Counting t-cliques: Worst-case to average-case reductions and Direct interactive proof systems

Oded Goldreich* Guy N. Rothblum†

March 9, 2018

Abstract

We present two main results regarding the complexity of counting the number of t-cliques in a graph.

1. *A worst-case to average-case reduction*: We reduce counting t-cliques in any n-vertex graph to counting t-cliques in typical n-vertex graphs that are drawn from a simple distribution of min-entropy $\widetilde{\Omega}(n^2)$. For any constant t, the reduction runs in $\widetilde{O}(n^2)$-time, and yields a correct answer (w.h.p.) even when the "average-case solver" only succeeds with probability $1/\text{poly}(\log n)$.
2. *A direct interactive proof system*: We present a direct and simple interactive proof system for counting t-cliques in n-vertex graphs. The proof system uses $t-2$ rounds, the verifier runs in $\widetilde{O}(t^2 n^2)$-time, and the prover can be implemented in $\widetilde{O}(t^{O(1)} \cdot n^2)$-time when given oracle access to counting $(t-1)$-cliques in $\widetilde{O}(t^{O(1)} \cdot n)$-vertex graphs. This result extends also to varying $t = t(n)$, yielding alternative interactive proof systems for sets in $\#\mathcal{P}$.

The results are both obtained by considering weighted versions of the t-clique problem, where weights are assigned to vertices and/or to edges, and the weight of cliques is defined as the product of the corresponding weights. These weighted problems are shown to be easily reducible to the unweighted problem.

*Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel. oded.goldreich@weizmann.ac.il

†Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel. rothblum@alum.mit.edu

Contents

1 Introduction

We study two (seemingly unrelated) aspects of the complexity of counting t-cliques. The first study refers to the relation between the worst-case complexity of problems in $\mathcal{P}$ and their average-case complexity. The second study seeks (direct and intuitive) doubly-efficient interactive proof system for problems in $\mathcal{P}$. Indeed, both studies are related to the recently emerging theory of "hardness within $\mathcal{P}$" [34], and counting t-cliques in graphs is a good test case for both studies for several reasons:

1. Counting t-cliques in (n-vertex) graphs is a natural candidate for "hardness within $\mathcal{P}$" (i.e., it is in $\mathcal{P}$ and is assumed to have worst-case complexity $n^{\Theta(t)}$);
2. Counting t-cliques is a well-studied and natural problem; and
3. Counting t-cliques has an appealing combinatorial structure (which is indeed capitalized upon in our work).

In Sections 1.1 and 1.2 we discuss each study seperately, whereas Section 1.3 reveals the common themes that lead us to present these two studies in one paper.

1.1 Worst-case to average-case reductions

1.1.1 Background

While most research in the theory of computation refers to worst-case complexity, the importance of average-case complexity is widely recognized (cf., e.g., [17, Chap. 1–10.1] versus [17, Sec. 10.2]). Worst-case to average-case reductions, which allow for bridging the gap between the two theories, are of natural appeal (to say the least). Unfortunately, worst-case to average-case reductions are known only either for "very high" complexity classes, such as $\mathcal{E}$ and $\#\mathcal{P}$ (see [5] and [26, 11, 19][1], resp.), or for "very low" complexity classes, such as $\mathcal{AC}_0$ (cf. [3, 4]). In contrast, presenting a worst-case to average-case reduction for $\mathcal{NP}$ is a well-known open problem, which faces significant obstacles as articulated in [15, 10].

In the context of fine-grained complexity. A recent work by Ball, Rosen, Sabin, and Vasudevan [6] initiated the study of worst-case to average-case reductions in the context of fine-grained complexity.[2] The latter context focuses on the exact complexity of problems in $\mathcal{P}$ (see, e.g., the survey by V. Williams [34]), attempting to classify problems into classes of similar polynomial-time complexity (and distinguishing, say, linear-time from quadratic-time and cubic-time). Needless to say, reductions used in the context of fine-grained complexity must preserve the foregoing classification, and the simplest choice – taken in [6] – is to use almost linear-time reductions.

The pioneering paper of Ball *et al.* [6] shows that there exist (almost linear-time) reductions from the worst-case of several natural problems in $\mathcal{P}$, which are widely believed to be "somewhat

[1] The basic idea underlying the worst-case to average-case reduction of the "permanent" is due to Lipton [26], but his proof implicitly presumes that the field is somehow fixed as a function of the dimension. This issue was addressed independently by [11] and in the proceeding version of [19]. In the current work, we shall be faced with the very same issue.

[2] In retrospect, as will be discussed shortly below, some prior results can be reinterpreted as belonging to this setting.

hard" (i.e., have super-linear time complexity (in the worst case)), to the average-case of some other problems that are in $\mathcal{P}$. In particular, this is shown for the Orthogonal Vector problem, for the 3-SUM problem, and for the All Pairs Shortest Path problem. Hence, the worst-case complexity of problems that are widely believed to be "somewhat hard" is reduced to the average-case complexity of problems in $\mathcal{P}$. Furthermore, the worst-case complexity of the latter problems matches (approximately) the best algorithms known for the former problems (although the actual worst-case complexity of these problems may in fact be lower).

In our prior work [21], we tighten the foregoing result by defining, for each polynomial p, a worst-case complexity class $\mathcal{C}^{(p)}$ that is a subset of $\mathrm{Dtime}(p^{1+o(1)})$, and showing, for any problem Π in $\mathcal{C}^{(p)}$, an almost linear-time reduction from solving Π in the *worst-case* to solving a different problem $\Pi' \in \mathcal{C}^{(p)}$ in the *average-case*. Furthermore, we showed that $\mathcal{C}^{(p)}$ contains problems whose average-case complexity almost equals their worst-case complexity.

Loosely speaking, the class $\mathcal{C}^{(p)}$ consists of counting problems that refer to $p(n)$ local conditions regarding the n-bit long input, where each local condition refers to $n^{o(1)}$ bit locations and can be evaluated in $n^{o(1)}$-time. In particular, for any constant $t > 2$ and $p_t(n) = n^t$, the class $\mathcal{C}^{(p_t)}$ contains problems such as t-CLIQUE and t-SUM.

We emphasize that the foregoing results present worst-case to average-case reductions for *classes of problems* (i.e., reducing the worst-case complexity of one problem to the average-case complexity of another problem). Hence, the foregoing results leave open the question whether there exists an almost linear-time worst-case to average-case reduction for a *problem* in $\mathcal{P}$, let alone a natural problem of conjectured high worst-case complexity.

Worst-case to average-case reductions for individual problems. As stated in Footnote 2, some prior results can be reinterpreted as worst-case to average-case reductions for seemingly hard problem in $\mathcal{P}$. In particular, this holds for problems shown to be random self-reducible. An archetypical case, presented by Blum, Luby, and Rubinfeld [9], is that of matrix multiplication: the product AB is computed as $(A+R)(B+S)-(A+R)S-R(B+S)+RS$, where R and S are random matrices. Note, however, that in this case the potential hardness of the problem is quite modest (since the product of n-by-n matrices can be computed in time $o(n^{2.373})$).

A far less known case was presented by Goldreich and Wigderson [22], who considered the problem of computing the function $f_n(A_1,...,A_{\ell(n)}) = \sum_{S\subseteq[\ell(n)]} \mathtt{DET}(\sum_{i\in S} A_i)$, where the A_i's are n-by-n matrices over a finite field and $\ell(n) = O(\log n)$. They conjectured that this function cannot be computed in time $2^{\ell(n)/3}$, and showed that it is random self-reducible (by $O(n)$ queries).[3] Note, however, that the foregoing problem is not as well-studied as any of the problems considered in [6, 21], including counting t-cliques. In contrast, here we show a worst-case to average-case reduction for the problem of counting t-cliques.

1.1.2 Our results

In contrast to the results of [6, 21], which reduce the worst-case complexity of one problem in a subclass of $\mathcal{P}$ to the average-case complexity of a different problem in the same class, here we reduce the worst-case complexity of counting t-cliques to the average-case complexity of counting

[3] They also showed that it is downwards self-reducible when the field has the form $GF(2^{m(n)})$ such that $m(n) = 2^{\lceil \log_2 n \rceil}$ (or $m(n) = 2 \cdot 3^{\lceil \log_3 n \rceil}$).

t-cliques. Doing so, we show that the worst-case and average-case complexity of counting t-cliques are essentially equal.

Theorem 1.1 (worst-case to average-case reduction for counting cliques of given size): *For any constant t, there exists a simple distribution on n-vertex graphs and a $\widetilde{O}(n^2)$-time worst-case to average-case reduction of counting t-cliques in n-vertex graphs to counting t-cliques in graphs generated according to this distribution such that the reduction outputs the correct value with probability* $2/3$ *provided that the error rate* (of the average-case solver) *is a constant smaller than one fourth. Furthermore, the reduction makes* $\text{poly}(\log n)$ *queries, and the distribution $\mathcal{G}_n$ can be generated in $\widetilde{O}(n^2)$-time and is uniform on a set of* $\exp(\widetilde{\Omega}(n^2))$ *graphs.*

We obtain a similar reduction for the problem of counting (simple) t-cycles (for odd $t \geq 3$).

The notion of average-case complexity that underlies the foregoing discussion (and Theorem 1.1) refers to solving the problem on at least a 0.76 fraction of the instances. This notion may also be called *typical-case complexity*. A much more relaxed notion, called *rare-case complexity*, refers to solving the problem on a noticeable[4] fraction of the instances (say, on a $1/poly(\log n)$ fraction of the n-bit long instances).

Theorem 1.2 (worst-case to rare-case reduction for counting cliques): *For any constant t, there exists a simple distribution on n-vertex graphs and a $\widetilde{O}(n^2)$-time worst-case to rare-case reduction of counting t-cliques in n-vertex graphs to counting t-cliques in graphs generated according to this distribution such that the reduction outputs the correct value with probability* $2/3$ *provided that the success rate* (of the rare-case solver) *is at least* $1/\text{poly}(\log n)$. *Furthermore, the reduction makes $\widetilde{O}(n)$ queries, and the distribution $\mathcal{G}_n$ can be generated in $\widetilde{O}(n^2)$-time and is uniform on a set of* $\exp(\widetilde{\Omega}(n^2))$ *graphs.*

1.2 Doubly-efficient interactive proof systems

1.2.1 Background

The notion of interactive proof systems, put forward by Goldwasser, Micali, and Rackoff [24], and the demonstration of their power by Lund, Fortnow, Karloff, and Nisan [27] and Shamir [31] are among the most celebrated achievements of complexity theory. Recall that an interactive proof system for a set S is associated with an interactive verification procedure, V, that can be made to accept any input in S but no input outside of S. That is, there exists an interactive strategy for the prover that makes V accepts any input in S, but no strategy can make V accept an input outside of S, except with negligible probability. (See [17, Chap. 9] for a formal definition as well as a wider perspective.)

The original definition does not restrict the complexity of the strategy of the prescribed prover and the constructions of [27, 31] use prover strategies of high complexity. This fact limits the applicability of these proof systems in practice. (Nevertheless, such proof systems may be actually applied when the prover knows something that the verifier does not know, such as an NP-witness to an NP-claim, and when the proof system offers an advantage such as zero-knowledge [24, 18].)

[4]Here a "noticeable fraction" is the ratio of a linear function over an almost linear function. We stress that this is not the standard definition of this notion (at least not in cryptography).

Doubly-efficient proof systems. Seeking to make interactive proof systems available for a wider range of applications, Goldwasser, Kalai and Rothblum put forward a notion of *doubly-efficient* interactive proof systems (also called *interactive proofs for muggles* [23] and *interactive proofs for delegating computation* [30]). In these proof systems the prescribed prover strategy can be implemented in polynomial-time and the verifier's strategy can be implemented in almost-linear-time. (We stress that unlike in *argument systems*, the soundness condition holds for all possible cheating strategies, and not only for feasible ones.) Restricting the prescribed prover to run in polynomial-time implies that such systems may exist only for sets in $\mathcal{BPP}$, and thus a polynomial-time verifier can check membership in such sets by itself. However, restricting the verifier to run in almost-linear-time implies that something can be gained by interacting with a more powerful prover, even though the latter is restricted to polynomial-time.

The potential applicability of doubly-efficient interactive proof systems was demonstrated by Goldwasser, Kalai and Rothblum [23], who constructed such proof systems for any set that has log-space uniform circuits of bounded depth (e.g., log-space uniform $\mathcal{NC}$). A recent work of Reingold, Rothblum, and Rothblum [30] provided such (constant-round) proof systems for any set that can be decided in polynomial-time and a bounded amount of space (e.g., for all sets in $\mathcal{SC}$).

Towards algorithmic design of proof systems. In our prior work [20], we proposed to develop a more "algorithmic" understanding of doubly-efficient interactive proofs; that is, to identify structures and patterns that facilitate the design of efficient proof systems. Specifically, we identified a natural class of polynomial-time computations, and constructed simpler doubly-efficient proof systems for this class.[5] The aforementioned class consists of all sets that can be locally-characterized by the conjunction of polynomially many local conditions, each of which can be expressed by Boolean formulae of polylogarithmic size. The class of locally-characterizable sets is believed not to be in Dtime(p) for any fixed polynomial p, and contains natural problems of interest such as determining whether a given graph *does not* contain a clique of constant size t.

The proof system presented in [20] capitalizes on the fact that membership in a locally characterizable set can be cast as satisfying polynomially many low-degree equations. Hence, analogously to [27, 31], the first step is recasting membership in a locally-characterizable set as an algebraic problem, which consists of computing the sum of polynomially many evaluations of a low-degree polynomial (where the particular polynomial is derived from the description of the locally-characterizable set). The interactive proof uses the sum-check protocol [27] to verify the correctness of the sum, and the same applies to counting versions of locally charcatizeable sets. Hence, the intuitive appeal of the problem of counting t-cliques is lost at the first step (in which we consider an algebraic version or extension of the original problem).

1.2.2 Our results

In the current work, we present a new (doubly-efficient) interactive proof for counting t-cliques in a graph. The proof system proceeds in iterations, where in the i^{th} iteration verifying *the number of* $(t-i+1)$*-cliques in a graph* is reduced to *verifying the number of* $(t-i)$*-cliques in a related graph.* Hence, the claims made in these iterations, whose complexity gradually decreases from one iteration to the next, all have a clear intuitive meaning that is similar to the original problem (counting the

[5] Indeed, the aforementioned class (of locally-characterizable sets) is a sub-class of $\mathcal{NC} \cap \mathcal{SC}$, yet the interactive proofs presented in [20] are significantly simpler than those in [23, 30].

number of cliques in a graph).

Beyond its conceptual appeal, this proof system has the concrete advantage that the (honest) prover's complexity is directly related to the complexity of the statement being proved: the prover can be implemented in linear time given an oracle for counting t-cliques. The proof system extends to varying $t = t(n)$, yielding an alternative interactive proof system for $\#\mathcal{P}$; in particular, we note that this proof system does not use the sumcheck protocol.

Theorem 1.3 (interactive proof systems for counting cliques of given size): *For an efficiently computable function $t : \mathbb{N} \to \mathbb{N}$, let S_t denote the set of pairs (G, N) such that the graph $G = ([n], E)$ has N distinct $t(n)$-cliques. Then, S_t has a $(t-2)$-round* (public coin) *interactive proof system* (of perfect completeness) *in which the verifier's running time is $\widetilde{O}(t(n)^2 \cdot n^2)$, and the prover's running time is $\widetilde{O}(t(n)^2 \cdot n^{1+\omega_{\rm mm}\cdot\lceil (t(n)-1)/3\rceil})$, where $\omega_{\rm mm}$ is the matrix multiplication exponent. Furthermore, the prover can be implemented in* $\mathrm{poly}(t(n))\cdot\widetilde{O}(n^2)$*-time, when given access to an oracle for counting $(t(n)-1)$-cliques in* $\mathrm{poly}(t(n)) \cdot \widetilde{O}(n)$*-vertex graphs.*

High-level structure of the new interactive proof system. As discussed above, the interactive proof proceeds in iterations. The $i^{\rm th}$ iteration is a (doubly efficient and interactive) reduction from verifying *the number of $(t-i+1)$-cliques in a graph* to *verifying the number of $(t-i)$-cliques in a related graph.* This proceeds in two conceptual steps.

First, we reduce verifying the number of t'-cliques in $G' = ([n'], E')$ to verifying, for each vertex $j \in [n']$, the number of t'-cliques in G' that contain the vertex j, which equals the number of $(t'-1)$-cliques in the graph induced by the neighbors of j. Next, we let the parties reduce the latter n' claims to a single claim regarding the number of $(t'-1)$-cliques in a new graph, which has the same number of vertices as G', where the reduction is via a single-round randomized interaction. That is, while the first step employs downwards reducibility, where the decreased parameter is the size of the counted cliques, the second step employes *batch verification* (cf., [30]), where *the verification of n' claims is reduced to a verification of a single claim of the same type.*

Essentially, the foregoing batch verification is performed by considering an error correcting encoding of the n' graphs by a sequence of $\mathrm{poly}(n')$ graphs, each having n' vertices. The prover sends a succinct description of the number of $(t'-1)$-cliques in these $\mathrm{poly}(n')$ graphs, and the verifier verifies that the sum of the values associated with the n' former graphs equals the claim regarding the number of t-cliques in G'. If so, then the verifier select one of the $\mathrm{poly}(n')$ graphs at random for the next iteration (in which it verifiers the number of $(t'-1)$-cliques in the selected graph). Indeed, the crucial point is finding a suitable encoding scheme, and this is discussed in Section 1.3.1.

We stress that the foregoing procedure does not use the sum-check protocol, and that each iteration starts and ends with an intuitive combinatorial claim to be verified. (Algebra "raises its ugly head" only in the encoding scheme, which utilizes a so-called multiplication code [28], and in the fact that cliques are indicated by products of all corresponding "edge indicators".) As noted above, A concrete advantage of the current interactive proof system over the one in [20] is that the prover's complexity is proportional to the complexity of counting t-cliques (which is lower than $n^{2.373\cdot\lceil t/3\rceil}$) rather than being proportional to n^t. The foregoing procedure works also for varying $t = t(n)$, yielding an *alternative interactive proof system for $\#\mathcal{P}$.*

Application to proofs of work. Proofs of work were introduced by Dwork and Naor [14] as a method for certifying, in an easily verifiable way, that an untrusted party expended non-trivial computational resources. This may be useful, for example, in fighting denial of service attacks. A proof of work usually consists of a procedure for generating puzzles that are moderately hard to solve, and a procedure for verifying the correctness of solutions. Our results immediately yield proofs of work based on the problem of counting t-cliques. Puzzles can be generated by sampling graphs from the hard distribution of Theorem 1.1, where the solution is the number of t-cliques in the graph. Solutions can be verified using the interactive proof system of Theorem 1.3. In particular, this yields an appealing proof of *useful* work system (cf. [7]). Moreover, our results imply that the work needed to solve puzzles and to prove the correctness of solutions is closely related to the complexity of counting t-cliques in worst-case graphs.

1.3 Techniques

One common theme in the two parts of this work is that we find it beneficial to consider "weighted generalizations" of the problem of counting t-cliques. Specifically, we consider graphs with either vertex or edge weights, and define the weight of the clique as the *product* of the corresponding weights (where arithmetic is performed over a finite field). Our definition stands in contrast to the standard practice of defining the weight of a set as the sum of its elements (cf. [2]), but in the case of vertex-weights it has a very appealing interpretation (see Section 1.3.1). In any case, after working with the weighted problems, we present reductions from the weighted problems back to the original problem (of counting t-cliques). The reduction for the case of weighted edges is more complex than the one for weighted vertices.

A second common theme is the manipulation of graphs via the manipulation of their vertex (or edge) weights. Encoding a sequence of weighted graphs is performed by encoding the corresponding sequences of weights (see Section 1.3.1), and performing self-correction is possible by considering sequences that extend each of the weights (see Section 1.3.2). These comments will hopefully become more clear when we get to the specifics.

1.3.1 For the direct interactive proof systems

Here it is useful to consider vertex-weighted graphs. A n-vertex graph with vertex weights $(w_1, ..., w_n)$ such that vertex $i \in [n]$ is assigned the weight w_i may be thought of as a succinct representation of a larger graph consisting of n independent sets such that the i^{th} independent set has w_i vertices and edges represented complete bipartite graphs between the corresponding independent sets. From this perspective, it is natural to define the weight of the clique S as $\prod_{i \in S} w_i$.

Given such a weighted graph $G = ([n], E)$, the set of sum of the weights of the t-cliques that contain a specific vertex $j \in [n]$ equals w_j times the sum of the weighted $(t-1)$-cliques in the graph induced by the neighbors of j. The latter graph is obtained by resetting the weights of non-neighbors of j to 0 and keeping the weights of the neighbors of j intact. Note that the topology of the graph (i.e., its vertex and edge sets) remain intact, only the weights are updated; that is, the weights $w = (w_1, ..., w_n)$ are replaced by the weights $w^{(j)} = (w_1^{(j)}, ..., w_n^{(j)})$ such that $w_i^{(j)} = w_i$ if $\{i, j\} \in E$ and $w_i^{(j)} = 0$ otherwise. Next, we encode the resulting sequence of weighted graphs, all having the same topology G, by encoding the n weights of each vertex such that the k^{th} codeword, denoted $\overline{c}_k$, encodes the weights of $k \in [n]$ in each of the n graphs (i.e., $w_k^{(1)}, ..., w_k^{(n)}$). Specifically, using the Reed-Solomon code (which is a "good" linear "multiplication code" [28]), we obtain n codewords such that multiplying any t of them (in a coordinatewise manner) yields an (m-long) encoding of the weights of the corresponding t-subset in the n graphs.[6]

In the corresponding iteration of the interactive proof system, the prover will send a codeword, denoted $c = (c_1, ..., c_m)$, that represents the sum of the $\binom{n}{t}$ codewords that encode the weights of all t-subsets of $[n]$. The verifier will decode this (m-long) codeword, check that the sum of the resulting n values equals the value claimed in this iteration, and send the prover a random position in the codeword (i.e., a random $r \in [m]$). The next iteration will refer to the weighted graph G with weights that are determined by the r^{th} coordinate of the codewords $\overline{c}_1, ..., \overline{c}_n$ (i.e., the weights are $\overline{c}_1[r], ..., \overline{c}_n[r]$), and the claimed value will be c_r (i.e., the r^{th} coordinate of the codeword sent by the prover).

Note that we capitalize on the fact that the sum of the weights of t-cliques in a weighted graph is expressed as a sum of t-way products of vertex weights. A crucial feature of the Reed-Solomon code, which enables the foregoing manipulation, is that multiplying together t codewords of the original code that has large distance (i.e., $1 - \epsilon$) yield a codeword of a code that has sufficiently large distance (i.e., $1 - t\epsilon$). And we also use the fact that the code is linear, but *c'est tout*. In particular, unlike Meir [28], we do not use tensor codes. Furthermore, unlike most work in the area, we do not use the sum-check protocol nor refer to objects (like low-degree extensions of Boolean functions) that have no direct intuitive meaning.

The foregoing description refers to vertex weights that reside in a finite field of polynomial (in n) size. To get back to the unweighted problem of counting t-cliques, we first reduce the weighted problem over $\mathrm{GF}(p)$ to $O(t^2 \log p)$ weighted problems over smaller fields, each of size $O(t^2 \log p)$. Next, we reduce each of these problems to an unweighted problem using the reduction outlined in the first paragraph of this section (i.e., replacing the vertices by independent sets of size that equals the vertex's weight and connecting vertices that reside in different independent sets if and only if the original vertices were connected). Since the weights are currently small, this blows-up the size of the graph by a small amount.

1.3.2 For the worst-case to average-case reductions

Here it is useful to considered edge-weighted graphs. In fact, we may ignore the graph and just consider weights assigned to all edges of the complete graph, since the non-existence of an edge can be represented by a weight of zero. Hence, we consider a symmetric matrix $W = (w_{j,k})$, and allow non-zero diagonal entries as representations of vertex-weights (as in Section 1.3.1).[7] We define the weight of the set of vertices S as a product of the weights of all edges that are incident at S (i.e., $\prod_{j \le k \in S} w_{j,k}$). Similarly to Section 1.3.1, we show that the sum of the weights of all t-subsets of $[n]$ is proportional to the number of t-cliques in a (much) larger graph, but the reduction in this case is more complex than in Section 1.3.1. Before discuss this reduction, we outline the ideas used in the worst-case to average-case and rare-case reductions of the edge-weighted problems.

The worst-case to average-case reduction. We first observe that the problem of computing the sum of the weights of t-subsets of vertices in an edge-weighted graph, when defined over a finite field, is random self-reducible. Specifically, given a n-by-n matrix $W = (w_{j,k})$ such that all $w_{j,k}$'s reside in $\{0, 1, ..., b\}$, we pick a prime field of size $p = O(n^t b^{t^2})$, select uniformly a random matrix

[6] Note that if all original weights are in $\{0, 1, ..., b\}$, then we can work with a prime field of size $p = O(n^t b^t)$, since the weight of each t-subset resides in $[0, b^t]$. In subsequent iterations, the claims will refer to the value modulo p. Recall that in this case $m = p$.

[7] Alternatively, one may consider the weights of the diagonal entries as weights of the corresponding self-loops.

$R \in \mathrm{GF}(p)^{n \times n}$, and obtain the values of the sum of weighted t-subsets for the matrices $W + iR$, for $i = 1, ..., t^2$. Note that we are interested in $\mathtt{val}(W)$, where $\mathtt{val}(X)$ is the sum over all t-subset S of $\prod_{j \leq k \in S} x_{j,k}$ (reduced modulo p). Hence, $\mathtt{val}(W + \zeta R)$ is a polynomial of degree $\binom{t}{2} + t < t^2$ in ζ, and its value at 0 can be determined based on its value at $1, ..., t^2$. Furthermore, for every $i \in \mathrm{GF}(p) \setminus \{0\}$, the matrix $W + iR$ is uniformly distributed in $\mathrm{GF}(p)^{n \times n}$. Using $O(t^2)$ non-zero evaluation points (for this polynomial) and employing the Berlekamp–Welch algorithm, we obtain a worst-case to average-case reduction for computing $\mathtt{val}$ modulo p.

The foregoing presentation refers to a fixed prime $p > n^t$, whereas it is not known how to determine such a prime in time $\widetilde{O}(n^2)$. Instead, we pick primes of the desired size at random, apply the self-correction process in each of the corresponding fields, and combine the results using Chinese Remaindering with error [19]. for details, see Section 3.3.2.

The worst-case to rare-case reduction. Turning from average-case to rare-case targeted reductions, we employ a methodology heralded by Impagliazzo and Wigderson [25], and stated explicitly in our prior work [21]. The methodology is pivoted at the notion of sample-aided reductions, which is extended in the current work. In the following definition (which is taken from [21]), a task consists of a computational problem along with a required performance guarantee (e.g., "solving problem Π on the worst-case" or "solving Π with success rate ρ under the distribution $\mathcal{D}$"). For sake of simplicity, we consider the case that the first task is a worst-case task.

Definition 1.4 (sample-aided reductions): *Let $\ell, s : \mathbb{N} \to \mathbb{N}$, and suppose that M is an oracle machine that, on input $x \in \{0,1\}^n$, obtains as an auxiliary input a sequence of $s = s(n)$ pairs of the form $(r, v) \in \{0,1\}^{n+\ell(n)}$. We say that M is an* sample-aided reduction of solving Π in the worst-case to the task $\mathtt{T}$ *if, for every procedure P that performs the task $\mathtt{T}$, it holds that*

$$\Pr_{r_1,...,r_s \in \{0,1\}^n} \left[\Pr[\forall x \in \{0,1\}^n \; M^P(x; (r_1, \Pi(r_1)), ..., (r_s, \Pi(r_s))) = \Pi(x)] \geq 2/3 \right] \; > \; 2/3, \qquad (1)$$

where the internal probability is taken over the coin tosses of the machine M and the procedure P.

Note that a sample-aided reduction implies an ordinary non-uniform reduction. Furthermore, coupled with a suitable downwards self-reduction for Π, a sample-aided reduction of solving Π in the worst-case to solving Π on the average (resp., in the rare-case) implies a corresponding standard reduction (of worst-case to average-case (resp., to rare-case)).

In this work we extend Definition 1.4 by allowing the sample to be drawn from an arbitrary distribution over $(\{0,1\}^n)^{s(n)}$; in particular, the $s(n)$ individual samples need not be independently and identically distributed. We note that both the foregoing implications hold also under this extension, where for the second implication we also require that the sample distribution be efficiently sampleable.

Our worst-case to rare-case reduction utilizes the worst-case to rare-case reduction of Sudan, Trevisan, and Vadhan [33], which applies to low-degree polynomials. We first observe that this reduction yields a sample-aided reduction, and then apply it to the edge-weighted clique-counting problem, while presenting a downwards self-reduction for the latter problem (where this reduction is analogous to the one used in Section 1.3.1). We stress that our sample-aided reduction utilizes correlated random samples (and the extension of Definition 1.4 is used here); specifically, in our application, the samples correspond to pairs of objects and we need multiple samples that coincide on the first element of the pair, for multiple choices of such first element.

Back to the unweighted problem. Having shown the (average-case and) rare-case hardness of the edge-weighted clique-counting problem, we seek to establish this result for the original counting problem (which refers to simple unweighted graphs). An adequate reduction is presented in Section 3.2; it is more complex than the reduction employed to the vertex-weighted problem, and it involves increasing the number of vertices in the graph by a factor of $O(\log n)^{\widetilde{O}(t^2)}$. (In contrast, the reduction employed to the vertex-weighted problem increases the number of vertices by a fcator of $O(t^3 \log n)$.)

Given that our reductions increase the number of vertices in the graph, and seeking to maintain that number, we present a reduction of counting t-cliques in $\widetilde{O}(n)$-vertex graphs to counting t-cliques in n-vertex graphs (see Section 3.3.3). Lastly, given that our worst-case to rare-case reduction reduces to several instance lengths, we also present a reduction from the rare-case problem of counting t-cliques under several distributions to counting them under one distribution (see Section 3.4.3).

1.4 Other related work

Several works have constructed interactive (and non-interactive) proof systems for clique-counting, where the interactive proof system of Thaler [32, Apdx] is most related to our work.[8] Specialized to the problem counting t-cliques, this interactive proof system uses $t-2$ rounds, with $\widetilde{O}(n)$ communication, $O(|E|+n)$ verification time, and $O(|E| \cdot n^{t-2})$ proving time. However, his proof system uses the sum-check protocol as well as the arithmetization approach of [27], which is also followed in [20]. In contrast, our proof system (of Theorem 1.3) has the salient feature of deviating from the arithmetization approach of [27], and maintaining the combinatorial flavor of the original problem throughout interaction. Furthermore, the complexity of our prover strategy is directly related to the complexity of counting the number of t-cliques in a graph. In particular, for fixed $t \geq 3$, known algorithms for this problem give a prover runtime of $\widetilde{O}(n^{1+\omega_{\rm mm} \cdot \lceil (t-1)/3 \rceil}) \ll |E| \cdot n^{t-2}$, where $\omega_{\rm mm}$ is the matrix multiplication exponent.

While it is unknown whether non-deterministic algorithms (equiv., non-interactive proof systems) can outperform the best algorithm known for counting t-cliques, Williams [35] showed that such an improvement is possible when allowing randomization; that is, non-interactive and randomized proof systems (as captured by the complexity class $\mathcal{MA}$) can outperform algorithms. Specifically, his (single-message) proof system for counting the number of t-cliques has proof length and verification time $\widetilde{O}(n^{\lfloor t/2 \rfloor + 2}) \ll n^{2/3}$. Subsequent improvement by Björklund and Kaski [8] yields an $\mathcal{MA}$ proof system with length and verification time $\widetilde{O}(n^{(\omega_{\rm mm}+\epsilon) \cdot t/6})$, where $\epsilon > 0$ is an arbitrarily small constant. The time to construct proofs in their system is $\widetilde{O}(n^{(\omega_{\rm mm}+\epsilon)t/3})$, matching the best algorithm known for solving the problem. Recall, however, that we construct *interactive* proof systems: using interaction lets us reduce the verification time (as well as the total communication) to $\widetilde{O}(n^2)$.

1.5 Notation and organization

For a natural number n, we let $[n] = \{1, ..., n\}$ and $[[n]] = \{0\} \cup [n]$. For a set U, we let $\binom{U}{t} = \{S \subseteq U : |S| = t\}$. For a prime p, we let $\mathrm{GF}(p)$ denote the finite field of cardinality p.

[8] We remark that the protocol of [32] operates in a more challenging streaming setting, which we do not consider or elaborate on in this work.

We often conduct our discussion with reference to a seemingly fixed number of vertices (denoted n), clique size (denoted t), and prime p; the reader should think of n as varying (or generic), and of t and p as possibly depending on n.

Organization. In Section 2 we present direct interactive proof systems for counting t-cliques in graphs, using the notion of vertex-weighted t-cliques as a methodological vehicle. In Section 3 we present worst-case to average-case (and to rare-case) reductions for counting t-cliques in graphs, using the notion of edge-weighted t-cliques as a methodological vehicle. The worst-case to average-case reduction for counting t-cycles (mentioned right after Theorem 1.1) is presented in Section 3.5.

2 Vertex-weighted t-Clique

We consider a generalization of the t-clique counting problem, in which *one is given a graph $G = ([n], E)$ along with vertex-weights, and is required to output the sum of the weights of all t-cliques in G*, where the weight of a set $S \subseteq [n]$ is defined as the *product* of the weights of its elements.[9] That is, for a (simple) graph $G = ([n], E)$ and a sequence of vertex weights, $w = (w_1, ..., w_n) \in (\mathbb{N} \cup \{0\})^n$, we let $\mathtt{CWC}_t^G(w)$ (standing for *count weighted cliques*) denote the sum of the weights of all t-cliques in G; that is,

$$\mathtt{CWC}_t^G(w) \stackrel{\text{def}}{=} \sum_{S \in \binom{[n]}{t} : \mathtt{CL}(G_S)} \prod_{j \in S} w_j \tag{2}$$

where G_S is the subgraph of G induced by S and $\mathtt{CL}(G')$ holds if G' is a clique. Indeed, $\mathtt{CWC}_t^G(1^n)$ equals the number of t-cliques in G. In general, as shown in Section 2.2, $\mathtt{CWC}_t^G(w)$ equals the number of t-cliques in a graph G' that is obtained by a blow-up of G in which the i^{th} vertex is replaced by an independent set of size w_i (and the edge $\{i, j\}$ is replaced by a complete bipartite graph between the i^{th} and j^{th} sets).

2.1 A direct interactive proof for counting weighted cliques

Towards presenting an interactive proof system for the value of $\mathtt{CWC}_t^G(1^n)$, we first observe that (for every $w \in (\mathbb{N} \cup \{0\})^n$) it holds that

$$t \cdot \mathtt{CWC}_t^G(w) = \sum_{i \in [n]} w_i \cdot \mathtt{CWC}_{t-1}^G(w^{(i)}) \tag{3}$$

$$\text{where } w_j^{(i)} = w_j \text{ if } \{i, j\} \in E, \text{ and } w_j^{(i)} = 0 \text{ otherwise.} \tag{4}$$

(This is the case since each pair (S, i) such that S is a t-subset of $[n]$ and $i \in S$ contributes equally to each side of Eq. (3), where the contribution equals $w_i \cdot \prod_{j \in S \setminus \{i\}} w_j$ if G_S is a clique and equals 0 otherwise. In particular, note that if all vertices in $S \setminus \{i\}$ are neighbors of i, then $\prod_{j \in S \setminus \{i\}} w_j^{(i)} = \prod_{j \in S \setminus \{i\}} w_j$ and otherwise $\prod_{j \in S \setminus \{i\}} w_j^{(i)} = 0$.)

Indeed, Eq. (3) reduced the evaluation of $\mathtt{CWC}_t^G$ to n evaluations of $\mathtt{CWC}_{t-1}^G$; equivalently, it reduces the verification of the value of $\mathtt{CWC}_t^G$ at one point to the verification of the value of $\mathtt{CWC}_{t-1}^G$

[9] Our definition stands in contrast to the standard practice of defining the weight of a set as the sum of its elements (cf. [2]).

at n points. Wishing to reduce these n value-verifications to a single one, we seek an error correcting code by which these n values can be encoded by a sequence of $\Omega(n)$ values such that verifying one of the values in the sequence suffice. As usual, the code of choice is the Reed-Solomon code (i.e., univariate polynomials of degree $n-1$), which calls for embedding the values in a finite field. Recalling that we are actually interested in the value of $\mathtt{CWC}_t^G(1^n) \leq t! \cdot \binom{n}{t} < n^t$, we embed all values in a finite field $\mathcal{F} = \mathrm{GF}(p)$, for a prime $p > n^t$, and reduce all w_j's modulo p. Now, we consider the following n polynomials $(f_j : \mathcal{F} \to \mathcal{F})_{j\in[n]}$:

$$f_j(z) \stackrel{\text{def}}{=} \sum_{i\in[n]} \mathtt{EQ}_i(z) \cdot w_j^{(i)} \tag{5}$$

where $\mathtt{EQ}_i : \mathcal{F} \to \mathcal{F}$ is a (degree $n-1$) polynomial such that $\mathtt{EQ}_i(i) = 1$ and $\mathtt{EQ}_i(k) = 0$ for every $k \in [n] \setminus \{i\}$ (i.e., $\mathtt{EQ}_i(z) = \prod_{j\in[n]\setminus\{i\}}(z-j)/(i-j)$). Note that $f_j(i) = w_j^{(i)}$ for every $i \in [n]$, and that each f_j can be computed in $O(n)$ field operations when given w (by using Eq. (4) and $\mathtt{EQ}_i(z) = \prod_{j\in[n]\setminus\{i\}}(z-j)/(i-j)$).[10] Letting $\mathtt{CWC}_t^{G,p}(w) \stackrel{\text{def}}{=} \mathtt{CWC}_t^G(w) \bmod p$, this leads to the following interactive reduction of the verification of the value of $\mathtt{CWC}_t^{G,p}$ at one point to the verification of the value of $\mathtt{CWC}_{t-1}^{G,p}$ at one (other) point.

Construction 2.1 (a generic iteration of the interactive proof system for counting cliques): *The iteration starts with a claim of the form $\mathtt{CWC}_t^{G,p}(w) = v$, where $w \in \mathcal{F}^n$ and $v \in \mathcal{F}$ are determined before* (by the previous iteration or by the main protocol).[11]

1. *The prover computes the $(t-1)\cdot(n-1)$ degree polynomial $P_{t-1} : \mathcal{F} \to \mathcal{F}$, where*

$$P_{t-1}(z) \stackrel{\text{def}}{=} \sum_{S\in\binom{[n]}{t-1}:\mathtt{CL}(G_S)} \prod_{j\in S} f_j(z), \tag{6}$$

   where the arithmetic is over $\mathcal{F} = \mathrm{GF}(p)$, and sends P_{t-1} to the verifier.

   Note that $P_{t-1}(z)$ can be computed by interpolation, using the values of P_{t-1} at less than tn points, and that (as shown below) *for every $k \in \mathcal{F}$ it holds that $P_{t-1}(k) = \mathtt{CWC}_{t-1}^{G,p}(w^{(k)})$, where $w_j^{(k)} = f_j(k)$ for every $j \in [n]$.*

2. *Upon receiving a polynomial $\widetilde{P}$ of degree $(t-1)\cdot(n-1)$, the verifier checks whether $t \cdot v \equiv \sum_{i\in[n]} w_i \cdot \widetilde{P}(i) \pmod p$, and rejects if equality does* not *hold. If equality holds, the verifier selects uniformly $r \in \mathcal{F}$, and sends it to the prover.*

The iteration ends with the claim that $\mathtt{CWC}_{t-1}^{G,p}(w') = v'$, where $v' = \widetilde{P}(r)$ and $w'_j = f_j(r)$ for every $j \in [n]$.

[10] Note that, for any $k \in [n]$ it holds that $(\mathtt{EQ}_1(k), ..., \mathtt{EQ}_n(k)) = 0^{k-1}10^{n-k}$, whereas for $k \in \mathcal{F} \setminus [n] \setminus [n]$ it holds that $\mathtt{EQ}_i(k) = \frac{\prod_{j\in[n]}(k-j)}{(k-i)\cdot\prod_{j\in[n]\setminus\{i\}}(i-j)}$. Hence, computing $(\mathtt{EQ}_1(k), ..., \mathtt{EQ}_n(k))$ reduces to computing $\prod_{j\in[n]}(k-j)$ and computing all $\prod_{k\in[m]} k$ for $m = 1, ..., n-1$, which in turn means that $(\mathtt{EQ}_1(k), ..., \mathtt{EQ}_n(k))$ can be computed in $O(n)$ field operations.

[11] Indeed, as hinted in the title of this construction, it will be applied iteratively by the main interactive proof systems, which contains little beyond these iterations (see the proof of Theorem 2.3).

Recall that both parties can compute each of the f_j's using $O(n)$ field operations, which implies that the verifier strategy can be implemented in $\widetilde{O}(t \cdot n^2)$-time (assuming the field has size $O(n^t)$).[12] In addition, the prover needs to construct the polynomial P_{t-1}, and a straightforward way of doing so can be implemented using $tn \cdot (O(n^2) + \binom{n}{t-1}) = O(n^t)$ field operations. We shall later see that the complexity can be improved to $O(t \cdot n^{1+\omega_{\mathtt{mm}}\lceil (t-1)/3 \rceil})$, where $\omega_{\mathtt{mm}}$ is the matrix multiplication exponent, by using the ideas that underlie the best algorithm known for deciding t-Clique (which also suffices for counting t-cliques). But before getting to this improvement, we analyze the effect of Construction 2.1.

Proposition 2.2 (analysis of Construction 2.1): *Let w, v, w' and v' be as in Construction 2.1.*

1. *If $\mathtt{CWC}_t^{G,p}(w) = v$ and both parties follow their instructions, then $\mathtt{CWC}_{t-1}^{G,p}(w') = v'$ holds.*
2. *If $\mathtt{CWC}_t^{G,p}(w) \neq v$ and the verifier follows its instructions, then either the verifier rejects or $\mathtt{CWC}_{t-1}^{G,p}(w') = v'$ holds with probability at most $tn/|\mathcal{F}|$.*

Proof: Using Eq. (6), Eq. (5), and Eq. (2), we get for every $i \in [n]$

$$\begin{aligned} P_{t-1}(i) &= \sum_{S \in \binom{[n]}{t-1}:\mathtt{CL}(G_S)} \prod_{j \in S} f_j(i) \\ &= \sum_{S \in \binom{[n]}{t-1}:\mathtt{CL}(G_S)} \prod_{j \in S} w_j^{(i)} \\ &= \mathtt{CWC}_{t-1}^{G,p}(w^{(i)}), \end{aligned}$$

and so $t \cdot \mathtt{CWC}_t^G(w) \equiv \sum_{i \in [n]} w_i \cdot P_{t-1}(i) \pmod p$. More generally (by the same argument), for every $k \in \mathcal{F}$, it holds that $P_{t-1}(k) = \mathtt{CWC}_{t-1}^{G,p}(w^{(k)})$, where $w_j^{(k)} = f_j(k)$ for every $j \in [n]$.

Under the hypothesis of Item 1, $v = \mathtt{CWC}_t^{G,p}(w)$ and the polynomial $\widetilde{P}$ received by the verifier equals P_{t-1}, which implies that the verifier does not reject (since $t \cdot v = \sum_{i \in [n]} w_i \cdot \widetilde{P}(i)$ over $\mathcal{F} = \mathrm{GF}(p)$). Furthermore, in this case $v' = \widetilde{P}(r)$ equals $P_{t-1}(r) = \mathtt{CWC}_{t-1}^{G,p}(w')$, where $w'_j = f_j(r)$ for every $j \in [n]$.

Under the hypothesis of Item 2, the polynomial P_{t-1} does *not* satisfy the equation $t \cdot v = \sum_{i \in [n]} w_i \cdot P_{t-1}(i)$ over $\mathcal{F}$. If the prover sets $\widetilde{P} = P_{t-1}$, then the verifier rejects, and otherwise $\widetilde{P}$ and P_{t-1} may agree on at most $(t-1) \cdot (n-1)$ points. In the latter case, unless r is one of the agreement points, it follows that $\mathtt{CWC}_{t-1}^{G,p}(w') = P_{t-1}(r) \neq \widetilde{P}(r)$, where $w'_j = f_j(r)$ for every $j \in [n]$. ■

The interactive proof system. Iteratively invoking Construction 2.1 yields an interactive proof system for the claim $\mathtt{CWC}_t^{G,p}(w) = v$, where $w \in \mathcal{F}^n$ and $v \in \mathcal{F}$. In the i^{th} iteration we start with a claim of the form $\mathtt{CWC}_{t-i+1}^{G,p}(w^{(i-1)}) = v^{(i-1)}$, and end with a claim of the form $\mathtt{CWC}_{t-i}^{G,p}(w^{(i)}) = v^{(i)}$. Hence, after $t-2$ iterations, we reach a claim of the form $\mathtt{CWC}_2^{G,p}(w^{(t-2)}) = v^{(t-2)}$, which the verifier can verify by itself. To apply this procedure to the problem of counting t-cliques in an n-vertex graph, we initiate it by selecting a field of prime cardinality greater than n^t, and setting $w^{(0)} = (1, ..., 1) \equiv 1^n$. Hence, we get –

[12] Note that $\widetilde{P}$ can be evaluated using $O(tn)$ field operations.

Theorem 2.3 (interactive proof systems for counting cliques of given size): *For an efficiently computable function $t : \mathbb{N} \to \mathbb{N}$, let S_t denote the set of pairs (G, N) such that the graph $G = ([n], E)$ has N distinct $t(n)$-cliques. Then, S_t has a $(t - 2)$-round* (public coin) *interactive proof system* (of perfect completeness) *in which the verifier's running time is $\widetilde{O}(t(n)^2 \cdot n^2)$, and the prover's running time is dominated by $O(t(n)^2 \cdot n)$ calls to the oracle $\overline{\texttt{CWC}}^{G,p}_{t(n)-1}$, where p is a prime number in $[n^{t(n)}, 2n^{t(n)}]$, and $\overline{\texttt{CWC}}^{G,p}_t$ answers the query $(i, w) \in [t] \times \mathrm{GF}(p)^n$ with $\texttt{CWC}^{G,p}_i(w)$.*

Indeed, this suggests an alternative construction of interactive proof systems for sets in co$\mathcal{NP}$. The fact that this construction is different from the celebrated construction of [27] is manifested by the (reduced) running time of the prover. Specifically, as shown in Section 2.2, the computation of $\texttt{CWC}^{G,p}_t$ is $O(t^2|w|)$-time reducible to counting the number of t-cliques in $\widetilde{O}(t^2 n)$-vertex graphs, which in turn has complexity $O(n^{\omega_{\texttt{mm}}\lceil t/3 \rceil})$, where $\omega_{\texttt{mm}}$ is the matrix multiplication exponent (see [29]).

Proof: On input $G = ([n], E)$ and N (which is supposed to equal the number of $t(n)$-cliques in G), the parties pick a finite field $\mathcal{F}$ of prime candinality greater than $n^{t(n)}$, and initialize $w = 1^n$ and $v = N$. (This prime can be selected at random in $[n^{t(n)}, 2n^{t(n)}]$ by either parties.) Next, they iteratively invoke Construction 2.1 for $t(n) - 2$ times (with decreasing clique-size parameter (starting at size $t(n)$ and ending at size 3)). In case the verifier did not reject (in any iteration), it is left with a claim of the form $\texttt{CWC}^G_2(w) = v$, which it can verify by itself in $O(n^2)$ time (since $\texttt{CWC}^G_2(w)$ equals $\sum_{\{j,k\} \in E} w_j \cdot w_k$). For sake of clarity, we spell out the protocol.

1. On input $G = ([n], E)$ and $N \leq \binom{n}{t(n)}$, the verifier selects uniformly a prime number p in $[n^{t(n)}, 2n^{t(n)}]$, and sends it to the prover. Both parties set $\mathcal{F} = \mathrm{GF}(p)$, and initiate $w^{(0)} \leftarrow 1^n$ and $v^{(0)} \leftarrow N$.

2. For $i = 1, ..., t(n) - 2$, the parties invoke Construction 2.1, while setting $t \leftarrow t(n) - i + 1$, $w \leftarrow w^{(i-1)}$ and $v \leftarrow v^{(i-1)}$. Once Construction 2.1 terminates, they set $w^{(i)} \leftarrow w'$ and $v^{(i)} \leftarrow v'$. (In all invocations, $\mathcal{F}$ is as set in Step 1.)

3. The verifier checks whether $\texttt{CWC}^G_2(w^{(t(n)-2)}) = v^{(t(n)-2)}$ by direct computation.

All claims of the theorem follow quite immediately. In particular, in the i^{th} iteration, the verifier performs $O((t(n) - i) \cdot n^2)$ field operations, whereas the prover's computation is dominated by less than $t(n) \cdot n$ invocations of the oracle $\texttt{CWC}^{G,p}_{t(n)-i}$. ■

Remark 2.4 (implementing $\texttt{CWC}_{t'-1}$ using $\texttt{CWC}_{t'}$): *We comment that, for any $G = ([n], E)$, the oracle $\texttt{CWC}^G_{t'-1}$* (resp., $\texttt{CWC}^{G,p}_{t'-1}$) *can be implemented using two oracle calls to $\texttt{CWC}^{G'}_{t'}$* (resp., $\texttt{CWC}^{G',p}_{t'}$), *where $G' = ([n + 1], E')$ such that $E' = E \cup \{\{j, n + 1\} : j \in [n]\}$. Specifically, $\texttt{CWC}^G_{t'-1}(w) = \texttt{CWC}^{G'}_{t'}(w1) - \texttt{CWC}^{G'}_{t'}(w0)$.*

Hence, $\overline{\texttt{CWC}}^G_t$ is $O(2^t n)$-time reducible to $\texttt{CWC}^{G'}_t$, where G' has t more vertices than G. Specifically, $\texttt{CWC}^G_{t-i}(w)$ is a linear combination (with coefficients in $\{\pm 1\}$) of the values $(\texttt{CWC}^{G'}_t(w\alpha 0^{t-i}))_{\alpha \in \{0,1\}^i}$. Observing that $f(\beta) \stackrel{\text{def}}{=} \texttt{CWC}^G_t(w\beta)$ depends only on the Hamming weight of $\beta \in \{0, 1\}^t$, we can write $\texttt{CWC}^G_{t-i}(w)$ as a linear combination of the values $(\texttt{CWC}^G_t(w0^j 1^{i-j} 0^{t-i}))_{j \in [[i]]}$. Hence, we obtain an $O(tn)$-time reduction of $\overline{\texttt{CWC}}^G_t$ to $\texttt{CWC}^{G'}_t$. A alternative ($O(tn)$-time) reduction is presented next.

Remark 2.5 (implementing $\texttt{CWC}_{t-i}$ using $\texttt{CWC}_t$): *For any $G = ([n], E)$, the oracle $\texttt{CWC}_{t-i}^{G}$* (resp., $\texttt{CWC}_{t-i}^{G,p}$) *can be implemented using one oracle call to $\texttt{CWC}_t^{G^{(i)}}$* (resp., $\texttt{CWC}_t^{G^{(i)},p}$), *where $G^{(i)}$ consists of $t-i$ independent sets of size n that are connected by a double-cover of G, and augmented by an i-clique that is connected to all these $(t-i)\cdot n$ vertices.*[13] *That is, $G^{(i)} = ([(t-i)n+i], E' \cup A)$ such that $E' = \{\{(a-1)n+j, (b-1)n+k\} : \{j,k\} \in E \,\&\, a \neq b \in [t-i]\}$ and $A = \{\{j, (t-i)n+k\} : j \in [(t-i)n] \,\&\, k \in [i]\}$. Specifically, $(t-i)! \cdot \texttt{CWC}_{t-i}^{G}(w) = \texttt{CWC}_t^{G^{(i)}}(w^{t-i}1^i)$. Seeking to use the same number of vertices in all graphs $G^{(i)}$, we may augment $G^{(i)}$ with $i \cdot n - i$ isolated vertices.*

Discussion: Our proof system for $\#\mathcal{P}$ versus the standard one

The standard interactive proof system for $\#\texttt{SAT}$ starts with a full arithmetization of the Boolean problem [27], which yields an arithmetic problem that has no clear intuitive meaning. This is done by writing the CNF formula as an arithmetic formula over GF(2), and then considering it as an Arithmetic formula over a larger field. Alternatively, given a Boolean formula over n variables, viewed as a function from $\{0,1\}^n$ to $\{0,1\}$, we write the low-degree extension of this function.

In contrast, we present an interactive proof system for counting the number of cliques in a graph that proceeds in iteration such that *in each iteration the clique counting problem is reduced to itself.* Hence, the claim made at the beginning (and the end) of each iteration has a clear intuitive meaning. (As noted upfront and elaborated in Section 2.2, the sum of weighted t-cliques in a graph G equals the number of t-cliques in a corresponding graph obtained by a suitable blow-up of G.) Furthermore, each iteration consists of a natural downwards reduction, which decreases the size parameter by one unit, and a batch verification that reduces n claims to a single one. Both reductions have an intuitive appeal, although the batch verification relies on a suitable error correcting code, which is a "multiplication code" in a sense akin to the sense used in Meir's work [28]. To simplify the exposition, we use the Reed-Solomon code, but any code supporting a sufficient number of multiplications will do.

The difference between our construction and the standard one may be articulated using Meir's observation [28] that the standard construction uses the fact that Reed-Muller codes are tensor codes (and that one can use arbitrary tensor (of multiplication) codes). Specifically, the standard construction uses a codeword that encodes the 2^n values of the Boolean formula over all assignments to the n variables. In contrast, we do not use tensor codes, and in each iteration we encode n values that correspond to n instances of the problem. (Indeed, like in the standard interactive proof systems for co$\mathcal{NP}$, we actually refer to the counting problem.)

2.2 Reducing counting vertex-weighted cliques to the unweighted case

Recall that the prover strategy underlying the interactive proof system presented in the proof of Theorem 2.3 can be implemented in $O(t(n)^2 n^2)$ time when given oracle access to $\overline{\texttt{CWC}}_t^{G,p}$, where p is a prime in $[n^{t(n)}, 2n^{t(n)}]$. In fact, the prover uses $O(t(n)\cdot n)$ calls to each of oracles $\texttt{CWC}_i^{G,p}$ for $i = 2, ..., t(n)-1$. We first observe that each such oracle can be implemented by using $O(t(n)^2 \log n)$ oracle calls to a corresponding oracle that is defined for weights that reside in $[O(t(n)^2 \log n)]$. Next, we show that each of these queries can be implemented by counting cliques is a simple $\widetilde{O}(t(n)^2 n)$-vertex graph (with no weights). These two reductions are summarized in the following result.

[13] The double-cover of $G = ([n], E)$ is a bipartite graph B with n vertices on each side such that $\langle 1, u\rangle$ and $\langle 2, v\rangle$ are connected in B if and only if $\{u, v\} \in E$.

Proposition 2.6 (reducing $\mathtt{CWC}_t^G$ to counting t-cliques in graphs): *For a graph $G = ([n], E)$ and a prime $p \in [n^t, 2n^t]$, let $\mathtt{CWC}_t^{G,p}(w) \stackrel{\rm def}{=} \mathtt{CWC}_t^G(w) \bmod p$, where $\mathtt{CWC}_t^G$ be as in Eq. (2). Then, the computation of $\mathtt{CWC}_t^{G,p}(w)$ can be reduced in $\widetilde{O}(t^2|w|)$-time to computing the number of t-cliques in an unweighted graph having $\widetilde{O}(t^2 n)$ vertices. Furthermore, the reduction uses $O(t^2 \log n)$ queries.*

Proof: We may assume, without loss of generality, that $w \in [p]^n$, since otherwise we reduce each coordinate of w modulo p. Observe that the value of $\mathtt{CWC}_t^G(w)$ is a non-negative integer that is smaller than $t!\binom{n}{t} \cdot p^t < (np)^t < n^{2t^2}$. For $m = O(t^2 \log n)$, it holds that the product of all primes in $[m]$ is larger than n^{2t^2}, and so we may compute $\mathtt{CWC}_t^G(w)$ by computing $\mathtt{CWC}_t^G(w)$ modulo each of these primes. Hence, for each prime p_i in $[m]$, we first reduce the weights modulo p_i, obtaining $w_j^{(i)} = w_j \bmod p_i$ for each $j \in [n]$, and then compute $\mathtt{CWC}_t^{G,p_i}(w^{(i)})$, which equals $\mathtt{CWC}_t^{G,p_i}(w)$. That is, $\mathtt{CWC}_t^G(w)$ is computed using the Chinese Remainder Theorem based on the values of $\mathtt{CWC}_t^{G,p_i}(w)$ for each prime $p_i \in [m]$.

Next, we efficiently reduce the computation of $\mathtt{CWC}_t^{G,p'}(w')$, where p' is a prime in $[m]$ and $w' \in [p']^n$, to the standard problem of counting t-cliques. Actually, we reduce the computation of $\mathtt{CWC}_t^G(w')$ to counting the number of t-cliques in a related graph G'. The reduction just maps the graph $G = ([n], E)$ with weights $w' = (w'_1, ..., w'_n) \in [m]^n$ to the graph $G' = (V', E')$ in which each vertex j of G is replaced by an independent set of size w'_j, and edges are replaced by complete bipartite graphs between the corresponding independent sets; that is

$$V' = \cup_{j \in [n]} V_j \quad \text{where the } V_j\text{'s are disjoint and } |V_j| = w'_j \tag{7}$$

$$E' = \cup_{\{j,k\} \in E} \{\{u, v\} : u \in V_j \;\&\; v \in V_k\}. \tag{8}$$

Indeed, the value of $\mathtt{CWC}_t^G(w')$ equals the number of t-cliques in G'.

Combining the two reductions, we obtain a $\widetilde{O}(t^2|w|)$-time reduction of computing $\mathtt{CWC}_t^{G,p}(w)$, where $p = O(n^t)$, to counting t-cliques in $O(t^2 n \log n)$-vertex graphs. Indeed, the reduction makes $O(t^2 \log n)$ calls, where in the $i^{\rm th}$ call we obtain the number of t-cliques in the graph $G^{(i)}$ in which vertex j of G is replaced by an independent set of size $w_j^{(i)} \in [p_i]$ such that $w_j^{(i)} \equiv w_j \pmod{p_i}$. ■

Proof of Theorem 1.3: Combining Theorem 2.3 with Proposition 2.6 yields Theorem 1.3. Specifically, the proof system asserted in Theorem 2.3 is almost as asserted in Theorem 1.3, except that the prover strategy in the former system relies on $O(t(n)^2 \cdot n)$ calls to the oracle $\overline{\mathtt{CWC}}_{t(n)-1}^{G,p}$, where p is a prime number in $[n^{t(n)}, 2n^{t(n)}]$, and $\overline{\mathtt{CWC}}_t^{G,p}$ answers the query $(i, w) \in [t] \times \mathrm{GF}(p)^n$ with $\mathtt{CWC}_i^{G,p}(w)$. Proposition 2.6 implies that each of these $O(t(n)^2 \cdot n)$ queries can be answered by making $O(t(n)^2 \log n)$ queries to an oracle that on input a $\widetilde{O}(t(n)^2 \cdot n)$-vertex graph G' and an integer $i \in [t(n) - 1]$ returns the number of i-cliques in G'. Lastly, using Remark 2.4 (and the discussion following it), we can implement each of these queries by $t(n)$ queries to an oracle that returns the number of $(t(n) - 1)$-cliques in $\widetilde{O}(t(n)^2 \cdot n)$-vertex graphs. Alternatively, using Remark 2.5, we can implement each of the i-clique queries by a single query to an oracle that returns the number of $(t(n) - 1)$-cliques in $\widetilde{O}(t(n)^3 \cdot n)$-vertex graphs. ■

3 Edge-weighted t-Clique

Here we consider edge weights in addition to the vertex weights that were considered in the previous section. For simplicity of exposition, we consider vertex weights as if they are the weights of the corresponding self-loops. Given that we use edge weight, there is no need to specify a graph since non-edges can be represented as edges with weight zero. Indeed, the weight of a t-subset of vertices will be defined as equal the product of the weights of all edges in the induced graph (including the self-loops). Hence, for a symmetric (and possibly reflexive) n-by-n matrix $W = (w_{j,k})_{j,k\in[n]}$, we let $\mathtt{CWC}_t(W)$ denote the sum of the weights of all t-subsets of vertices; that is,

$$\mathtt{CWC}_t(W) \stackrel{\rm def}{=} \sum_{S\in\binom{[n]}{t}} \prod_{j\le k:j,k\in S} w_{j,k} \tag{9}$$

Indeed, for a simple graph $G = ([n], E)$ and $w \in (\mathbb{N}\cup\{0\})^n$, the quantity $\mathtt{CWC}_t^G(w)$ equals $\mathtt{CWC}_t(W)$, where $w_{j,j} = w_j$ for every $j \in [n]$, and $w_{j,k} = 1$ if $\{j, k\} \in E$ and $w_{j,k} = 0$ otherwise (i.e., if $\{j, k\} \in \binom{[n]}{2}\setminus E$). As in Section 2, we shall show how to reduce the computation of $\mathtt{CWC}_t$ to counting t-cliques in graphs, alas the current reduction (presented in Section 3.2) is less straightforward and incurs a larger overhead.

As in Section 2, we shall also consider the restriction and reduction of $\mathtt{CWC}_t$ to prime fields; that is, abusing notation, for a prime p, we let $\mathtt{CWC}_t^p(W) \stackrel{\rm def}{=} \mathtt{CWC}_t(W) \bmod p$. (We believe that in doing so we risk no confusion, because in the current section a graph will never be used as a superscript to $\mathtt{CWC}_t$.)

Our motivation for the current generalization is that it supports a worst-case to average-case reduction, which will be used to prove Theorem 1.1. But before describing this reduction, we show that the results of the previous section extend to the current generalization. First, note that

$$t\cdot\mathtt{CWC}_t(W) = \sum_{i\in[n]} w_{i,i}\cdot\mathtt{CWC}_{t-1}(W^{(i)}),\ \text{where } W^{(i)} = (w^{(i)}_{j,k}) \text{ satisfies} \tag{10}$$

$$w^{(i)}_{j,k} = \begin{cases} 0 & \text{if } j = k = i \\ w_{i,j}\cdot w_{j,j} & \text{if } j = k \in [n]\setminus\{i\} \\ w_{j,k} & \text{otherwise (i.e., } j \neq k) \end{cases} \tag{11}$$

(This is the case since each pair (S, i) such that S is a t-subset of $[n]$ and $i \in S$ contributes equally to each side of Eq. (10), where the contribution is $\prod_{j\in S} w_{i,j}\cdot\prod_{j\le k\in S\setminus\{i\}} w_{j,k}$, which equals $w_{i,i}\cdot\prod_{j\le k\in S\setminus\{i\}} w^{(i)}_{j,k}$.) Note that $W^{(i)}$ and W differ only on the self-loops (i.e., $w^{(i)}_{j,k} = w_{j,k}$ for every $j \neq k$), where $w^{(i)}_{i,i} = 0$ and $w^{(i)}_{j,j} = w_{i,j}\cdot w_{j,j}$ for every $j \in [n]\setminus\{i\}$.

Organization. In Section 3.1 we present an interactive proof system for the problem of "counting" edge-weighted t-cliques (i.e., the set of pairs $(W, \mathtt{CWC}_t(W))$). This result is presented for the sake of elegancy, and is not used anywhere else in this work; it is actually subsumed by combining the results of Sections 2.1 and 3.2. Sections 3.2–3.4 are the core of the current section (i.e., Section 3). In Section 3.2 we show that computing $\mathtt{CWC}_t$ (i.e., "counting" edge-weighted t-cliques) can be reduced to computing the number of t-cliques in unweighted graphs. In Section 3.3 (resp., Section 3.4) we present a worst-case to average-case (resp., rare-case) reduction for the edge-weighted clique counting problem, establishing Theorem 1.1 (resp., Theorem 1.2). Lastly, in Section 3.5, we employ the ideas developed in Sections 3.2–3.3 in order to present a worst-case to average-case reduction for the problem of counting t-cycles (in unweighted graphs).

3.1 The interactive proof system

As in Section 2.1, we shall reduce the evaluation of $\mathtt{CWC}_t^p$ to a single evaluation of $\mathtt{CWC}_{t-1}^p$, by considering the following polynomials over $\mathcal{F} = \mathrm{GF}(p)$ (for $j,k \in [n]$):

$$f_{j,k}(z) \stackrel{\text{def}}{=} \sum_{i\in[n]} \mathtt{EQ}_i(z) \cdot w_{j,k}^{(i)} \tag{12}$$

where $\mathtt{EQ}_i : \mathcal{F} \to \mathcal{F}$ is (a degree $n-1$ polynomial) as in Section 2.1. Recall that $f_{j,k}(i) = w_{j,k}^{(i)}$ for every $i \in [n]$, and that each $f_{j,k}$ can be computed using $O(n)$ field operations (when given W).

Construction 3.1 (Construction 2.1, revised): *The iteration starts with a claim of the form $\mathtt{CWC}_t^p(W) = v$, where $W \in \mathcal{F}^{n\times n}$ and $v \in \mathcal{F}$ are determined before.*

1. *The prover computes the $\left(\binom{t}{2} + t\right) \cdot (n-1)$ degree polynomial $P_{t-1} : \mathcal{F} \to \mathcal{F}$, where*

$$P_{t-1}(z) \stackrel{\text{def}}{=} \sum_{S\in\binom{[n]}{t-1}} \prod_{j\le k\in S} f_{j,k}(z), \tag{13}$$

   and sends it to the verifier.

   Note that $P_{t-1}(z)$ can be computed by interpolation, using the values of P_{t-1} at less than $t^2 n$ points, and that for every $\ell \in \mathcal{F}$ it holds that $P_{t-1}(\ell) = \mathtt{CWC}_{t-1}(W^{(\ell)})$, where $w_{j,k}^{(\ell)} = f_{j,k}(\ell)$ for every $j,k \in [n]$.

2. *Upon receiving a polynomial $\widetilde{P}$ of degree at most $t^2 n$, the verifier checks whether $t \cdot v \equiv \sum_{i\in[n]} w_{i,i} \cdot \widetilde{P}(i) \pmod p$, and rejects if equality does not hold. If equality holds, the verifier selects uniformly $r \in \mathcal{F}$, and sends it to the prover.*

The iteration ends with the claim that $\mathtt{CWC}_{t-1}^p(W') = v'$, where $v' = \widetilde{P}(r)$ and $w'_{j,k} = f_{j,k}(r)$ for every $j,k \in [n]$.

Recall that both parties can compute each of the $f_{j,k}$'s using $O(n)$ field operation, which implies that the verifier strategy can be implemented using $\widetilde{O}(n^3 + t^2 n^2)$ field operations. (Actually, the verifier can be implemented using $O(t^2 n^2)$ field operations; see Proposition 3.3.) In addition, the prover needs to construct the polynomial P_{t-1}, and a straightforward way of doing so can be implemented using $O(n^t)$ field operations. As in the previous section, we shall see that the complexity can be improved to $O(t \cdot n^{1+\omega_{\mathtt{mm}}\lceil (t-1)/3\rceil})$, where $\omega_{\mathtt{mm}}$ is the matrix multiplication exponent. But again, we first state the effect of a single iteration.

Proposition 3.2 (analysis of Construction 3.1): *Let W, v, W' and v' be as in Construction 3.1.*

1. *If $\mathtt{CWC}_t^p(W) = v$ and both parties follow their instructions, then $\mathtt{CWC}_{t-1}^p(W') = v'$ holds.*
2. *If $\mathtt{CWC}_t^p(W) \neq v$ and the verifier follows its instructions, then either the verifier rejects or $\mathtt{CWC}_{t-1}^p(W') = v'$ holds with probability at most $t^2 n/|\mathcal{F}|$.*

The proof of Proposition 3.2 is analogous to the proof of Proposition 2.2. The key observation here is that, for every $i \in [n]$, it holds that

$$\begin{aligned}
P_{t-1}(i) &= \sum_{S \in \binom{[n]}{t-1}} \prod_{j \leq k \in S} f_{j,k}(i) \\
&= \sum_{S \in \binom{[n]}{t-1}} \prod_{j \leq k \in S} w_{j,k}^{(i)} \\
&= \mathtt{CWC}_{t-1}^{p}(W^{(i)}),
\end{aligned}$$

and so $t \cdot \mathtt{CWC}_t(W) \equiv \sum_{i \in [n]} w_{i,i} \cdot P_{t-1}(i) \pmod{p}$. More generally, for every $\ell \in \mathcal{F}$, it holds that $P_{t-1}(\ell) = \mathtt{CWC}_{t-1}(W^{(\ell)})$, where $w_{j,k}^{(\ell)} = f_{j,k}(\ell)$ for every $j, k \in [n]$.

Proposition 3.3 (implementing the verifier of Construction 3.1): *The mapping $(W, \ell) \mapsto (f_{j,k}(\ell))_{j,k \in [n]}$ can be computed in $O(n^2)$ field operations.*

Proof: We rewrite Eq. (12) as

$$f_{j,k}(z) \stackrel{\text{def}}{=} w_{j,k} \cdot \sum_{i \in [n]} \mathtt{EQ}_i(z) \;+\; \sum_{i \in [n]} \mathtt{EQ}_i(z) \cdot E_{j,k}^{(i)} \tag{14}$$

where $E_{j,k}^{(i)} = w_{j,k}^{(i)} - w_{j,k}$. The key observation is that $E_{j,k}^{(i)} = 0$ for $j \neq k$, so we need to compute $E_{j,k}^{(i)}$ only for n^2 values of $i, j, k \in [n]$, and each $E_{j,k}^{(i)}$ can be computed in a constant number of field operations (see Eq. (11)). Computing all the $\mathtt{EQ}_i$'s can be done in $O(n^2)$ field operations, which allows to compute $n+1$ linear combinations of these values (i.e., $\sum_{i \in [n]} \mathtt{EQ}_i(z)$ and $\sum_{i \in [n]} E_{j,j}^{(i)} \cdot \mathtt{EQ}_i(z)$ for $j \in [n]$) in $O(n^2)$ field operations. ■

Theorem 3.4 (Theorem 2.3, slightly revised): *For an efficiently computable function $t : \mathbb{N} \to \mathbb{N}$, let S_t denote the set of pairs $(W, \mathtt{CWC}_{t(n)}(W))$ such that W is a symmetric n-by-n Boolean matrix. Then, S_t has a $(t-2)$-round* (public coin) *interactive proof system* (of perfect completeness) *in which the verifier's running time is $\widetilde{O}(t(n)^3 \cdot n^2)$, and the prover's running time is dominated by $O(t(n)^3 \cdot n)$ calls to the oracle $\overline{\mathtt{CWC}}_{t(n)-1}^{p}$, where p is a prime number in $[n^{t(n)}, 2n^{t(n)}]$, and $\overline{\mathtt{CWC}}_t^{p}$ answers the query $(i, W) \in [t] \times \mathrm{GF}(p)^{n \times n}$ with $\mathtt{CWC}_i^{p}(W)$.*

Proof: We proceed as in the proof of Theorem 2.3. On input W and N (which is supposed to equal $\mathtt{CWC}_{t(n)}(W)$), the parties pick a finite field $\mathcal{F} = \mathrm{GF}(p)$ such that $p \in [n^{t(n)}, 2n^{t(n)}]$, and iteratively invoke Construction 3.1. In case the verifier did not reject (in any of the $t(n)-2$ iterations), it is left with a claim of the form $\mathtt{CWC}_2^{p}(W') = v'$, which it can verify by itself using $O(n^2)$ field operations (since $\mathtt{CWC}_2(W')$ equals $\sum_{j<k \in [n]} w'_{j,j} \cdot w'_{j,k} \cdot w'_{k,k}$). ■

Digest. Constructions 2.1 and 3.1 combine a downward reduction of the (clique size) parameter t with a *batch verification* of many claims to the verification of a single claim of similar type. Such batch verification is implicit in the celebrated sum-check protocol [27] and the notion was made explicit and applied in a wider context in [30]. Nevertheless, the current constructions catch the eye in applying batch verification to a more natural problem and doing so in a more intuitive manner. Specifically, in Construction 2.1 numerous instances of counting $(t-1)$-cliques in weighted versions of a graph are reduced to such a single instance, whereas in Construction 3.1 numerous instances of a generic problem (with no fixed parameter) are reduced to a single instance of the same problem.

3.2 Reducing counting edge-weighted cliques to the unweighted case

Analogously to Proposition 2.6, we show that computing $\mathtt{CWC}_t$ can be reduced to counting the number of t-cliques in simple unweighted graphs. We present the reduction in two explicit steps, which correspond to the two steps in the proof of Proposition 2.6, first reducing the magnitude of the weights in the matrix, and then reducing to the case of simple graphs (with no weights).

Proposition 3.5 (reducing the weights in the computation of $\mathtt{CWC}_t$): *The computation of $\mathtt{CWC}_t$ for matrices with entries in $[[m]]$ can be reduced in $\widetilde{O}(t^2n^2\log m)$-time to computing $\mathtt{CWC}_t^p$ for primes $p \in [t^2\log(nm), 2t^2\log(nm)]$. The reduction makes less than $t^2\log(nm)$ queries, each referring to an n-by-n matrix.*

Proof: As in the first part of the proof of Proposition 2.6, we merely use the Chinese Remainder Theorem, while relying on the fact that $\mathtt{CWC}_t(W)$ resides in the interval $[0, n^t \cdot m^{t^2}]$. Specifically, given a matrix $W \in [[m]]^{n\times n}$, for each prime $p \in [t^2\log(nm), 2t^2\log(nm)]$, we query $\mathtt{CWC}_t^p$ on $W^{(p)} = W \bmod p$ (i.e., $W^{(p)} = (w_{j,k} \bmod p)_{j,k\in[n]}$). ■

While the overhead of the foregoing reduction is polynomial in t and $\log m$, the overhead of the following reduction is exponential in these values. Hence, the following reduction is applied only for small values of t (e.g., $t = O(1)$) and small weights (i.e., small p's).

Theorem 3.6 (reducing $\mathtt{CWC}_t^p$ to counting t-cliques in graphs): *The computation of $\mathtt{CWC}_t^p : \mathrm{GF}(p)^{n\times n} \to \mathrm{GF}(p)$ can be reduced in $\widetilde{O}(p^{t^2}n^2)$-time to computing the number of t-cliques in an unweighted graph having $n'' \stackrel{\rm def}{=} \widetilde{O}(2^{t^2\lceil\log_2 p\rceil} \cdot n)$ vertices. Furthermore, $\mathtt{CWC}_t(W) = \mathtt{CWC}_t(T(W))/t!$, where $T : \mathrm{GF}(p)^{n\times n} \to \{0,1\}^{n''\times n''}$ is a $\widetilde{O}(p^{t^2}n^2)$-time computable one-to-one mapping and $T(W)$ is an n''-by-n'' symmetric and reflexive Boolean matrix* (which represents an n''-vertex graph).

(Note that n'' is intentionally defines as a function of $\lceil\log_2 p\rceil$ rather than as a function of p; hence, the problems of computing $\mathtt{CWC}_t^p$, for all $p \in [2^{\ell(n)-1}, 2^{\ell(n)}]$ are reduced to the same clique counting problem.)

Proof: Viewing the (symmetric) matrix $W \in \mathrm{GF}(p)^{n\times n}$ as an n-vertex (complete) graph $G = ([n], \binom{[n]}{2})$ with vertex and edge weights, we first get rid of the vertex weights. This is done exactly as in (the second part of the proof of) Proposition 2.6; that is, by replacing each vertex (of G) by an independent set of size that equals its weight, which is in $[p]$, and placing complete bipartite graphs between these independent sets with edge weights that equal the weight of the corresponding edge. (That is, the vertex v is replaced by an independent set of size $w_{v,v}$, denoted I_v, and the edge between u and v is replaced by a complete bipartite graph between I_u and I_v such that each edge in this bipartite graph has weight $w_{u,v}$.) Hence, we derive a graph with $\widetilde{n} \stackrel{\rm def}{=} \sum_{v\in[n]} w_{v,v} \le n' \stackrel{\rm def}{=} n \cdot p$ vertices. Augmenting this graph with $n' - \widetilde{n}$ isolated vertices, we obtain an n'-vertex graph G' with edge weights in $[p]$. We denote the weight of edge $e = \{u, v\}$ by w_e (whereas all vertex weights are set to 1).

Next, we construct a graph G'' that consists of $p^{\binom{t}{2}}$ isolated copies of graphs of the form $G''_{\bar{i}}$, where $\bar{i} = (i_{j,k})_{j<k\in[t]} \in [p]^{\binom{t}{2}}$, and each graph $G''_{\bar{i}}$ consists of t independent sets such that each pair of sets is connected by a *subgraph* of the double-cover of G'. (Recall that the double-cover of $G' = ([n'], E')$ is a bipartite graph B' with n' vertices on each side such that $\langle 1, u\rangle$ and $\langle 2, v\rangle$ are connected in B' if and only if $\{u, v\} \in E'$.) Specifically, for each $\bar{i} = (i_{j,k})_{j<k\in[t]} \in [p]^{\binom{t}{2}}$, the graph $G''_{\bar{i}}$ consists of the vertex-set $\{\langle\bar{i}, j, v\rangle : j \in [t] \,\&\, v \in [n']\}$ such that vertices $\langle\bar{i}, j, v\rangle$ and $\langle\bar{i}, k, u\rangle$ are connected if and only if ($j \neq k$ and) $w_{\{v,u\}} \geq i_{j,k}$. We stress that the crux of the construction is that, for $j \neq k$, the vertices $\langle\bar{i}, j, v\rangle$ and $\langle\bar{i}, k, u\rangle$ are connected if and only if $i_{j,k} \in [w_{\{v,u\}}]$.

Note that in the case that all weights equal p (i.e., $w_{\{v,u\}} = p$ for every $\{u, v\} \in E'$), each graph $G''_{\bar{i}}$ consists of t independent sets that are connected by double-covers of G'. In this case, each t-clique in G' yields $t!$ cliques of size t in each $G''_{\bar{i}}$, where these $t!$ cliques correspond to all possible permutations over $[t]$; specifically, for each clique $\{v_1, ..., v_t\}$ in G' and each permutation π over $[t]$, the set $\{\langle\bar{i}, \pi(1), v_1\rangle, ..., \langle\bar{i}, \pi(t), v_t\rangle\}$ is a clique in $G''_{\bar{i}}$. On the other hand, the only t-cliques in $G''_{\bar{i}}$ are t-subsets of the form $\{\langle\bar{i}, 1, v_1\rangle, ..., \langle\bar{i}, t, v_t\rangle\}$ such that $\{v_1, ..., v_t\}$ is a t-clique in G'.

In the general case (of arbitrary edge weights $(w_e)_{e\in E'} \in [p]^{|E'|}$), for each permutation π over $[t]$, each t-clique $\{v_1, ..., v_t\}$ in G', yields the clique $\{\langle\bar{i}, \pi(1), v_1\rangle, ..., \langle\bar{i}, \pi(t), v_t\rangle\}$ in $G''_{\bar{i}}$ if and only if for all $j < k \in [t]$ it holds that $i_{\pi(j),\pi(k)} \in [w_{\{v_j,v_k\}}]$. Hence, for each permutation π, an "image" of this t-clique appears in $\prod_{j<k} w_{\{v_j,v_k\}}$ of the graphs $G''_{\bar{i}}$. On the other hand, the only t-cliques in G'' have the form $\{\langle\bar{i}, 1, v_1\rangle, ..., \langle\bar{i}, t, v_t\rangle\}$ such that $\{v_1, ..., v_t\}$ is a t-clique in G' and $i_{j,k} \in [w_{\{v_j,v_k\}}]$ holds for all $j \neq k \in [t]$.

To summarize, denoting by $W' = (w'_{j,k})$ the matrix that corresponds to the edge weights in G' (i.e., $w'_{j,j} = 1$ and $w'_{j,k} = w_{\{j,k\}}$ if $\{j, k\}$ is an edge of G' (and $w_{j,k} = 0$ otherwise)), we have $\mathtt{CWC}_t(W) = \mathtt{CWC}_t(W')$ and $t! \cdot \mathtt{CWC}_t(W')$ equals the number of t-cliques in G''. ■

Remark 3.7 (reducing $\mathtt{CWC}_i^p$ to counting t-cliques, where $i \in [t-1]$): *Generalizing Theorem 3.6, for every $i \in [t]$, we obtain a mapping $T^{(i)} : \mathrm{GF}(p)^{n\times n} \to \mathrm{GF}(2)^{n^{(i)}\times n^{(i)}}$ such that $\mathtt{CWC}_i(W) = \mathtt{CWC}_i(T^{(i)}(W))/i!$ and $n^{(i)} = \widetilde{O}(2^{i^2\lceil\log_2 p\rceil} \cdot n)$, where indeed $T^{(t)} = T$ and $n^{(t)} = n''$. Wishing to reduce all $\mathtt{CWC}_i$'s to counting t-cliques, we* augment *the graph produced by $T^{(i)}$ with $n'' - n^{(i)} - 2(t-i)$ isolated vertices and a clique of size $t-i$ that is connected by a complete biparitite graph to all original vertices* (in the graph produced by $T^{(i)}$). *Observing that the original graph produced by $T^{(i)}$ has no cliques of size greater than i, it follows that there is a one-to-one correspondance between the i-cliques in the original graph and the t-cliques in the augmented graph. We denote the augmented graph derived from W when wishing to compute $\mathtt{CWC}_i(W)$ by $T'(i, W)$, and note that $T'(i, W)$ has $n'' - (t-i)$ vertices.*

Corollary 3.8 (on implementing the prover of Theorem 3.4): *For every fixed t and $p = O(n^t)$, the computation of $\mathtt{CWC}_t^p : \mathrm{GF}(p)^{n\times n} \to \mathrm{GF}(p)$ can be reduced in $\widetilde{O}(n^2)$-time to computing the number of t-cliques in unweighted graphs having $\widetilde{O}(n)$ vertices.*

(Indeed, the $\widetilde{O}$ notation hides factors that are exponential in t^2.)

Proof: The reduction of Proposition 3.5 reduces the computation of $\mathtt{CWC}_t^p$ to few computations of $\mathtt{CWC}_t^q$ for primes $q = O(t^2 \log(np)) = O(\log n)$. The reduction of Theorem 3.6 reduces each such computation to counting t-cliques in a graph with $\widetilde{O}(q^{t^2} \cdot n) = \widetilde{O}(n)$ vertices. Analogously, computing $\mathtt{CWC}_i^p$ is reduced to counting t-cliques in $\widetilde{O}(n)$-vertex graphs (by using Remark 3.7). ■

3.3 The worst-case to average-case reduction

We focus on the worst-case and average-case problems of computing $\mathtt{CWC}_t^p : \mathrm{GF}(p)^{n\times n} \to \mathrm{GF}(p)$, while allowing t and p to vary with n. While it is natural to assume that $t = t(n)$ is easily determined given n, we cannot make this assumption regarding the determination of a prime $p = p(n)$ of size $\Theta(n^t)$, since determining such a prime may take time $\Omega(n^t)$, whereas our focus here is on reductions that run much faster (i.e., they must be certainly faster than the complexity of computing $\mathtt{CWC}_t$).[14] Indeed, we can resolve the problem by adopting some form of Cramer's conjecture (which asserts that the interval $[m, m+O(\log^2 m)]$ contains a prime), but prefer not to make such assumptions.[15]

For starters, we shall ignore the foregoing issue, and consider the problem Π_n^p of computing $\mathtt{CWC}_t^p : \mathrm{GF}(p)^{n\times n} \to \mathrm{GF}(p)$, where n, p and t are viewed as generic (and are given to all algorithms as auxiliary inputs). We shall later consider the problem Π_n in which the instances are pairs of the form (p, W) such that p is an $\ell(n)$-bit long prime and $W \in \mathrm{GF}(p)^{n\times n}$, and the problem is to compute $\mathtt{CWC}_{t(n)}^p(W)$.

3.3.1 The reduction of $\mathtt{CWC}_t^p$ for fixed t and p

For sake of asymptotic presentation, we let $t = t(n)$ and $p = p(n)$ be functions of n. Recall that we assume that the reductions asserted next are given n, t and p as auxiliary inputs.

Theorem 3.9 (worst-case to average-case reduction for Π_n^p): *Fixing functions $t : \mathbb{N} \to \mathbb{N}$ and $p : \mathbb{N} \to \mathbb{N}$ such that $p(n)$ is a prime number in $[\omega(t(n)^2), n^{O(t(n))}]$, we let Π_n^p denote the problem of computing $\mathtt{CWC}_{t(n)}^{p(n)}(W)$ for n-by-n matrices W over $\mathrm{GF}(p(n))$. Then, Π_n^p is randomly self-reducible in $\widetilde{O}(t(n)^3 \cdot n^2)$ time with $t(n)^2$ queries.[16] Furthermore, there is a worst-case to average-case reduction of Π_n^p to itself that makes $O(t(n)^2)$ queries, runs in $\widetilde{O}(t(n)^3 \cdot n^2)$ time, and outputs the correct value with probability $2/3$ provided that the error rate of the average-case solver is a constant smaller than one half.*

Note that Theorem 3.4 extends to the set of tuples $(n, p(n), W, \mathtt{CWC}_{t(n)}^{p(n)}(W))$ such that $n \in \mathbb{N}$ and $W \in \mathrm{GF}(p(n))^{n\times n}$. Recall that (by Corollary 3.8), for every fixed t, the computation of $\mathtt{CWC}_t^p$ can be reduced in $\widetilde{O}(n^2)$-time to computing the number of t-cliques in unweighted $\widetilde{O}(n)$-vertex graphs. Alternatively, $\mathtt{CWC}_t^p$ can be computed in time $\widetilde{O}(n^{\omega_{\mathtt{mm}}\lceil t(n)/3\rceil})$, where $\omega_{\mathtt{mm}}$ is the matrix multiplication exponent (by extending the ideas of [29]; see Appendix).

Proof: Fixing $t = t(n) > 1$ and $p = p(n)$, we let $\mathcal{F} = \mathrm{GF}(p(n))$. For any $W, R \in \mathcal{F}^{n\times n}$, consider the univariate polynomial $P_t(z) = \mathtt{CWC}_t^p(W + zR)$, where the arithmetic is over $\mathcal{F}$. Recalling Eq. (9), observe that P_t has degree $t + \binom{t}{2} < t^2$, and so for every W and R, the value of $\mathtt{CWC}_t^p(W)$ can be obtained by querying P_t at t^2 points. Hence, given W, the random self-reducibility process

[14] Indeed, this issue is less acute in the context of interactive proofs, since we may instruct the parties to select such a prime at random (rather than determine it).

[15] Indeed, for our purposes, it suffices to assume that interval $[m, m + \mathrm{poly}(\log m)]$ contains a prime. Actually, we can get meaningful results even when only assuming that interval $[m, m + m^{o(1)}]$ contains a prime.

[16] Recall that a problem is is randomly self-reducible in time t with q queries if there exists an oracle machine of time complexity t and query complexity q that solves the problem (in the worst-case) by making uniformly distributed queries to the problem itself. (We stress that the queries may depend on one another; it is only required that each query is uniformly distributed among the problem's instances.) Indeed, such a reduction yields a worst-case to average-case reduction that supports average-case error rate of $1/3t$.

select $R \in \mathcal{F}^{n\times n}$ uniformly at random, and queries the corresponding polynomial at the points $1, ..., t^2$. Note that these queries correspond to t^2 queries to $\texttt{CWC}_t^p$ such that each query is uniformly distributed in $\mathcal{F}^{n\times n}$ (by virtue of the random R).

Recovery under error rate of 1/9 is possible by picking a random R, querying the corresponding polynomial on $3t^2$ (non-zero) points, and employing the Berlekamp–Welch algorithm. In this case, with probability at least 2/3, at least a 2/3 fraction of the queries are answered correctly. (Note that each of these queries corresponds to a collection of n^2 points on n^2 random lines that pass through W at their origin.) To support an error rate of $\eta < 1/2$ (equiv., a success rate of $0.5+\epsilon$ for $\epsilon = 0.5-\eta$), we pick a random collection of n^2 curves of degree two, denoted $C_{i,j} : \mathcal{F} \to \mathcal{F}$, that pass (at their origin) through the n^2 corresponding entries of W (i.e., $C_{i,j}(0) = w_{i,j}$), and consider the polynomial (of degree at most $2t^2$) that represents the value of $\texttt{CWC}_t^p(C(z))$, where $C(z) = (C_{i,j}(z))$. In this case, letting $\epsilon = 0.5 - \eta$, with probability at least 2/3, at least a $0.5 + 0.5 \cdot \epsilon$ fraction of the $O(t^2/\epsilon)$ queries are answered correctly, which suffices for correct decoding. ■

3.3.2 The reduction of Π_n

Here, for efficiently computable functions $t, \ell : \mathbb{N} \to \mathbb{N}$, we consider the problem Π_n in which the instances are pairs of the form (p, W) such that p is an $\ell(n)$-bit long prime and $W \in \mathrm{GF}(p)^{n\times n}$, and the problem is to compute $\texttt{CWC}_{t(n)}^p(W)$.

Showing a worst-case to average-case reduction for Π_n is more complex than doing so for Π_n^p, because when given the input (p, W) it may be the case that the average-case solver just fails on all inputs of the form $(p, \cdot)$. Hence, we shall use Chinese Remaindering (with errors [19]) in order to obtain the value of $\texttt{CWC}_{t(n)}^p(W)$ (or rather $\texttt{CWC}_{t(n)}(W)$) from the values of $\texttt{CWC}_{t(n)}^{p'}(W)$ for other primes $p' \in [2^{\ell(n)-1}, 2^{\ell(n)}]$.

Theorem 3.10 (worst-case to average-case reduction for Π_n):[17] *Let $\ell : \mathbb{N} \to \mathbb{N}$ be such that $\ell(n) \in [3 \log t(n) + \log\log n + \omega(1), O(t(n) \log n)]$. Then, there exists a worst-case to average-case reduction of Π_n to itself that makes $\widetilde{O}(t(n)^5 \log n)$ queries, runs in $\widetilde{O}(t(n)^6 \cdot n^2)$ time, and outputs the correct value with probability 2/3 provided that the error rate of the average-case solver is a constant smaller than one fourth.*

Proof: Denoting the error rate of the average-case solver by $\eta < 1/4$, and letting $\epsilon = 1/4 - \eta$, we first observe that for at least a $0.5 + \epsilon$ fraction the primes in $I_n \stackrel{\rm def}{=} [2^{\ell(n)-1}, 2^{\ell(n)}]$, the solver has an error rate of at most $0.5 - \epsilon$; that is, for each such prime p, hereafter called good, the solver solves Π_n^p correctly on at least a $0.5 + \epsilon$ fraction of the instances. Hence, for each good prime, we can apply the reduction presented in the proof of Theorem 3.9, but the problem is that the prime that is part of the worst-case instance may not be good.

Hence, on input (p, W), where $p \in I_n$ and $W \in \mathrm{GF}(p)^{n\times n}$, we first try to obtain the (integer) value of $\texttt{CWC}_{t(n)}(W)$, and then reduce the result modulo p. Basically, we shall obtain the value of $\texttt{CWC}_{t(n)}(W)$ by selecting $m = O(\epsilon^{-1}t(n)^3 \log n)/\ell(n)$ random primes $p_1, ..., p_m$ in I_n, hoping that at least $(0.5 + 0.5\epsilon) \cdot m$ of them are good, and combining the values $(\texttt{CWC}_{t(n)}^{p_i}(W))_{i\in[m]}$ using Chinese Remaindering with errors [19, Sec. 3]. The analysis of the latter decoding relies on the fact that the (non-negative) value of $\texttt{CWC}_{t(n)}(W)$ is bounded above by $\binom{n}{t(n)} \cdot (2^{\ell(n)+1})^{t(n)^2} < n^{O(t(n)^3)}$, whereas the

[17] Recall that Π_n is defined in terms of of the functions $t, \ell : \mathbb{N} \to \mathbb{N}$. It refers to instances of the form (p, W) such that p is an $\ell(n)$-bit long prime and $W \in \mathrm{GF}(p)^{n\times n}$, and calls for computing $\texttt{CWC}_{t(n)}^p(W)$.

product of the smallest ϵm primes in I_n exceeds $(2^{\ell(n)-1})^{\epsilon m} = \exp(O(t(n)^3 \log n))$.[18] Specifically, on input (p, W), we proceed as follows.

1. Select at random $m = \frac{O(\epsilon^{-1} \cdot t(n)^3 \log n)}{\ell(n)}$ primes in I_n.
2. For each selected prime, denoted p_i, invoke the worst-case to average-case procedure for $\Pi_n^{(p_i)}$ on the instance (p_i, W), and denote the result by v_i. (The aforementioned reduction is the one presented in the proof of Theorem 3.9, except that the error probability should be reduced to $1/3m$.)[19]
3. Apply Chinese Remaindering with errors (for error rate $0.5 - 0.5\epsilon$) on $v_1, ..., v_m$, and output the result reduced modulo p.

The theorem follows by using the fact that, with high probability, at least a $0.5 + 0.5\epsilon$ fraction of the primes selected in Step 1 are good. ■

Corollary 3.11 (worst-case to average-case reduction for counting cliques): *Let t be a constant and $b(n) = \Theta(t^3 \log n)$. Let $\mathcal{G}_n$ be a distribution on $\widetilde{O}(n)$-vertex graphs obtained by selecting a prime $p \in [b(n), 2b(n)]$ and $W \in \mathrm{GF}(p)^{n \times n}$, and outputting $T(W)$ where T is the mapping presented in Theorem 3.6. Then, there exists a worst-case to average-case reduction of counting t-cliques in n-vertex graphs to counting t-cliques in graphs generated according to $\mathcal{G}_n$ such that the reduction runs in $\widetilde{O}(n^2)$ time, makes $\widetilde{O}(\log n)^2$ queries, and outputs the correct value with probability $2/3$ provided that the error rate of the average-case solver is a constant smaller than one fourth.*

Proof: Given an n-vertex graph G, using Proposition 3.5, we reduce counting the number of t-cliques in G to making $O(\log n)$ queries to oracles of the form $\mathtt{CWC}_t^p$ such that p is a prime in $[b(n), 2b(n)]$. Next, setting $\ell(n) = \lceil \log b(n) \rceil$ (and using Theorem 3.10), we reduce answering each of these queries to solving the problem Π_n on at least $0.75 + \epsilon$ of the instances, where $\epsilon > 0$ is an arbitrary constant. (We do so after reducing the error probility of the reduction to $o(1/\log n)$.) Lastly, using the mapping T of Theorem 3.6, we map the $\widetilde{O}(t^5 \log n)$ random queries made by the worst-case to average-case reduction to $\widetilde{O}(n)$-vertex graphs. We stress that a procedure that counts t-cliques in $\mathcal{G}_n$ correctly with probability $0.75 + \epsilon$, yields a procedure that answers Π_n correctly with probability $0.75 + \epsilon$. ■

3.3.3 Length reduction

Corollary 3.11 falls short from establishing Theorem 1.1 only in one aspect: It reduces worst-case n-vertex graph instances to average-case instances that are n''-vertex graphs, for $n'' = \widetilde{O}(n)$. Wishing to have a worst-case to average-case reduction that preserves the number of vertices (in the corresponding instances), we seek a reduction that reduces the number of vertices in the graph. It seems easiest to present such a reduction in the worst-case setting. Indeed, we show a reduction

[18] Specifically, if $\ell(n) \leq c \cdot t(n) \log n$, then letting $m = 2c \cdot (\epsilon^{-1} t(n)^3 \log n)/\ell(n)$ will do, since $\binom{n}{t(n)} \cdot (2^{\ell(n)+1})^{t(n)^2} < n^{(c+o(1)) \cdot t(n)^3}$, whereas $(2^{\ell(n)-1})^{\epsilon m} = 2^{(1-o(1)) \cdot 2c \cdot t(n)^3 \log n}$. The unique decoding condition in [19] essentially requires an error rate of $0.5 - \frac{0.5k}{m}$, assuming that the product of the smallest k (out of m) primes exceeds the encoded (non-zero) integer.

[19] Hence, each invokation generates $O(t(n)^2 \log m)$ queries, totaling in $\widetilde{O}(m) \cdot t(n)^2$ queries.

of counting t-cliques in n-vertex graphs to counting t-cliques in n'-vertex graphs such that $n' = n/\text{poly}(\log n)$. Applying Corollary 3.11 to the resulting n'-vertex graphs, we reduce to n''-vertex graphs such that $n'' = \widetilde{O}(n') = n$.

Proposition 3.12 (reducing the number of vertices in the t-clique counting problem): *Let t be a constant and $k : \mathbb{N} \to \mathbb{N}$ such that $k(n) < n$. Then, there exists an $O((n/k(n))^t \cdot k(n)^2)$-time reduction of counting t-cliques in n-vertex graphs to counting t-cliques in $k(n)$-vertex graphs. Furthermore, the reduction performs $O(n/k(n))^t$ queries.*

Proof: Consider an arbitrary partition of $[n]$ into $m = \lceil t \cdot n/k(n) \rceil$ sets $V_1, ..., V_m$ such that $|V_i| \le k(n)/t$ (for every $i \in [m]$). For every $I \subset [m]$ of size at most t, let $V_I = \cup_{i \in I} V_i$ and G_I be the subgraph of G induced by V_I. Next, *augment* each G_I to a $k(n)$-vertex graph, denoted G'_I, by possiblly adding $k(n) - |V_I|$ isolated vertices (and note that the t-cliques in G'_I are exactly those in G_I). Observe that for each t-clique of G there exists a unique I (of size at most t) such that this t-clique appears in G_I but does not appear in any $G_{I'}$ such that $I' \subset I$. Letting N_I denote the number of t-cliques that appear in G_I but do not appear in any $G_{I'}$ such that $I' \subset I$, it follows that the number of t-cliques in G equals $\sum_{i \in [t]} \sum_{I \in \binom{[m]}{i}} N_I$. Hence, on input $G = ([n], E)$, the reduction proceeds as follows.

1. For each $I \in \cup_{i \in [t]} \binom{[m]}{i}$, it queries for the number of t-cliques in G'_I, denoting the result by r_I.
2. It computes all N_I based on the values obtained in Step 1. Specifically, for $i = 1, ..., t$, and for every $I \in \binom{[m]}{i}$, it sets $N_I \leftarrow r_I - \sum_{I' \subset I} N_{I'}$, where $N_\emptyset = 0$.

The reduction outputs the sum of all the N_I's. Observing that the reduction makes $\sum_{i \in [t]} \binom{m}{i} < m^t$ queries, the claim follows. ■

Proof of Theorem 1.1: Combining Corollary 3.11 with Proposition 3.12 yields Theorem 1.1. Specifically, for a suitable polynomial p, we set $k(n) = n/p(\log n)$, and reduce counting t-cliques in n-vertex graphs to counting them in $k(n)$-vertex graphs (using Proposition 3.12). Then, we apply Corollary 3.11 with n replaced by $k(n)$, reducing the worst-case problem for $k(n)$-vertex graphs to an average-case problem regarding the distribution $\mathcal{G}_{k(n)}$, which is supported by $\widetilde{O}(k(n))$-vertex graphs. Indeed, a suitable choice of the polynomial p implies that $\widetilde{O}(k(n)) = \widetilde{O}(n)/p(\log n) \le n$, and Theorem 1.1 follows (possibly by augmenting the graph with isolated vertices). ■

3.4 The worst-case to rare-case reduction

In Section 3.3, we considered worst-case to average-case reductions, where average-case refers to a constant error rate. Specifically, in Theorem 3.9 we required the error rate to be smaller than 1/2, whereas in Theorem 3.10 the error rate was required to be smaller than 1/4. Here we aim to increase the error rate to almost 1 (i.e., deal with error rate that is merely bounded away from 1). Equivalently, we consider solvers that provide the correct answer quite rarely; that is, they answer correctly on a small fraction of the instances. We shall seek and present worst-case to rare-case reductions.

For technical convenience, we shall be dealing with a small variant of the problems considered in Section 3.3. Specifically, we shall first consider the problem $\overline{\Pi}_n^p$ in which one is given $W \in \text{GF}(p)^{n \times n}$

and 1^i such that $i \in [t(n)]$, and the task is to compute $\texttt{CWC}_i^p(W)$, where (again) n, p and $t = t(n)$ are viewed as generic (and are given to all algorithms as auxiliary inputs).[20] Note that the length of the input $(W, 1^i)$ is $n^2\lceil\log_2 p\rceil + i$, and that $n^2\lceil\log_2 p\rceil + 1 > (n-1)^2\lceil\log_2 p\rceil + t(n-1)$, since $t(n-1) \leq n-1$ (w.l.o.g.). We shall later consider the problem $\overline{\Pi}_n$ in which the instances have the form $(p, W, 1^i)$ such that p is an $\ell(n)$-bit long prime, $W \in \mathrm{GF}(p)^{n\times n}$, and $i \in [t(n)]$, and the problem is to compute $\texttt{CWC}_i^p(W)$. (Here we shall use $n^2\ell(n)+1 > (n-1)^2\ell(n-1)+t(n-1)$, which presumes that ℓ is non-decreasing.)

3.4.1 The reduction of $\overline{\texttt{CWC}}_t^p$ for fixed t and p

As in Section 3.3, we start by treating $p = p(n)$ as if it are fixed (and waive this postulate later (in Section 3.4.2).

Theorem 3.9 provides a worst-case to average-case reduction, where average-case refers to error rate below one half. Here we aim to increase the error rate to almost 1 (i.e., deal with error rate that is merely bounded away from 1). Equivalently, we consider solvers that provide the correct answer quite rarely; that is, they answer correctly on a small fraction of the instances. Recall that we assume that the reductions asserted next are given n, t and p as auxiliary inputs.

Theorem 3.13 (worst-case to rare-case reduction for $\overline{\Pi}_n^p$): *Fixing efficiently computable functions $t, p : \mathbb{N} \to \mathbb{N}$ and $\rho : \mathbb{N} \to (0,1]$ such that t and $1/\rho$ are monotonically non-decreasing and $p(n) \leq n^{\mathrm{poly}(t(n))}$, there exists a worst-case to rare-case reduction of $\overline{\Pi}_n^{p(n)}$ to itself that makes $\mathrm{poly}(t(n)/\rho(n)) \cdot \widetilde{O}(n)$ queries, runs in $\mathrm{poly}(t(n)/\rho(n)) \cdot \widetilde{O}(n^2)$ time, and outputs the correct value with probability $2/3$ provided that the success rate of the rare-case solver is at least $\rho(n)$* (and $\rho(n) > p(n)^{-1/3}$).

Note that Theorem 3.4 extends to the set of tuples $(n, p(n), W, 1^i, \texttt{CWC}_i^{p(n)}(W))$ such that $n \in \mathbb{N}$ and $W \in \mathrm{GF}(p(n))^{n\times n}$.

Overview of the proof of Theorem 3.13. Fixing $n \in \mathbb{N}$, $p = p(n)$ and $t \in [t(n)]$, we consider the formal polynomial $\texttt{CWC}_t^p(Z)$, where $Z = (z_{j,k})$ is an n-by-n matrix of formal variables, and note that this is an n^2-variate polynomial of total degree at most t^2. Given a rare-case solver, which solves $\texttt{CWC}_t^p$ correctly on at least a ρ fraction of $\mathcal{F}^{n\times n}$, where $\rho = \rho(n)$ and $\mathcal{F} = \mathrm{GF}(p(n))$, we apply the list decoding result of Sudan, Trevisan, and Vadhan [33, Thm. 29], and obtain an explicit list of $O(1/\rho)$ oracle machines such that *one of these machines* (when given oracle access to the said rare-case solver) *computes $\texttt{CWC}_t^p$ correctly on all n-by-n matrices over $\mathcal{F}$*. Furthermore, by employing a low-degree tester, we can discard machines that compute n^2-variate functions that are not close to a polynomial of degree at most t^2, and using self-correction all the remaining machines can be made to compute low-degree polynomials. Moreover, if we can obtain samples of the form $(R, \texttt{CWC}_t^p(R))$, for random $R \in \mathcal{F}^{n\times n}$, then we can also discard machines that compute functions that are far from $\texttt{CWC}_t^p$, which means that we discard all machines that (after self-correction) do not compute $\texttt{CWC}_t^p$.

The question is how do we obtain such a sample (of solved instances). The answer, inspired by [25], is that we can use the downwards self-reducibility of $\overline{\Pi}_n^p$ in order to obtain such a sample. First, note that $\overline{\Pi}_n^p$ is downwards self-reducible by virtue of Eq. (10); in fact, we use $\overline{\Pi}_n^p$ rather than

[20] In other words, we refer to solving $\overline{\texttt{CWC}}_t^p$ rather than $\texttt{CWC}_t^p$.

$\overline{\Pi}_n^p$ is order to support such a downwards reduction (since the instance $(W, 1^t)$ will be reduced to instances of the form $(W', 1^{t-1})$). Second, note that we can obtain correct answers to the reduced instances by using the foregoing worst-case to rare-case reduction. (We stress that the size of the sample does not grow with the number of queries we need to serve.)

The actual process will go the other way around: For $i = 2, ..., t-1$, we generate a sample of instances of the form $(\cdot, 1^{i+1})$ and solve them by downward reduction to instances of the form $(\cdot, 1^i)$, which in turn are solved (either directly if $i = 2$ or) by using the worst-case to rare-case reduction (which uses a sample for instances of the form $(\cdot, 1^i)$). When we complete the $t - 1^{\text{st}}$ iteration, we have a sample that allows us to solve the original instance (which has the form $(\cdot, 1^t)$) by using the worst-case to rare-case reduction. (Indeed, at each iteration, the worst-case to rare-case reduction generates queries that are forwarded to a rare-case solver.)

The actual proof of Theorem 3.13 proceeds as follows. First, we recall the definition of a sample-aided reduction, which underlies the foregoing discussion. Then, we present a sample-aided reduction from solving $\overline{\Pi}_n^p$ in the worst-case to solving $\overline{\Pi}_n^p$ in the rare-case. Next, we show that $\overline{\Pi}_n^p$ is downwards self-reducible, and finally we combine the two reductions (as outlined above) to derive the claimed result.

A sample-aided reduction. We start by spelling out the notion of a reduction that obtains uniformly distributed "solved instances" of the problem that it tries to solve. This notion, termed a sample-aided reduction, is implicit in [25] and was explicitly presented (in greater generality) in our prior work [21]. Here we specialize the definition of [21] to the case of worst-case to rare-case reductions.

Definition 3.14 (sample-aided worst-case to rare-case reductions): *Let $\ell, s : \mathbb{N} \to \mathbb{N}$, and suppose that M is an oracle machine that, on input $x \in \{0,1\}^n$, obtains as an auxiliary input a sequence of $s = s(n)$ pairs of the form $(r, v) \in \{0,1\}^{n+\ell(n)}$. We say that M is an* sample-aided reduction of solving Π' on the worst-case to solving Π' on a ρ fraction of the instances *if, for every procedure P that answers correctly on at least a ρ fraction of the instances of length n, it holds that*

$$\mathrm{Pr}_{r_1,...,r_s \in \{0,1\}^n} \left[\mathrm{Pr}[\forall x \in \{0,1\}^n \ M^P(x; (r_1, \Pi'(r_1)), ..., (r_s, \Pi'(r_s))) = \Pi'(x)] \geq 2/3 \right] \ > \ 2/3, \quad (15)$$

where the internal probability is taken over the coin tosses of the machine M and the procedure P. The function $s : \mathbb{N} \to \mathbb{N}$ is called the sample complexity *of the reduction.*

Clearly, the error probability of M and of the sample can be decreased by repetitions.

We now turn back to the proof of Theorem 3.13. Noting that the instances of $\overline{\Pi}_n^p$ have varying length, we let $\overline{\Pi}_{n,i}^p$ denote the problem $\overline{\Pi}_n^p$ restricted to instances of length $n^2 \lceil \log_2 p \rceil + i$; that is, the set of instances of $\overline{\Pi}_{n,i}^p$ is restricted to the instances of $\overline{\Pi}_n^p$ that have the form $(W, 1^i)$, where $i \in [t(n)]$ and $W \in \mathrm{GF}(p(n))^{n \times n}$.

Proposition 3.15 (a sample-aided version of Theorem 3.13): *Let t, p and ρ be as in Theorem 3.13. Then, there exists a sample-aided reduction M of sample complexity $O(\log(1/\rho))$ such that, for every $n \in \mathbb{N}$, $p = p(n)$ and $i \in [t(n)]$, machine M reduces solving $\overline{\Pi}_{n,i}^p$ on the worst-case to solving $\overline{\Pi}_{n,i}^p$ on a $\rho(n)$ fraction of the instances, while making* $\mathrm{poly}(t(n)/\rho(n))$ *queries and running in* $\mathrm{poly}(t(n)/\rho(n)) \cdot \widetilde{O}(n^2)$ *time.*

Proof: As stated in the overview, the list decoding result of Sudan, Trevisan, and Vadhan [33, Thm. 29], yields an explicit list of $O(1/\rho(n))$ oracle machines such that one of these machines (when given oracle access to a rare-case solver for $\overline{\Pi}_{n,i}^p$) solves $\overline{\Pi}_{n,i}^p$ in the worst-case. Indeed, this list is constructed by making making $\text{poly}(t(n)/\rho(n))$ queries (to the rare-case solver) and running in $\text{poly}(t(n)/\rho(n)) \cdot \widetilde{O}(n^2)$ time. By employing a low-degree tester, within similar complexity, we can discard machines that compute n^2-variate functions that are not close to a polynomial of degree at most t^2. Furthermore, by employing a self-corrector (and within similar complexity), we can augment all machines so that they compute low degree polynomials. (Actually, the complexities in both augmentations are independent of ρ, and "being close" may mean agreeing on 90% of the instances.) Hence, each of these (augmented) machines either computes $\overline{\Pi}_{n,i}^p$ or computes a function (indeed a different low-degree polynomial) that is far from $\overline{\Pi}_{n,i}^p$. By using the sample of $O(\log(1/\rho))$ solved instances of $\overline{\Pi}_{n,i}^p$, we can identify the machines that compute $\overline{\Pi}_{n,i}^p$, and pick one of them to handle the actual input (given to our reduction). ■

A downwards self-reduction reduction. It is convenient to state the downwards self-reducibility feature of $\overline{\Pi}_n^p$ by using the notation $\overline{\Pi}_{n,i}^p$. (Recall that $\overline{\Pi}_{n,i}^p$ denotes the problem $\overline{\Pi}_n^p$ restricted to instances of length $n^2\lceil\log_2 p\rceil + i$.)

Proposition 3.16 (downwards self-reducibility feature of $\overline{\Pi}_n$): *Let t, p and ρ be as in Theorem 3.13. Then, there exists a $\widetilde{O}(\text{poly}(t(n)) \cdot n^2)$-time reduction of $\overline{\Pi}_{n,i}^p$ to $\overline{\Pi}_{n,i-1}^p$ that works for every $n \in \mathbb{N}$ and $i \in [2, t(n)]$, while making n queries if $i > 2$ and no queries otherwise.*

The time bound presumes a model of computation in which the cost of making the (equal length) queries $q_1, ..., q_m$ is $|q_1| + \sum_{j\in[n-1]} \Delta(q_j, q_{j+1})$, where $\Delta(x, y)$ denotes the Hamming distance between x and y. This model is justified by the actual cost of composing the reduction with a procedure that solves the reduced instances.

Proof: As stated in the overview, the reduction is given by Eq. (10), where in the case of $i = 2$ a direct calculation will do (i.e., on input $(W, 1^2)$, we just need to compute $\sum_{j<k\in[n]} w_{j,j}w_{j,k}w_{k,k}$). For the case of $i > 2$, a straightforward implementation of the reduction yield running time that is related to n^3, since we have to compute the matrices $W^{(q)}$ for $q = 1, ..., n$. Fortunately, these n matrices are closely related, so each can be derived from the previous matrix by making only $2n$ changes, since each $W^{(q)}$ differs from the input matrix W on at most n predetermined entries (i.e., on the entries (j, j) for $j \in [n]$, see Eq. (11)). ■

Combining Propositions 3.15 and 3.16. Towards combining the two reductions, note that each invocation of the sample-aided (worst-case to rare-case) reduction (for $\overline{\Pi}_{n,t(n)}^p$) generates $q = \text{poly}(t(n)/\rho(n))$ queries (to the rare-case solver) and requires a solved sample of size $O(\log(1/\rho))$. Actually, since we generate $t(n) - 2$ of these solved samples, using the sample-aided reduction itself, we have to reduce the error probabilities of the reduction so that, with probability at least $1 - (1/6t(n))$ (over the choice of the sample), each query is answered correctly with probability $1 - (1/6sn \cdot t(n))$, where s is the size of the sample; setting $s = \widetilde{O}(\log(1/\rho) \cdot \log(t(n)))$ will do. To summarize, on input $(W, 1^t)$, where $W \in \text{GF}(p)^{n\times n}$ and $t \in [t(n)]$, the worst-case to rare-case reduction asserted in Theorem 3.13 proceeds as follows.

1. First, the reduction generates a sample of s solved instances of $\overline{\Pi}_{n,2}^p$ by generating s instances and solving them directly (see the case of $i = 2$ in Proposition 3.16).

2. Next, for $i = 3, ..., t$, the reduction generates a sample of s solved instances of $\overline{\Pi}_{n,i}^p$ by generating s random instances and solving each of them by using the downwards reduction of $\overline{\Pi}_{n,i}^p$ to $\overline{\Pi}_{n,i-1}^p$ (see the main case of Proposition 3.16), and solving the resulting $s \cdot n$ instances by using the sample-aided reduction of Proposition 3.15 (while using the solved sample generated for $\overline{\Pi}_{n,i-1}^p$).

3. Finally, having a solved sample for $\overline{\Pi}_{n,t}^p$, the reduction solves the (input) instance $(W, 1^t)$ by invoking the sample-aided reduction of Proposition 3.15 (while using the said solved sample).

In all cases, the queries generated by the worst-case to rare-case reduction (of Proposition 3.15) are forwarded to the rare-case solver for $\overline{\Pi}_{n,i}^p$ for $i = 3, ..., t$. Theorem 3.13 follows. ■

3.4.2 The reduction of $\overline{\Pi}_n$

We now consider the problem $\overline{\Pi}_n$ in which the instances have the form $(p, W, 1^i)$ such that p is an $\ell(n)$-bit long prime, $W \in \mathrm{GF}(p)^{n\times n}$, and $i \in [t(n)]$, and the problem is to compute $\mathtt{CWC}_i^p(W)$.

Showing a worst-case to average-case reduction for $\overline{\Pi}_n$ is more complex than doing so for $\overline{\Pi}_n^p$, because when given the input $(p, W, 1^i)$ it may be the case that the rare-case solver just fails on all inputs of the form $(p, \cdot, 1^i)$. As in Section 3.3.2, we shall obtain the value of $\mathtt{CWC}_i^p(W)$ (or rather $\mathtt{CWC}_i(W)$) from the values of $\mathtt{CWC}_i^{p'}(W)$ for other primes $p' \in [2^{\ell(n)-1}, 2^{\ell(n)}]$.

Theorem 3.17 (worst-case to average-case reduction for $\overline{\Pi}_n$):[21] *Let $\ell : \mathbb{N} \to \mathbb{N}$ be such that $\ell(n) \in [3\log t(n) + \log\log n + \omega(1), O(t(n)\log n)]$. Fixing efficiently computable functions $t : \mathbb{N} \to \mathbb{N}$ and $\rho : \mathbb{N} \to (0, 1]$ such that t and $1/\rho$ are monotonically non-decreasing and $\rho(n) \geq 2^{\ell(n)/3}$, there exists a worst-case to rare-case reduction of $\overline{\Pi}_n$ to itself that makes $\mathrm{poly}(t(n)/\rho(n)) \cdot \widetilde{O}(n)$ queries, runs in $\mathrm{poly}(t(n)/\rho(n)) \cdot \widetilde{O}(n^2)$ time, and outputs the correct value with probability $2/3$ provided that the success rate of the rare-case solver is at least $\rho(n)$.*

Planning to adapt the proof of Theorem 3.13, analogously to the adaptation presented in Section 3.3.2, we face a problem: A sample of (uniformly and independently distributed) instances of $\overline{\Pi}_n$ is unlikely to contain an instance that refers to the same prime as the worst-case instance given to us, let alone contain several instances that refer to the same prime. So such a sample is not going to help us to identify an oracle machine that solves the problem $\overline{\Pi}_{n,i}^p$, where p is the prime in the worst-case instance (which has the form $(p, \cdot, 1^i)$).

A generalized notion of sample-aided reductions. To overcome this problem, we generalize the notion of sample-aided reductions so that it is applicable to any sample of solved instances, where the instances in the sample need not be independent of one another (nor be uniformly distributed in the relevant domain). For sake of convenience, we reproduce Definition 3.14, while generalizing it in a single point: The sequence of samples in Eq. (16) is selected from an arbitrary distribution over $(\{0,1\}^n)^s$, denoted $\mathcal{D}_n$, rather than from the uniform distribution over $(\{0,1\}^n)^s$.

Definition 3.18 (Definition 3.14, generalized): *Let $s : \mathbb{N} \to \mathbb{N}$, and suppose that M is an oracle machine that, on input $x \in \{0,1\}^n$, obtains as an auxiliary input a sequence of $s = s(n)$ pairs of*

[21] Recall that $\overline{\Pi}_n$ is defined in terms of of the functions $t, \ell : \mathbb{N} \to \mathbb{N}$. It refers to instances of the form $(p, W, 1^i)$ such that p is an $\ell(n)$-bit long prime, $W \in \mathrm{GF}(p)^{n\times n}$, and $i \in [t(n)]$, and calls for computing $\mathtt{CWC}_i^p(W)$.

the form $(r,v) \in \{0,1\}^{n+\ell(n)}$*. Let* $\mathcal{D}_n$ *be an arbitrary distribution over* $(\{0,1\}^n)^s$*, and* $\mathcal{D} = (\mathcal{D}_n)$*. We say that* M *is an* $\mathcal{D}$-sample-aided reduction of solving Π' on the worst-case to solving Π' on a ρ fraction of the instances *if, for every procedure* P *that answers correctly on at least a* ρ *fraction of the instances of length* n*, it holds that*

$$\Pr_{(r_1,\ldots,r_s)\sim\mathcal{D}_n}\left[\Pr[\forall x \in \{0,1\}^n \; M^P(x;(r_1,\Pi'(r_1)),\ldots,(r_s,\Pi'(r_s))) = \Pi'(x)] \geq 2/3\right] \; > \; 2/3, \quad (16)$$

where the internal probability is taken over the coin tosses of the machine M *and the procedure* P*. The function* $s : \mathbb{N} \to \mathbb{N}$ *is called the* sample complexity *of the reduction.*

Definition 3.14 is obtained as a special case by letting $\mathcal{D}_n$ be the uniform distribution over $(\{0,1\}^n)^s$.

Proposition 3.19 (a sample-aided version of Theorem 3.17): *Let* t, ℓ *and* ρ *be as in Theorem 3.13. Then, there exist distributions* $\mathcal{D} = (\mathcal{D}_{n,i})$ *such that* $\mathcal{D}_{n,i}$ *is a distribution of sequences of instances for* $\overline{\Pi}_{n,i}$*, and a* $\mathcal{D}$*-sample-aided reduction* M *of sample complexity* $\widetilde{O}(1/\rho)$ *such that the following holds: For every* $n \in \mathbb{N}$ *and* $i \in [t(n)]$*, machine* M *reduces solving* $\overline{\Pi}_{n,i}$ *on the worst-case to solving* $\overline{\Pi}_{n,i}$ *on a* $\rho(n)$ *fraction of the instances, while making* $\mathrm{poly}(t(n)/\rho(n)) \cdot (\log n)/\ell(n)$ *queries and running in* $\mathrm{poly}(t(n)/\rho(n)) \cdot \widetilde{O}(n^2)$ *time. Furthermore, there exists a randomized algorithm that on input* (n,i) *outputs a sample of* $\mathcal{D}_{n,i}$ *in time that is almost linear in the sample's length.*

Proof: We first observe that for at least an $\rho/2$ fraction of the primes p in $I_n \stackrel{\text{def}}{=} [2^{\ell(n)-1}, 2^{\ell(n)}]$, the rare-case solver is correct on at least a $\rho/2$ fraction of the instances in $\overline{\Pi}^p_{n,i}$. We call such a prime good. For each good prime p, we can proceed as in the proof of Proposition 3.15, provided that we are given a sample of (independently distributed) solved instances of $\overline{\Pi}^p_{n,i}$. Again, as in Section 3.3.2, it may be that the prime that is part of our worst-case instance is not good, let alone that the sample $\mathcal{D}_{n,i}$ should be independent of the worst-case instance. So again, we employ Chinese Remaindering, but this time without error (or rather with erasure faults only, which refer to the primes for which no oracle machine works). Specifically, on input $(p, W, 1^i)$ and a sample of $\mathcal{D}_{n,i}$ (to be determined later), we proceed as follows.

1. Suppose that the sample of $\mathcal{D}_{n,i}$ contains m different primes (in I_n) and each prime is coupled with m' different matrices.
2. For each selected prime, denoted p_j, invoke the sample-aided worst-case to rare-case procedure for $\overline{\Pi}^{p_j}_{n,i}$ on the instance $(p_j, W, 1^i)$, and let the result be denoted v_j if an output was produced. The aforementioned reduction is the one presented in the proof of Proposition 3.15, and in the current context (where p_j may not be good) this reduction may generate no output (which happens if all oracle machines were found to fail).
3. Denoting the set of j's for which an output was generated by J, apply Chinese Remaindering (without errors) on the pairs (p_j, v_j) for $j \in J$, and output the result reduced modulo p.

At this point we can specify the distribution $\mathcal{D}_{n,i}$, by presenting an algorithm that generates it. On input (n,i), this algorithm first selects $m = O(t(n)^2/\rho) \cdot (\log n)/\ell(n)$ uniformly and independent distributed primes in I_n. For each selected prime p_j, it selects $m' = O(\log(1/\rho))$ uniformly and independent distributed n-by-n matrices over $\mathrm{GF}(p_j)$. The output is the list of $m \cdot m'$ triples $(p_j, M', 1^i)$ such that p_j was selected and $M' \in \mathrm{GF}(p_j)^{n \times n}$ was selected for it.

The proposition follows by noting that, with high probability, the sample contains at least $O(t(n)^3 \log n)/\ell(n)$ good primes, which implies that the Chinese Remaindering is applied with sufficiently many values (i.e., whose product exceeds $\binom{n}{t(n)} \cdot (2^{\ell(n)})^{t(n)^2}$).[22] ■

Completing the proof of Theorem 3.17. We first note that the downwards self-reduction presented in Proposition 3.16 applies to $\overline{\Pi}_n$; indeed, for every n, i and $p \in I_n$, this reduction maps instances of $\overline{\Pi}^p_{n,i}$ to instances of $\overline{\Pi}^p_{n,i-1}$. The sample-aided reduction of Proposition 3.19 and the aforementioned downwards self-reduction can be combined just as in Section 3.4.1. Specifically, on input $(p, W, 1^t)$ (where $p \in I_n$, $W \in \mathrm{GF}(p)^{n\times n}$ and $t \in [t(n)]$), the worst-case to rare-case reduction asserted in Theorem 3.17 proceeds as follows.

1. First, the reduction generates a sample of solved instances of $\overline{\Pi}_{n,2}$ by taking a sample of $\mathcal{D}_{n,2}$ and (directly) solving all instances that appear in it.
2. Next, for $i = 3, ..., t$, the reduction generates a sample of solved instances of $\overline{\Pi}_{n,i}$ by taking a sample of $\mathcal{D}_{n,i}$ and solving each instance that appears in it by using the downwards reduction of $\overline{\Pi}_{n,i}$ to $\overline{\Pi}_{n,i-1}$ (see the main case of Proposition 3.16), and solving the resulting instances by using the sample-aided reduction of Proposition 3.19 (while using the solved sample generated for $\overline{\Pi}_{n,i-1}$).
3. Finally, having a solved sample for $\overline{\Pi}_{n,t}$, the reduction solves the (input) instance $(p, W, 1^t)$ by invoking the sample-aided reduction of Proposition 3.19 (while using the said solved sample).

In all cases, the queries generated by the worst-case to rare-case reduction (of Proposition 3.19) are forwarded to the rare-case solver for $\overline{\Pi}_{n,i}$ for $i = 3, ..., t$. Theorem 3.17 follows. ■

Corollary 3.20 (worst-case to rare-case reduction for counting cliques): *Let t be a constant and $b(n) = \Theta(t^3 \log n)$. For every $i \in [t]$, let $\mathcal{G}_n^{(i)}$ be a distribution on $(\widetilde{O}(n) - (t-i))$-vertex graphs obtained by selecting a prime $p \in [b(n), 2b(n)]$ and $W \in \mathrm{GF}(p)^{n\times n}$, and outputting $T'(i, W)$ where T' is the mapping presented in Remark 3.7. Then, for every non-increasing $\rho : \mathbb{N} \to (0, 1]$ such that $\rho(n) \geq 1/\mathrm{poly}(\log n)$, there exists a worst-case to rare-case reduction of counting t-cliques in n-vertex graphs to counting t-cliques in graphs generated according to the $\mathcal{G}_n^{(i)}$'s such that the reduction runs in $\widetilde{O}(n^2)$ time, makes $\widetilde{O}(n)$ queries, and outputs the correct value with probability $2/3$ provided that the success rate of the rare-case solver is at least $\rho(n)$.*

Note that the different $\mathcal{G}_n^{(i)}$'s are distributed over graphs with different number of vertices; that is, the support of $\mathcal{G}_n^{(i)}$ contains $n'' - (t - i)$ vertices, where $n'' = \widetilde{O}(n)$.

Proof: Given an n-vertex graph G, using Proposition 3.5, we reduce counting the number of t-cliques in G to making $O(\log n)$ queries to oracles of the form $\mathtt{CWC}^p_t$ such that p is a prime in $[b(n), 2b(n)]$. Next, setting $\ell(n) = \lceil \log b(n) \rceil$ (and using Theorem 3.17), we reduce answering each of these queries to solving the problem $\overline{\Pi}_n$ on at least $\rho(n)$ of the instances. (We do so after reducing the error probility of the reduction to $o(1/\log n)$.) Lastly, using the mapping T' of Remark 3.7, we map the $\widetilde{O}(n)$ random queries made by the (worst-case to rare-case) reduction to queries about

[22] This is the case since the product is greater than $(2^{\ell(n)})^{O(t(n)^3 \log n)/\ell(n)}$, which in turn is greater than $2^{t(n)\log n} \cdot (2^{\ell(n)})^{t(n)^2}$.

the number of cliques in $\widetilde{O}(n)$-vertex graphs; that is, the query $(p, W, 1^i)$ is mapped to $T'(i, W)$, and the answer is divided by $t!$ and reduced modulo p. (We stress that a procedure that counts t-cliques in the $\mathcal{G}_n^{(i)}$'s correctly with probability at least $\rho(n)$, yields a procedure that answers $\overline{\Pi}_n$ correctly with probability at least $\rho(n)$.) ■

3.4.3 Reducing several lengths to one length

Corollary 3.20 falls short from establishing Theorem 1.2 in two aspects: Firstly, it reduces worst-case n-vertex (graph) instances to rare-case instances that are $n^{(i)}$-vertex graphs, for t different values of $n^{(i)} = \widetilde{O}(n)$, and secondly these lengths are all larger than n. We already saw how to deal with the second problem in Section 3.3.3, so here we focus on handling the first problem. We shall show how to reduce counting t-cliques in the rare-case on $O(1)$ different instance lengths to counting t-cliques in the rare-case for one instance length. More generally, we sho how to reduce counting t-cliques in the rare-case on $O(1)$ different instance distributions to counting t-cliques in the rare-case for one instance distribution.

Theorem 3.21 (reducing several rare-case t-clique counting tasks to one): *For constants $t, m \in \mathbb{N}$, let $\mathcal{G}^{(1)}, ...\mathcal{G}^{(m)}$ be distributions on graphs such that $\mathcal{G}^{(i)}$ is distributed on $n^{(i)}$-vertex graphs, and let $n = \max_{i\in[m]}\{n^{(i)}\}$. Suppose that each $\mathcal{G}^{(i)}$ is sampleable in time $T(n)$. Then, there exists a distribution $\mathcal{G}''$ on $\widetilde{O}(n)$-vertex graphs that is sampleable in time $O(T(n))$ such that, for each $i \in [m]$, counting t-cliques in the rare-case on $\mathcal{G}^{(i)}$ is $\widetilde{O}(T(n)+n^2)$-time reducible to counting t-cliques in the rare-case on $\mathcal{G}''$. Specifically, for every $i \in [m]$ and every ρ such that $\mathcal{G}^{(i)}$ has min-entropy $\omega(\rho^{-3})$, there exists a* $\mathrm{poly}(1/\rho)\cdot(T(n)+\widetilde{O}(n^2))$*-time randomized algorithm that, with very high probability, outputs a deterministic* $\mathrm{poly}(1/\rho)\cdot(T(n)+\widetilde{O}(n^2))$*-time reduction of counting t-cliques on a $\Omega(\rho^3)$ measure of the instances in $\mathcal{G}^{(i)}$ to counting t-cliques on a ρ measure of the $\widetilde{O}(n/\rho)$-vertex instances in $\mathcal{G}''$.*

Proof: We proceed in three steps. First, we consider a distribution on n'-vertex graphs produced as follows: For every $i \in [m]$, we take a sample $G^{(i)}$ of $\mathcal{G}^{(i)}$, and consider a graph G' that consists of isolated copies of blow-ups of the graphs $G^{(1)}, ..., G^{(m)}$. Specifically, each vertrex in $G^{(i)}$ is replaced by an independent set of size n^{i-1} and edges (of $G^{(i)}$) are replaced by complete bipartite graphs between the corresponding sets. Hence, the number of t-cliques in G' equals $N' = \sum_{i\in[m]} n^{(i-1)\cdot t}\cdot N_i$, where N_i is the number of t-cliques in $G^{(i)}$. Since each N_i is smaller than n^t, it is easy to extract the N_i's from N'.

The problem with the foregoing construction is that the graph G' is way too big (i.e., $n' = \sum_{i\in[m]} n^{i-1}\cdot n^{(i)}$). Instead (and this is our second step), we set $b = O(\rho^{-1}\log(n^{mt}))$, select at random a prime $p \in [b, 2b]$, and let G'' be similar to G' except that the graph $G^{(i)}$ is blown-up by a factor of $1 + (n^{i-1} - 1 \bmod p)$ rather than by a factor of n^{i-1}. Hence, the number of t-cliques in G'' reduced modulo p equals $\sum_{i\in[m]} n^{(i-1)\cdot t}\cdot N_i \bmod p$, whereas the number of vertices in G'' is $n'' = O(bn) = \widetilde{O}(n/\rho)$. (Indeed $\mathcal{G}''$ is defined as the result of the foregoing process that generates G''.)

We call a seqeuence of graphs $(G^{(1)}, ..., G^{(m)})$ good if, conditioned on its being genertated by $\mathcal{G}^{(1)}\times\cdots\times\mathcal{G}^{(m)}$, with probability at least $\rho/2$ (on the choice of p), the rare-case solver outputs the number of t-cliques in the resulting graph G''. Note that the probability that a sequence is good is at least $\rho/2$, since the rare-case solver is correct on at least a ρ measure of the graphs $G'' \leftarrow \mathcal{G}''$. On the other hand, if the generated sequence of graphs is good, then applying Chinese Remaindering with large error rate [19, Sec. 4] to the values obtained for the corresponding graphs G'' (obtained for all primes in $[b, 2b]$), yields a sequence of $O(1/\rho)$ numbers that includes the currect number of t-cliques in the corresponding graph G' (i.e., the large graph we cannot afford to query about). We shall select a number on the list at random, and output it.

Hence, a random reduction from counting t-cliques under the distribution $\mathcal{G}^{(i)}$ works as follows. On input $G^{(i)} \leftarrow \mathcal{G}^{(i)}$, it generates at random $G^{(j)} \leftarrow \mathcal{G}^{(j)}$ for each $j \in [m] \setminus \{i\}$, and generates the corresponding graph G'', for each prime $p \in [b, 2b]$. It then queries for the number of t-cliques in each of these $O(b/\log b)$ graphs, and applies Chinese Remaindering with error rate $1 - \rho/2$, and selects uniformly an element in the resulting list (of length $O(1/\rho)$). (The hope is that this number equals N', the number of t-cliques in the corresponding graph G'.) Finally, the reduction extracts from this number, a guess for the number of t-cliques in $G^{(i)}$, and outputs it.

Observe that, with probability at least a $\rho/4$, over the choice of $G^{(i)} \leftarrow \mathcal{G}^{(i)}$, it holds that with probability $\rho/4$ the random augmentation of $G^{(i)}$ into a sequence of m graphs is good. Hence, for at least a $\rho/4$ measure of $\mathcal{G}^{(i)}$, the randomized reduction answers correctly with probability at least $\Omega(\rho^2)$.

Finally (and this is our last step), we consider a randomized algorithm that selects a random hashing function mapping n''-vertex graphs to strings of length that equals the number of coins that is used by the foregoing reduction. Using pairwise independent hashing, it follows that, with very high probability, the resulting deterministic algorithm answers correctly on a $\Omega(\rho^3)$ measure of $\mathcal{G}^{(i)}$. ■

Digest regarding the third step in the foregoing proof. The starting point of this step is a randomized process (i.e., a randomized reduction coupled with an oracle) that successed with probability $\Omega(\rho^2)$ on a $\Omega(\rho)$ measure of the instances. In the current context (and surely generically), we may not be able to recoginize the correct answer, and so it is unclear how to increase the success probability on the good instances (which constitute a $\Omega(\rho)$ measure of all instances). On the other hand, selecting one random sequence of coin outcomes for all instances may be good for $\Omega(\rho)$ measure of the instances only with probability $\Omega(\rho^2)$. Selecting a random sequence for each instance yields good choices for $\Omega(\rho^2)$ measure of the good instances, and selecting a random pairwise independent hashing has the same effect.

Proof of Theorem 1.2: Combining Corollary 3.20 with Theorem 3.21 and Proposition 3.12 yields Theorem 1.2. Again (as in Section 3.3.3), we start by setting $k(n) = n/p(\log n)$, for a suitable polynomial p, and applying Proposition 3.12. Next, we apply Corollary 3.20 with n replaced by $k(n)$, and lastly we apply Theorem 3.21. Note that the polynomial p is chosen such that $\widetilde{O}(k(n)) = \widetilde{O}(n)/p(\log n) \leq n$, where the polylogarithmic function implicit in the $\widetilde{O}$-notation is the product of the overheads incurred by Corollary 3.20 and Theorem 3.21. Likewise, the function ρ used in Theorem 3.21 should be set to the success rate claimed by Theorem 1.2, and Corollary 3.20 should be proved for success rate of $\Omega(\rho^3)$. Under these choices, $\mathcal{G}''$ satisfies the claim of Theorem 1.2. ■

3.5 On counting simple cycles in simple graphs

In Section 3.2 we reduced the problem of counting weighted t-cliques in graphs with small edge and vertex weights to the problem of counting t-cliques in unweightred graphs. Here, we first present a different reduction for the case of $t = 3$, and then extend it to the case of t-cycles for any odd $t \geq 3$.

The case of triangles. Viewing the (symmetric) matrix pf weights $W \in \mathrm{GF}(p)^{n\times n}$ as an n-vertex graph with vertrex and edge weights, we first get rid of the vertex weights. This is done exactly as in Proposition 2.6 (see also Section 3.2). Hence, we derive a graph G' with $n' \leq n \cdot p$ vertices and edge weights (denoted $w_{i,j}$'s) in $[p]$.

Next, we replace each vertex in G' with an independent set of size p, and place a $w_{i,j}$-regular bipartite graph between the i^{th} and j^{th} sets, where $w_{i,j}$ is the weight of the edge $\{i, j\}$ in G'. Specifically, identifying each independent set with $\mathrm{GF}(p)$, and assuming that $i < j$, for every $k \in [w_{i,j}]$, we connect vertex $x \in \mathrm{GF}(p)$ of the i^{th} independent set to vertex $2x + k \bmod p$ of the j^{th} independent set. Indeed, this defines a $w_{i,j}$-regular bipartite graph between these two independent sets. Denoting the resulting graph by G'', we show that the number of triangles in the subgraph of G'' induced by any three sets $i_1, i_2, i_3 \in [n']$ such that $i_1 < i_2 < i_3$ equals $w_{i_1,i_2} w_{i_2,i_3} w_{i_3,i_1}$.

For every $k_{1,2} \in [w_{i_1,i_2}]$, $k_{2,3} \in [w_{i_2,i_3}]$ and $k_{3,1} \in [w_{i_3,i_1}]$, consider the number of triangles that use the corresponding matching among these sets (i.e., the i_1^{st}, i_2^{nd}, and i_s^{rd} sets). Such a generic triangle starts at vertex x in the i_1^{th} set, and goes through vertices $y = 2x + k_{1,2}$ and $z = 2y + k_{2,3}$ of the i_2^{th} and i_3^{th} sets, provided that $z = 2x + k_{1,3}$ holds. That is, $2 \cdot (2x + k_{1,2}) + k_{2,3} = 2x + k_{1,3}$ must hold, which uniquely determines x.

On counting simple odd cycles in simple graphs. The argument extends to counting the number of t-cycles, for any *odd* t. To see this, let $\pi_k : \mathrm{GF}(p) \to \mathrm{GF}(p)$ denote the k^{th} mapping used in the regular bipartite graphs (i.e., $\pi_k(x) = 2x + k \bmod p$), and consider a t-cycle that goes through the independent sets $i_1, ..., i_t$ (and back to i_1). Note that in each step, we use either one of the foregoing mappings or its inverse, where the choice is determined according to the relative order of the vertices; that is, in the j^{th} step we use the forward mapping if and only if $i_j < i_{j+1}$. Hence, for every $k_{1,2} \in [w_{i_1,i_2}], ..., k_{t-1,t} \in [w_{i_{t-1},i_t}]$ and $k_{t,1} \in [w_{i_t,i_1}]$, the t-cycle starts at a generic vertex x in the i_1^{st} set, moves to vertex $\pi_{k_{1,2}}^{\sigma_{1,2}}(x))$ of the i_2^{nd} set, and so on, such that

$$\pi_{k_{t,1}}^{\sigma_{t,1}}(\pi_{k_{t-1,t}}^{\sigma_{t-1,t}}(\cdots(\pi_{k_{1,2}}^{\sigma_{1,2}}(x))\cdots)) = x, \tag{17}$$

where the $\sigma_{j,j+1}$'s are determined as above (i.e., $\sigma_{j,j+1} = 1$ if $i_j < i_{j+1}$ and $\sigma_{j,j+1} = -1$ otherwise). Note that Eq. (17) simplifies to $2^{\sigma_{t,1}} \cdot 2^{\sigma_{t-1,t}} \cdots 2^{\sigma_{1,2}} \cdot x + b = x$, where b is determined by the $k_{j,j+1}$'s and the $\sigma_{j,j+1}$'s. Finally, since the multiplicative order of 2 mod p is larger than t (and t is odd), the foregoing equation has a single solution.[23]

Lastly, we observe that the proof of Theorem 3.10 can be adapted to the case of computing the weight of simple t-cycles in graphs with edge weights ($w_{j,k}$'s). Denoting the set of simple t-cycles

[23] The solution is $x = b/(2^{\sigma_{t,1}+\sigma_{t-1,t}+\cdots+\sigma_{1,2}} - 1)$. Here we used the fact that $2^t < p$, which holds for constant t. We comment that using $t = t(n)$ such that $2^t > p$ (but $p \gg t$) may require replacing 2 by an element of order at least $t + 1$ in the multiplicative group (mod p).

(over $[n]$) by C_t, the key point is replacing $\sum_{S\in\binom{[n]}{t}} \prod_{j\leq k:j,k\in S} w_{j,k}$ by $\sum_{(i_1,...,i_t)\in C_t} \prod_{j\in[t]} w_{i_j,i_{j+1}}$, where i_{t+1} is viewed as i_1. (Indeed, we can set the weights of all self-loops to 1.)[24] Hence, we get

Theorem 3.22 (worst-case to average-case reduction for counting cycles in graphs): *Let $\mathcal{G}_n$ be the distribution on $\widetilde{O}(n)$-vertex graphs that is outlined above, and η be any constant smaller than 1/4. Then, for every odd constant $t \geq 3$, there exists a worst-case to average-case reduction of counting t-cycles in n-vertex graphs to counting t-cycles in graphs generated according to $\mathcal{G}_n$ such that the reduction runs in $\widetilde{O}(n^2)$ time, makes $\widetilde{O}(\log n)$ queries, and outputs the correct value with probability 2/3 provided that the error rate* (of the average-case solver) *is at most η.*

Note that getting rid of edge weights in $[p]$ has a cost of blowing-up the number of vertices by a factor of p, whereas the blow-up in Section 3.2 is $t \cdot p^{\binom{t}{2}}$, where $p = O(t^3 \log n)$ (as per the proof of Theorem 3.10). This blow-up is hidden in the $\widetilde{O}$-notations, which means that Theorem 3.22 holds also for varying $t = t(n)$ provided $t \leq \text{poly}(\log n)$. Furthermore, we acan get appealing results also in case of larger $t(n)$ (e.g., $t(n) = n^{0.1}$).

We also mention the counting t-cycles can be reduced to counting t-cliques (in simple graphs) as follows. Assume that $c = (v_1, ..., v_t)$ is counted as different from its t cyclic shifts, and its reverse (i.e., $(v_t, ..., v_1)$). Call these $2t$ cycles the translations of c. Assuming that t is odd, we first reduce counting t-cycles in $G = ([n], E)$ to counting t-cycles in G' defined as t double-covers of $G = ([n], E)$ that are joined in a cycle; that is, the vertex set of G' is $\mathbb{Z}_t \times [n]$, and the edge set is $\cup_{i\in[t]}\{\{(i, u), (i + 1 \bmod t, v)\} : \{u, v\} \in E\}$. Now, the $2t$ translations of each t-cycle in G contributes $2t$ translations of a cycle to G'; that is, $(v_1, ..., v_t)$ gives rise to $((1, v_1), ..., (t, v_t))$ and its translations, whereas each t-cycle in G' must have the form $((1, u_1), ..., (t, u_t))$ (such that $\{u_i, u_{i+1}\}$ is in E). Next, we reduce counting t-cycles in G' to counting t-cliques in G'' that is obtained from G' by adding complete bipartite graphs between each pair of non-adjecent independent sets of G', where the i^{th} independent set of G' is $\{(i, v) : v \in [n]\}$. Note that each t-cycle of G' contributes a t-clique in G'', whereas each t-clique of G'' must have the form $((1, u_1), ..., (t, u_t))$ and so yields a t-cycle in G'.

References

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the Current Clique Algorithms are Optimal, So is Valiant's Parser. In *46th IEEE Symposium on Foundations of Computer Science*, pages 98–117, 2015.

[2] Amir Abboud, Kevin Lewi, and Ryan Williams. Losing Weight by Gaining Edges. In *22nd ESA*, pages 1–12, 2014

[3] Miklos Ajtai. Σ_1^1-formulae on finite structures. *Ann. Pure Appl. Logic*, Vol. 24 (1), pages 1–48, 1983.

[4] Laszlo Babai. Random oracles separate PSPACE from the Polynomial-Time Hierarchy. *IPL*, Vol. 26, pages 51–53, 1987.

[24] We could have done so also in Section 3.3 (but chose to remain consistent with Section 3.1).

[5] Laszlo Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has Subexponential Time Simulations unless EXPTIME has Publishable Proofs. *Complexity Theory*, Vol. 3, pages 307–318, 1993.

[6] Marshall Ball, Alon Rosen, Manuel Sabin and Prashant Nalini Vasudevan. Average-case fine-grained hardness. In *49th ACM Symposium on the Theory of Computing*, pages 483–496, 2017.

[7] Marshall Ball, Alon Rosen, Manuel Sabin and Prashant Nalini Vasudevan. Proofs of Useful Work. IACR Cryptology ePrint Archive, Report 2017/203, 2017.

[8] Andreas Björklund and Petteri Kaski. How Proofs are Prepared at Camelot. In *35th ACM Symposium on Principles of Distributed Computing*, pages 391–400, 2016.

[9] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-Testing/Correcting with Applications to Numerical Problems. *Journal of Computer and System Science*, Vol. 47 (3), pages 549–595, 1993. Preliminary version in *22nd STOC*, 1990.

[10] Andrej Bogdanov and Luca Trevisan. On Worst-Case to Average-Case Reductions for NP Problems. *SIAM Journal on Computing*, Vol. 36 (4), pages 1119–1159, 2006.

[11] Jin-yi Cai, Aduri Pavan, and D. Sivakumar. On the Hardness of Permanent. In *16th STACS*, pages 90–99, 1999.

[12] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science A*, Vol. 141 (12), pages 109–131, 1995.

[13] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer-Verlag Monographs in Computer Science, 1999.

[14] Cynthia Dwork and Moni Naor. Pricing via Processing or Combatting Junk Mail. In the proceedings of *CRYPTO*, pages 139–147, 1992.

[15] Joan Feigenbaum and Lance Fortnow. Random-Self-Reducibility of Complete Sets. *SIAM Journal on Computing*, Vol. 22 (5), pages 994–1005, 1993.

[16] Jorg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, Vol. 33 (4), pages 892–922, 2004.

[17] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[18] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, Vol. 38, No. 3, pages 691–729, 1991. Preliminary version in *27th FOCS*, 1986.

[19] Oded Goldreich, Dana Ron, and Madhu Sudan. Chinese Remaindering with Errors. *IEEE Trans. Information Theory*, Vol. 46 (4), pages 1330–1338, 2000. Preliminary version in *31st STOC*, 1999.

[20] Oded Goldreich and Guy N. Rothblum. Simple Doubly-Efficient Interactive Proof Systems for Locally-Characterizable Sets. *ECCC*, TR17-018, February 2017.

[21] Oded Goldreich and Guy N. Rothblum. Worst-case to Average-case reductions for subclasses of P. *ECCC* TR17-130, 2017.

[22] Oded Goldreich and Avi Wigderson. Derandomization that is rarely wrong from short advice that is typically good. *ECCC*, TR02-039, 2002.

[23] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating Computation: Interactive Proofs for Muggles. *Journal of the ACM*, Vol. 62(4), Art. 27:1-27:64, 2015. Extended abstract in *40th STOC*, pages 113–122, 2008.

[24] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, Vol. 18, pages 186–208, 1989. Preliminary version in *17th STOC*, 1985. Earlier versions date to 1982.

[25] Russell Impagliazzo and Avi Wigderson. Randomness vs Time: Derandomization under a Uniform Assumption. *Journal of Computer and System Science*, Vol. 63 (4), pages 672–688, 2001.

[26] Richard J. Lipton. New directions in testing. *Distributed Computing and Cryptography*, J. Feigenbaum and M. Merritt (ed.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematics Society, Vol. 2, pages 191–202, 1991.

[27] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, Vol. 39, No. 4, pages 859–868, 1992. Extended abstract in *31st FOCS*, 1990.

[28] Or Meir. IP = PSPACE Using Error-Correcting Codes. *SIAM Journal on Computing*, Vol. 42 (1), pages 380–403, 2013.

[29] Jaroslav Nesetril and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, Vol. 26, No. 2, pages 415–419, 1985.

[30] Omer Reingold, Guy N. Rothblum, Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *48th ACM Symposium on the Theory of Computing*, pages 49–62, 2016.

[31] Adi Shamir. IP = PSPACE. *Journal of the ACM*, Vol. 39, No. 4, pages 869–877, 1992. Preliminary version in *31st FOCS*, 1990.

[32] Justin Thaler. Semi-Streaming Algorithms for Annotated Graph Streams. In *43rd International Colloquium on Automata, Languages, and Programming*, pages 59:1–59:14, 2016.

[33] Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom Generators without the XOR Lemma. *Journal of Computer and System Science*, Vol. 62 (2), pages 236–266, 2001.

[34] Virginia Vassilevska Williams. Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis. In *10th International Symposium on Parameterized and Exact Computation*, pages 17–29, 2015.

[35] Ryan Williams. Strong ETH Breaks With Merlin and Arthur: Short Non-Interactive Proofs of Batch Evaluation. In *31st Conference on Computational Complexity*, pages 2:1–2:17, 2016.

Appendix: Computing $\texttt{CWC}_t^p$

We show that the ideas underlying the best known algorithm for deciding the existence of t-cliques in graphs [16], extend to compute $\texttt{CWC}_p^t$. Hence, $\texttt{CWC}_t^p$ can be computed using $O(n^{\omega_{\texttt{mm}}\lceil t/3\rceil})$ field operations, where $\omega_{\texttt{mm}}$ is the matrix multiplication exponent.

Let $\mathcal{F} = \mathrm{GF}(p)$. For any $t \in \mathbb{N}$ and an n-by-n matrix $W \in \mathcal{F}^{n\times n}$, we first reduce computing $\texttt{CWC}_{3t}^p(W)$, to computing $\texttt{CWC}_3(W')$, where W' has dimention $n' = \binom{n}{t}$. Specifically, the rows (resp., columns) of $W' = (w'_{A,B})$ correspond to t-subsets of rows (resp., columns) of $W = (w_{i,j})$, and

$$w'_{A,B} = \begin{cases} 0 & \text{if } |A \cap B| \notin \{0,t\} \\ \prod_{i\le j\in A} w_{i,j} & \text{if } A = B \\ \prod_{i\in A, j\in B} w_{i,j} & \text{otherwise (i.e., } A\cap B = \emptyset) \end{cases}$$

Note that w' can be constructed in time $O(n^{2t})$, and that $\binom{3t}{t,t,t} \cdot \texttt{CWC}_{3t}(W) = 6 \cdot \texttt{CWC}_3(W')$. (The cases of $\texttt{CWC}_{3t-1}$ and $\texttt{CWC}_{3t-2}$ are easily reduced to $\texttt{CWC}_{3t}$.)[25]

Turning to the computation of $\texttt{CWC}_3(W)$, where $W = (w_{i,j})$ is an n-by-n matrix, for every $i < j \in [n]$, the sum of the weights of the 3-subsets that contain $\{i,j\}$ equals

$$w_{i,j} \cdot w_{i,i} \cdot w_{j,j} \cdot \sum_{k\in[n]\setminus\{i,j\}} w_{i,k} w_{k,j} \cdot w_{k,k},$$

where terms of the latter sum can be written as $w_{i,k}w'_{k,j}$ such that $w'_{k,j} = w_{k,j} \cdot w_{k,k}$. Hence, these n^2 sums can be computed by multiplying W and W', and $3 \cdot \texttt{CWC}_3(W)$ is obtained as $\sum_{i<j} w_{i,j} \cdot w_{i,i} \cdot w_{j,j} \cdot p_{i,j}$, where $p_{i,j}$ is the $(i,j)^{\text{th}}$ entry of the product matrix WW'.

[25] The computation of $\texttt{CWC}_t$ is reduced to the computation of $\texttt{CWC}_{t+1}$ by augmenting the original graph with an auxiliary vertex x that is conneced to all original vertices and considering two weighting function that assign x weights 1 and 0, respectively; that is, $\texttt{CWC}_t^G(w) = \texttt{CWC}_{t+1}^{G'}(w1) - \texttt{CWC}_{t+1}^{G'}(w0)$, where $G' = ([n+1], E')$ equals the graph $G = ([n], E)$ augmented with a vertex (denoted $x = n+1$) that is connected to all verices of G.