(a \vee z_i)$ with the clause (z_i).

2. Otherwise, if φ' contains $r > t$ distinct clauses of the form $(a \vee b \vee z_i)$ for literals $a, b, z_1, \ldots, z_r$, remove φ' from $\mathcal{S}$ and add the following two formulas after simplification: In the first, replace the clauses $(a \vee b \vee z_i)$ of φ' with the clause $(a \vee b)$. In the second, replace each clause $(a \vee b \vee z_i)$ with the clause (z_i).

3. Otherwise, if φ' contains $r > t^2$ distinct clauses of the form $(a \vee y_i \vee z_i)$ for literals $a, y_1, \ldots, y_r, z_1, \ldots, z_r$, remove φ' from $\mathcal{S}$ and add the following two formulas after simplification: In the first, replace the clauses $(a \vee y_i \vee z_i)$ of φ' with the clause (a). In the second, replace each clause $(a \vee y_i \vee z_i)$ with the clause $(y_i \vee z_i)$.

We observe that all three rules preserve the correctness of the algorithm. We also observe that if for some φ' no rule applies, then each literal occurs in at most t binary clauses by step 1, in at most t^2 clauses of size 3 by step 3, and in at most one unit clause (since it is simple). This gives $2t^2$ occurrences per literal and $4t^2N$ clauses, which means that the algorithm only stops when all of the formulas in $\mathcal{S}$ are sparse.

It remains to bound the number of formulas in $\mathcal{S}$ when the algorithm stops. Since every of the above steps increases the size of $\mathcal{S}$ by one, this also bounds the running time. Consider the binary tree describing the execution of the algorithm, where the root is labeled by φ, and for every node in the tree labeled by a formula φ', the 2 child of the node are the two formulas obtained from φ' by the above algorithm. Leaves are formulas with at most $4t^2N$ clauses. Consider any root-to-leaf path.

Bounding the number of new clauses along the path. Every clause introduced by a step of the algorithm is a proper subset of one of its parent's clauses. Since its parent was simple before the step, no existing clause is contained in the new clause. Moreover, every new clauses that were introduced in the same node are distinct and have the same width. Thus, the new clauses are indeed new and survive simplification at that step.

Once a unit clause is introduced, it remains for the rest of the path. Thus, at most $2N$ unit clauses are introduced. To bound the number of binary clauses introduced, we mark each binary clause when it is introduced, leaving the original binary clauses of φ unmarked.

We claim that at every node in the tree each literal occurs in at most $2t$ marked clauses. Indeed, binary clauses are introduced only by step 2 and 3. Immediately before either step is applied, every literal occurs in at most t binary clauses, since otherwise, step 1 would have applied first. Step 2 introduces just one new binary clause. In Step 3, each literal b occurs in at most t of the newly introduced binary clauses. Indeed, otherwise more than t of the selected clauses of size 3 would contain both a and b, and Step 2 would have applied first. This proves the claim.

A marked binary clause disappears only when a new unit clause is introduced. By the above claim, when a new unit clause is introduced, at most $2t$ marked binary clauses disappear. So, at most $4tN$ marked clauses disappear along the path. We next observe that at most $2tN$ marked clauses survive at the leaf. Indeed, by the claim above each of the $2N$ literals appears in at most $2t$ such clauses, and each such clause contains two literals. Overall, at most $6tN$ binary clauses are introduced over the entire path.

Counting the leaves. Every step of the algorithm introduces at least one unit or binary clause, so the path has length at most $2N + 6tN \leq 7tN$. We next bound the number of times the path continues to the second child of some node (where each step of the algorithm yields

two children). In steps 1 and 2, taking the second child introduces at least t unit clauses. In step 3, it introduces more than t^2 binary clauses. Consequently, the total number of second-child choices is at most

$$\frac{2N}{t} + \frac{6tN}{t^2} = \frac{8N}{t}.$$

Finally, to bound the number of leaves s it is enough to bound the number of possible paths of length at most $7tN$, with at most $\frac{8N}{t}$ second-child choices. We get that

$$s \leq \sum_{j=0}^{\lfloor 8N/t \rfloor} \binom{7tN}{j} \leq 2^{O(N \log(t)/t)}.$$

We finally choose t such that $\log(t)/t$ is small enough compared to ε. □

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il