




Fast Approximate Counting of Cycles

Keren Censor-Hillel   

Department of Computer Science, Technion, Haifa, Israel

Tomer Even 

Department of Computer Science, Technion, Haifa, Israel

Virginia Vassilevska Williams  

Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

We consider the problem of approximate counting of triangles and longer fixed length cycles in directed graphs. For triangles, Tětek [ICALP'22] gave an algorithm that returns a $(1 \pm \varepsilon)$ -approximation in $\tilde{O}(n^\omega/t^{\omega-2})$ time, where t is the unknown number of triangles in the given n node graph and $\omega < 2.372$ is the matrix multiplication exponent. We obtain an improved algorithm whose running time is, within polylogarithmic factors the same as that for multiplying an $n \times n/t$ matrix by an $n/t \times n$ matrix. We then extend our framework to obtain the first nontrivial $(1 \pm \varepsilon)$ -approximation algorithms for the number of h -cycles in a graph, for any constant $h \geq 3$. Our running time is

$$\tilde{O}\left(\text{MM}\left(n, n/t^{1/(h-2)}, n\right)\right), \text{ the time to multiply } n \times \frac{n}{t^{1/(h-2)}} \text{ by } \frac{n}{t^{1/(h-2)}} \times n \text{ matrices.}$$

Finally, we show that under popular fine-grained hypotheses, this running time is optimal.

2012 ACM Subject Classification Mathematics of computing \rightarrow Approximation algorithms; Mathematics of computing \rightarrow Graph algorithms

Keywords and phrases Approximate triangle counting, Approximate cycle counting Fast matrix multiplication, Fast rectangular matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.37

Category Track A: Algorithms, Complexity and Games

Funding *Keren Censor-Hillel*: The research is supported in part by the Israel Science Foundation (grant 529/23).

Virginia Vassilevska Williams: Supported by NSF Grant CCF-2330048, BSF Grant 2020356, and a Simons Investigator Award.

Acknowledgements We would like to thank the anonymous reviewers for their invaluable feedback and for identifying a technical issue in a previous version of our paper.

1 Introduction

Detecting small subgraph patterns inside a large graph is a fundamental computational task with many applications. Research in this domain has flourished, leading to fast algorithms for many tractable versions of the subgraph isomorphism problem: given a fixed (constant size) graph H , detect whether a large graph G contains H as a subgraph, list all copies of H in G , count the copies (exactly or approximately) and more.

The topic of this paper is the fast estimation of the number of copies of a pattern H in a graph G . One of the most studied patterns H is the *triangle* whose detection, listing and approximate counting has become a prime testing ground for ideas in classic graph algorithms [25, 3, 30, 8, 31], sublinear and distributed algorithms [24, 23, 20, 22, 21, 12, 15, 26, 19, 33, 11, 13, 14], streaming [10, 6, 5, 27, 28], parallel [7, 29, 32] algorithms and more. This is largely because triangles are arguably the simplest subgraph patterns and moreover, often algorithms for the triangle version of the (detection, counting or listing) problem formally lead to algorithms for other patterns as well (see Nešetřil and Poljak [30]).



© Keren Censor-Hillel, Tomer Even, and Virginia Vassilevska Williams;
licensed under Creative Commons License CC-BY 4.0

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).

Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;

Article No. 37; pp. 37:1–37:20



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Detecting and finding a triangle, and counting the number of triangles in an n -vertex graph can all be reduced to fast matrix multiplication [25], and the fastest algorithm for these versions has running time $O(n^\omega)$, where ω is the exponent of square matrix multiplication, currently $\omega \leq 2.371552$ [35]. It is also believed that even detecting a triangle requires $n^{\omega-o(1)}$ time, due to known fine-grained reductions that show that Boolean matrix multiplication and triangle detection are equivalent, at least for combinatorial algorithms [34].

Obtaining an approximate count \hat{t} to the number of triangles t in an n -node graph such that $(1 - \varepsilon)t \leq \hat{t} \leq (1 + \varepsilon)t$ for an arbitrarily small constant $\varepsilon > 0$ can be used to detect whether a graph has a triangle, as the algorithm would be able to distinguish between $t = 0$ and $t = 1$. Thus, it is plausible that when the number of triangles is $O(1)$, $n^{\omega-o(1)}$ time is needed to obtain an approximate triangle count.

When the number of triangles t in G is large, however, a simpler sampling approach can obtain a good estimate of t : repeatedly sample a triple of vertices and check whether they form a triangle; in expectation $O(n^3/t)$ samples are sufficient to get a constant factor approximation.

The best known algorithm for approximately counting triangles is by Tětek [33], with running time $\tilde{O}(n^\omega/t^{\omega-2})$. When t becomes constant, the running time becomes $\tilde{O}(n^\omega)$, which is believed to be optimal, as we mentioned earlier. When t becomes $\Theta(n)$, the running time is the same as the naïve sampling algorithm, $\tilde{O}(n^2)$. This quadratic running time is provably necessary even for randomized algorithms (see [21]). Nevertheless, it is unclear whether $\tilde{O}(n^\omega/t^{\omega-2})$ time is needed for all values of t between $O(1)$ and $\Omega(n)$.

*Is there a faster algorithm for approximate triangle counting
when the triangle count is in $[\Omega(1), O(n)]$?*

As triangle counting is an important special case of fixed subgraph isomorphism counting, a natural question is, what is the fastest algorithm for approximately counting arbitrary subgraphs H ?

Dell, Lapinskas and Meeks [18] provide a general reduction from approximate H counting to detecting a “colorful” H in an n -node, m -edge graph, so that a $T(n, m)$ time detection algorithm can be converted into an $\tilde{O}(\varepsilon^{-2}T(n, m))$ time $(1 \pm \varepsilon)$ -approximation algorithm. For many patterns¹ such as triangles and k -cliques or directed h -cycles, the colorful and normal versions of the detection problems are equivalent (e.g. via color-coding [2] and layering). While the detection running time is provably necessary to approximately count when the number of copies of the pattern is constant, similarly to the case of triangles, when the number of copies t is large, faster sampling algorithms are possible. Unfortunately, the reduction of [18] doesn’t seem easy to extend to provide runtime savings that grow with t . Thus, we ask:

*What is the best approximate counting algorithm for subgraph patterns H
with running time depending on the number t of copies of H ?*

As triangles are also cycles, one special case of the above question is when H is a cycle on h vertices.

¹ This equivalence is not true in general: detecting cycles of fixed even length is believed to be computationally easier than detecting colorful even cycles which are known to be equivalent to the directed version of the problem.

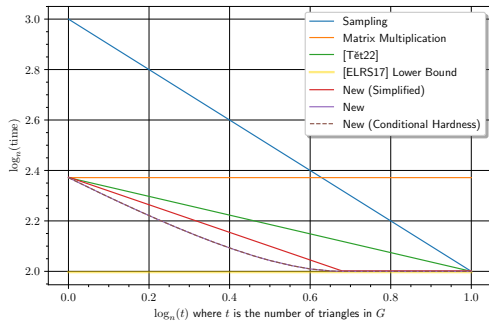
1.1 Our Contribution

The main result of our paper is a new algorithm for approximating the number of h -cycles in a given *directed* graph, for any constant $h \geq 3$, together with a conditional lower bound from a fine-grained hypothesis, showing that the running time of our algorithm is likely tight.

Our main theorem is:

► **Theorem 1** (Approximating the Number of h -Cycles). *Let G be a given graph with n vertices and let $h \geq 3$ be a fixed integer. There is a randomized algorithm that outputs an approximation \hat{t} for the number t of h -cycles in G such that $\Pr [(1 - \varepsilon)t \leq \hat{t} \leq (1 + \varepsilon)t] \geq 1 - 1/n^2$, for any constant $\varepsilon > 0$. The running time is bounded by $\tilde{O}(\text{MM}(n, n/t^{1/(h-2)}, n))$, the fastest running time to multiply an $n \times n/t^{1/(h-2)}$ matrix by an $n/t^{1/(h-2)} \times n$ matrix.*

As long as $\omega > 2$, the running time in the theorem is always upper-bounded by $\tilde{O}\left(n^\omega/t^{(\omega-2)/(1-\alpha)} + n^2\right)$, where $\omega \leq 2.371552$ is the square matrix multiplication exponent mentioned earlier and $\alpha \geq 0.321334$ [35] is the largest real number such that one can multiply $n \times n^\alpha$ by $n^\alpha \times n$ matrices in $n^{2+o(1)}$ time.² It is easy to see that for any value $\omega > 2$, our $\tilde{O}\left(n^\omega/t^{(\omega-2)/(1-\alpha)}\right)$ time for *triangles* is faster than the previous state of the art $\tilde{O}(n^\omega/t^{\omega-2})$ for approximate counting of triangles [33] for all t between $\Omega(1)$ and $O(n)$, answering our first question in the introduction. Figure 1 plots our two running times for approximate triangle counting together with Tětek’s algorithm, naive sampling and the $O(n^\omega)$ time exact counting algorithm.



■ **Figure 1** A comparison between our new running times for approximate triangle counting with prior work, together with the lower bounds, both conditional and unconditional.

Method	Runtime
Sampling	n^3/t
Matrix Multiplication	n^ω
[33]	$n^\omega/t^{\omega-2}$
[21] Lower Bound	n^2
New (Simplified)	$n^\omega/t^{(\omega-2)/(1-\alpha)}$
New	$\text{MM}(n, n, n/t)$
New (Conditional Hardness)	$\text{MM}(n, n, n/t)$

■ **Figure 2** Comparative Runtime Analysis.

We obtain our algorithm via a simplification and generalization of Tětek’s approach that allows us to both obtain an improved running time for triangles, but also to get faster algorithms for longer cycles. The approach can also extend to other patterns; we leave this as future work.

Our algorithm for longer cycles is arguably the first non-trivial algorithm for the problem with a negative dependence on the number of cycles t . To our knowledge, prior to our work the only approximate counting algorithms for h -cycles for $h > 3$ in directed graphs (or in

² We use $\text{MM}(a, b, c)$ to denote the time complexity of multiplying two matrices with dimensions $a \times b$ and $b \times c$.

undirected graphs when h is odd³) were to either use naive random sampling resulting in an $O(n^4/t)$ time or to approximate the answer in the best h -cycle detection time, $O(n^\omega)$ (e.g. via [18]), a running time that does not depend on t .

We complement our algorithms for approximately counting h -cycles with a *tight* conditional lower bound under a popular fine-grained hypothesis. The k -Clique Hypothesis of fine-grained complexity (e.g. [1, 4, 9, 16]) postulates that the current fastest algorithms for detecting a k -clique in a graph (for constant $k \geq 3$) are optimal, up to $n^{o(1)}$ factors. We formulate a natural hypothesis about the complexity of *triangle detection* in unbalanced tripartite graphs that is motivated by and in part implied by the k -Clique Hypothesis. Then we show, under that hypothesis:

► **Theorem 2.** *Under fine-grained hypotheses, in the word-RAM model with $O(\log n)$ bit words, for any constant integer $h \geq 3$, any randomized algorithm that, when given an n node directed graph G , can distinguish between G being h -cycle-free and containing $\geq t$ h -cycles needs $\text{MM}(n, n/t^{1/(h-2)}, n)^{1-o(1)}$ time. The same result holds for undirected graphs as well whenever h is odd.*

As any algorithm that can approximate the number t of h -cycles multiplicatively, can distinguish between 0 and t h -cycles, we get that our algorithm running times are essentially tight. We present our lower bound for triangles in Figure 1 as a dotted line. Together with the lower bound by [21], our lower bound shows that our algorithm is (conditionally) optimal for all values of t . Due to space considerations, the lower bound proof is deferred to the full version.

Similarly to Tětek’s algorithm, our algorithms for approximate h -cycle counting can be used to obtain improved h -cycle counting algorithms for *sparse* graphs, where the running time is measured in terms of the number of edges m . In particular, for triangles, one can simply substitute our new algorithm in terms of n in Tětek’s argument [33] to obtain an approximate counting algorithm that runs in time $\tilde{O}\left(m^{2\omega/(\omega+1)}/t^{\frac{2(\omega-1)}{\omega+1} + \frac{\alpha(\omega-2)}{(1-\alpha)(\omega+1)}}\right)$. This running time is always faster than Tětek’s $\tilde{O}\left(m^{2\omega/(\omega+1)}/t^{\frac{2(\omega-1)}{\omega+1}}\right)$ for any $\omega > 2$. One can similarly adapt the algorithms of Yuster and Zwick [37] and their analysis in [17] to obtain approximate counting algorithms for longer cycles. We leave this to future work.

1.2 Technical Overview

To frame our technical contribution, we first briefly overview the approach of [33]. The latter gives a randomized approximate counting algorithm for triangles, in time $\tilde{O}(n^\omega/t^{\omega-2})$. In a nutshell, the algorithm finds a subset of vertices S that contains all Λ -heavy vertices and no $\Lambda/\text{polylog}(n)$ -light vertices – a vertex is called Λ -heavy if it participates in at least Λ triangles, and otherwise it is called Λ -light. Then, the algorithm approximately counts the number of Λ -heavy triangles, which are triangles with at least one heavy vertex. The algorithm then continues by sampling subsets of vertices from the set $V - S$, where each vertex is kept independently uniformly at random with some probability, and processing the sampled graphs by recursion.

³ The detection problem for even h -cycles in undirected graphs is known to be much easier than that for odd cycles, and for directed graphs, as for every even constant integer h , an $O(n^2)$ time algorithm was developed by Yuster and Zwick [36]. Meanwhile, directed h -cycles and undirected odd h -cycles are believed to require $n^{\omega-o(1)}$ time to detect.

Our technical contribution consists of three parts: (I) We simplify the above recursive approach, (II) we improve upon the component that finds heavy vertices, and (III) we improve upon the component that counts the number of cycles that contain heavy vertices. A compelling aspect of our technique is that it applies to any constant-length cycle. In what follows, we overview each of these aspects.

I. The Recursive Template. In [33], each recursive invocation triggers seven further recursive calls and takes the median of their return values. As the depth of the recursion increases, the algorithm needs to use a precision parameter that becomes exponentially tighter.

In contrast, our algorithm initiates only a single recursive call. This allows us to avoid having to compute the median of several subcalls, which makes it easier to apply standard amplification tools. In particular, it allows us to *fix the precision parameter*.

In addition, by reducing the recursive tree to a “path”, we *simplify the analysis* of the running time.

A prime feature of our simplification of the recursion is that it allows us to present it as a *template* for approximate subgraph counting for any fixed subgraph H , provided one designs the two black boxes (one for finding a superset S of the Λ -heavy vertices with no $\Lambda/\text{polylog}(n)$ -light vertices, and another for approximately counting the number of copies of H that intersect the set S).

For triangles, our improvement comes from simplifying the recursion and implementing the black box that finds heavy vertices faster, using rectangular matrix multiplication. Crucially, our implementations of these black boxes are general, in the sense that they apply to constant length h -cycle. Specifically, we find the set S in time $\tilde{O}(\text{MM}(n, n/\Lambda^{1/(h-2)}, n))$. For such S , we find an $(1 + \epsilon)$ approximation for the number of copies of H that intersect S in time $\tilde{O}(n^2/\epsilon^3)$, which is independent of Λ and of the cardinality of S .

II. Finding the Heavy Vertices. The algorithm in [33] finds Λ -heavy vertices by sampling a subset of vertices uniformly at random, and using matrix multiplication to detect triangles inside the induced sampled subgraph. This takes $\tilde{O}(n^\omega/\Lambda^{\omega-2})$ time, by $\tilde{O}(\Lambda^2)$ repetitions of multiplying $n/\Lambda \times n/\Lambda$ matrices.

At the heart of our approach for finding the heavy vertices lies non-uniform sampling, and computing the product of rectangular matrices rather than square ones. We obtain a running time of $\tilde{O}(\text{MM}(n, n/\Lambda^{1/(h-2)}, n)) = \tilde{O}(n^\omega/\Lambda^{\gamma_h})$ for h -cycles, where $\gamma_h \triangleq \frac{\omega-2}{(1-\alpha)(h-2)}$. This comes from multiplying an $n \times n/\Lambda^{1/(h-2)}$ matrix by an $n/\Lambda^{1/(h-2)} \times n$ matrix. In [35] it was shown that $\alpha \geq 0.321334$, and therefore $\frac{\omega-2}{1-\alpha} \geq 1.47(\omega-2)$, which establishes that our algorithm is never slower, and is faster (if $\omega > 2$), where the gap increases with Λ (for sufficiently large Λ , the folklore naïve sampling algorithm is superior).

Our starting point for finding the heavy vertices is the *color-coding* technique of [2], which is widely employed for detecting h -cycles for $h = \mathcal{O}(1)$. This technique colors the vertices using h colors uniformly independently at random and looks for *colorful* h -cycles, which are h -cycles with exactly one vertex of each color. This restriction allows for faster detection but suffers some probability of missing h -cycles that are colored out of order, which can be overcome with sufficiently large probability by repeating this process.

To find colorful h -cycles, we utilize matrix multiplication. However, we do so in a refined manner. Rather than considering all vertices, we sample a subset of vertices from each color class in a nonuniform manner. To illustrate this, consider the task of finding Λ -heavy vertices w.r.t. triangles. We assign a random color to each vertex, and denote the three color classes by V_1, V_2, V_3 . We focus on identifying the Λ -heavy vertices within V_1 . Fix some $i \in [\log \Lambda]$.

We sample each vertex from V_2 with probability $2^i/\Lambda$, and we sample each vertex from V_3 with probability $1/2^i$. Let H_i denote the induced graph obtained by all vertices from V_1 and the sampled vertices from V_2 and V_3 , where we also direct edges from V_j to $V_{j+1 \pmod 3}$ and discard monochromatic edges. We show that for every Λ -heavy vertex v , there exists an index $i \in [\log \Lambda]$ such that v is in a triangle in H_i with some probability at least p_{heavy} , where $1/p_{\text{heavy}} = \tilde{O}(1)$. On the other hand, for $\Lambda/\text{polylog}(n)$ -light vertex u , we show that for every $i \in [\log \Lambda]$, the vertex u is in a triangle in H_i with probability at most $p_{\text{heavy}}/2$. Therefore, we can distinguish between these cases. Checking whether v is in a triangle in H_i can be done in $\tilde{O}(\text{MM}(n, n, n/\Lambda))$ time. Using amplification, we approximate the probability that v is in a triangle in H_i for every $v \in V_1$, and thus distinguish heavy vertices from lighter ones.

We generalize our approach for h -cycles by coloring the vertices with h colors, and directing edges and discarding monochromatic edges, as for triangles. We also discard edges between non-consecutive color classes. To find Λ -heavy vertices in the first color class, we sample in a nonuniform manner a subset of vertices from the j -th color class for $2 \leq j \leq h$, where the product of the sampling probabilities of the color classes should be at most $1/\Lambda$, as for triangles. Let H denote the obtained random induced subgraph. The running time of computing the exact number of h -cycles each vertex in H participates in, which is dominated by the size of the smallest color class in H , becomes $\tilde{O}(\text{MM}(n, n, n/\Lambda^{1/(h-2)}))$. To see why, consider an h -partite graph G with n vertices in each part. Suppose G has a vertex $v \in V_1$ with a neighbor $u \in V_2$, such that all h -cycles that intersect v , also intersect the edge (u, v) . Now, suppose each vertex set V_j , for $3 \leq j \leq h$ has a subset W_j of $\Lambda^{1/(h-2)}$ vertices, such that any h -tuple of the form $(v, u, w_3, w_4, \dots, w_h)$ is an h -cycle in G , where $w_j \in W_j$ for $3 \leq j \leq h$. This implies that v is Λ -heavy. Note that if we keep each vertex from the j -th color class with a probability of $o(1/\Lambda^{1/(h-2)})$, we are unlikely to sample any vertex from W_j , and therefore we fail to learn that v is Λ -heavy. On the other hand, if we sample vertices from each class with probability $\Omega(1/\Lambda^{1/(h-2)})$, the smallest color class is of size $\Omega(n/\Lambda^{1/(h-2)})$.

III. Counting the Heavy Copies. Given a graph G and a subset of vertices S , where each vertex in S participates in at least a and at most b copies of h -cycles for $h = \mathcal{O}(1)$, we show how to compute a $(1 + \epsilon)$ approximation for the number of h -cycles that intersect the set S , in time $\tilde{O}(n^2 b/\epsilon a)$. In particular, the runtime is independent of size of the set S .

Consider a naïve approach, which approximates the average number of h -cycles that a vertex from S intersects, and let us see why it fails to provide a good approximation for the total number of h -cycles intersecting S . Suppose $h = 3$ and $|S| = 3$ and each vertex $v \in S$ participates in exactly one triangle in G . Based solely on the number of triangles in which a vertex participates, it is impossible to distinguish the case where the set S intersects one triangle in G from the case in which it intersects three triangles in G . The issue here is double counting, as we did not avoid counting the same cycle more than once. For triangles, this obstacle can be avoided by replacing G with a tripartite graph G' , where each of the three parts is a copy of V , and for each edge in G there are six edges in G' , one for every ordered pair of parts. It is easy to see that every triangle in G corresponds to six triangles in G' , and thus an estimate on G' directly gives an estimate on G . That is, we sample a subset F of vertices from S , and for each copy v' of $v \in F$ in the, say, first part of in the tri-partition G' , we compute the number of triangles that go through it in G' . This avoids double counting, because each triangle in G' intersects copies of the set F from the first part at most once (as vertices in the same part form an independent set and hence cannot be in

the same triangle). To summarize, restricting to triangles would significantly simplify this part to a single Chernoff inequality for independent random variables. The source of this simplicity is that triangles are cliques. However, larger cycles are not cliques. If we simply repeat every vertex h times to create a new graph G' as in the triangle case, we could create h -cycles in G' that do not correspond to h -cycles in G : every *closed walk* of length h would become an h -cycle. As in [2], we use color-coding to overcome this, and this necessitates a more careful probabilistic analysis.

First, we sample a subset of vertices from S and for each sampled vertex v we approximate the number of h -cycles which go through v (and therefore intersect the set S). The crux of our algorithm is that in order to approximate the above, we approximate the number of h -cycles which go through v and intersect the set S exactly k times, for each $1 \leq k \leq h$. The summation of these approximations yields our final result, and it naturally avoids the pitfall of double-counting.

To approximate the number of h -cycles intersecting the set S exactly k times for some k , we color the graph with $h - 1$ colors and color vertex v with the color h . This ensures that any colorful h -cycle intersects v . Then, we choose $k - 1$ color classes from the first $h - 1$ classes, and retain only vertices of S within those classes. For the remaining color classes, we keep only vertices that are not in S . The color class h is fixed and always contains only v . This promises that each colorful h -cycle with v in this auxiliary graph intersects S exactly k times. The number of ways to choose exactly $k - 1$ color classes that keep only vertices from S is $\binom{h-1}{k-1}$. We compute the number of h -cycles in each such auxiliary graph. We prove that the expectation of this number is some fixed constant multiplicative factor off from the number of h -cycles intersecting v and S exactly k times. Finally, we prove that the variance of this random variable is suitably bounded. Therefore, conducting this process $\tilde{O}(b/(a\varepsilon))$ times enables us to obtain an $(1 \pm \varepsilon)$ approximation for its expectation by Chebyshev's inequality. We compute the number of h -cycles in this auxiliary graph using rectangular matrix multiplication. Since the auxiliary graph is h -partite and one part contains only a single vertex, we get a running time of $\text{MM}(n, n, 1) = \tilde{O}(n^2)$. Thus, we achieve our claimed running time of $\tilde{O}(n^2 b/\varepsilon a)$.

We mention that we invoke this procedure on the set of vertices given by the previous component of finding heavy vertices, which is called upon in every recursion step. A crucial observation that we make is that not only does this set contain all Λ -heavy vertices and no $\Lambda/\text{polylog}(n)$ -light vertices, but rather we also know that it does not contain $(2^h \Lambda)$ -heavy vertices, because those are handled during previous steps of the recursion. This means that we invoke this procedure for a, b that differ only by $\text{polylog}(n)$ and ε^2 factors, and thus we effectively get a running time of $\tilde{O}(n^2/\varepsilon^3)$ for counting h -cycles through Λ -heavy vertices.

Roadmap. Section 2 contains our template for the recursion, and proves its correctness for any graph \mathbb{H} given implementations of two black boxes, one that finds heavy vertices and another that counts the copies of \mathbb{H} that contain heavy vertices. Section 3 proves the running time that our template obtains for h -cycles, given the running times of implementations of the two black boxes. We implement our black boxes for h -cycles in Sections 4 and 5. Missing proofs, as well as our hardness result, appear in the full version.

1.3 Preliminaries

Let G be a graph on n vertices. Let \mathbb{H} be a fixed graph with $h = \mathcal{O}(1)$ vertices. For a subgraph $G' \subseteq G$, and a subset of vertices S , denote by $t_{G'}(S)$ number of copies of \mathbb{H} in G' which intersect S . Denote by $\tau = \tau_{G'}$ the maximal number of copies of \mathbb{H} in G' in which a

vertex participates. We say that a vertex v is Λ -heavy (in G) if $t_G(v) \geq \Lambda$, and otherwise it is Λ -light. We say that a copy C of \mathbb{H} is Λ -heavy if C contains at least one Λ -heavy vertex. Let G be a graph and p some parameter that could depend on G . We denote by $G[p]$ a random induced subgraph of G obtained by keeping each vertex from G independently with probability p . We use $t(1 \pm \varepsilon)$ to denote the closed interval $[t(1 - \varepsilon), t(1 + \varepsilon)]$. We say that a value $\hat{t} = t(1 \pm \varepsilon)$ if $\hat{t} \in [t(1 - \varepsilon), t(1 + \varepsilon)]$. We assume that $\varepsilon \in (0, 1/2]$, which might depend on n . If ε is bigger, our algorithm assumes $\varepsilon \leq 1/2$. Finally, all logarithms in this paper are base 2.

► **Definition 3 (Fast Matrix Multiplication Definitions).** We denote the time it takes to compute the product of two matrices of dimension $n^a \times n^b$ and $n^b \times n^c$ by either $\text{MM}(n^a, n^b, n^c)$ or $n^{\omega(a,b,c)}$. We also abuse the notation and write $\omega = \omega(1, 1, 1)$, and $\omega(k) = \omega(1, k, 1)$. Note that for any permutation $\pi : [3] \rightarrow [3]$ we have $\omega(x_1, x_2, x_3) = \omega(x_{\pi(1)}, x_{\pi(2)}, x_{\pi(3)})$. In addition to ω , we will also use α to be the largest real number such that n by n^α by n matrix multiplication can be done in $n^{2+o(1)}$ time.

2 The Recursive Template

Organization. In this section, we present an algorithm for approximating the number of copies of a graph \mathbb{H} in a graph G , denoted by t , which builds upon two black boxes. The first black box, called **Find-Heavy**, takes a graph H and a parameter Λ as input and computes a superset of the Λ -heavy vertices, excluding any $\Lambda/\text{polylog}(n)$ -light vertices. We denote the computed superset of heaviest vertices as S . The second black box, called **Count-Heavy**, is used to compute an approximation for the number of *heavy-copies* of \mathbb{H} in G , which is the set of all copies that contain at least one vertex from S . Our algorithm for subgraph approximate counting that uses the specified black boxes consists of two parts: a doubling algorithm called **Doubling-Template**, and a recursive algorithm called **Template**, which is the main focus of this section.

The Template Algorithm. The $\text{Template}_{\varepsilon'}(G, \Lambda)$ algorithm takes two parameters: a graph G and a heaviness threshold Λ . The output of the algorithm is a value \hat{t} , which, with a probability of at least $2/3$, is within the range $t \pm (t \cdot \varepsilon' + \Lambda \cdot \text{polylog}(n) / \varepsilon')$, where ε' is the fixed precision parameter of the algorithm.

We next explain how the recursive **Template** algorithm works. The algorithm does the following. (1) Find the heaviest vertices using the **Find-Heavy** black box, and denote this set by V_Λ . (2) Compute an approximation to the number of heavy copies of \mathbb{H} , i.e., copies of \mathbb{H} with at least one vertex from V_Λ , using the **Count-Heavy** black box, and denote the output by \hat{t}_Λ . (3) Let $H = G[V(G) - V_\Lambda]$, and let $F \leftarrow H[p]$. That is, F is an induced subgraph of H , where each vertex from H joins F independently with probability p . (4) Make a recursive call to $\text{Template}_{\varepsilon'}(H, \Lambda \cdot p^{|\mathbb{H}|})$ and let \hat{t}_H denote its output. (5) Return $\hat{t}_\Lambda + \hat{t}_H/p^{|\mathbb{H}|}$. The analysis of the probability that the algorithm produces a good approximation appears in the proof of Lemma 7. The running time of the algorithm depends on the implementation of the black boxes. In the next section, we analyze the running time of the algorithm for the case where \mathbb{H} is a cycle.

The Doubling-Template Algorithm. The **Doubling-Template** algorithm is a doubling algorithm, which starts with an initial guess for t , denoted by $W_0 = n^{|\mathbb{H}|}$. This is the maximal number of copies of \mathbb{H} an n vertex graph can contain ($h! \cdot \binom{n}{|\mathbb{H}|} \leq n^{|\mathbb{H}|}$). The algorithm then makes $\tilde{O}(1)$ calls to $\text{Template}_{\varepsilon'}(G, \Lambda_0)$, where $\Lambda_0 \leftarrow W \cdot \varepsilon^2 / 8Q$, where $Q = 8 \log^4(n)$, and

computes their median, which we denote by \hat{t}_0 . If $\hat{t}_0 \geq W_0$ the doubling algorithm stops and outputs \hat{t}_0 as its approximation for t . Otherwise, the guess for the value of t is decreased by a factor of 2. The main point here is that the smaller the value of Λ given to the recursive algorithm is, the better approximation we get, while simultaneously increasing the running time. The guess W is a guess for the highest heaviness threshold the algorithm can start with to output a good approximation, and not an actual guess for the value of t (although both quantities are related).

Formally, the black boxes that we assume are the following.

Find-Heavy(G, Λ)

Input: A graph G , some parameter Λ .
Output: A subset V_Λ of vertices, such that with probability at least $1 - \frac{1}{n^4}$, $\forall v \in V(G)$:

1. If $t_G(v) \geq \Lambda$, then $v \in V_\Lambda$.
2. If $t_G(v) \leq \Lambda/(\log n)^{h^2}$, then $v \notin V_\Lambda$.

Count-Heavy $_{\varepsilon'}$ (G, V_Λ, a, b)

Input: A graph G , a precision parameter ε' , a subset of vertices V_Λ , and two real numbers $0 < a \leq b$, such that $\forall v \in V_\Lambda$ we have $t_G(v) \in [a, b]$.
Output: \hat{t}_Λ which satisfies $\Pr[\hat{t}_\Lambda = t_G(V_\Lambda)(1 \pm \varepsilon')] \geq 1 - \frac{1}{n^4}$.

It should be noted that **Count-Heavy** cannot be applied to the entire graph, as it might contain a vertex v with $t_G(v) = 0$. Moreover, even if all vertices have $t_G(v) > 0$, employing this black box on the entire graph might result in slower running time. Indeed, in our implementation of the black box, the runtime is contingent on the ratio b/a . Therefore, we only employ this black box with $b/a = \mathcal{O}(1/\varepsilon^2)$.

■ **Algorithm 1** $\text{Template}_{\varepsilon'}(G, \Lambda)$.

Input: A graph $G = (V, E)$, a heaviness threshold Λ , and a precision parameter ε' .

- 1 $V_\Lambda \leftarrow \text{Find-Heavy}(G, \Lambda)$; $\triangleright V_\Lambda$ is a superset of Λ -heavy vertices in G
with no $\Lambda/(\log n)^{h^2}$ -light vertices
- 2 $a_\Lambda \leftarrow \frac{\Lambda}{(\log n)^{h^2}}, b_\Lambda \leftarrow \Lambda \cdot \frac{8Q}{\varepsilon^2}$; $\triangleright Q = 8 \log^4(n)$
- 3 $\hat{t}_\Lambda \leftarrow \text{Count-Heavy}_{\varepsilon'}(G, V_\Lambda, a_\Lambda, b_\Lambda)$; $\triangleright \hat{t}_\Lambda$ is a $(1 \pm \varepsilon')$ approximation for
 $t_G(V_\Lambda)$ (the number of copies of H intersecting V_Λ)
- 4 **if** $\Lambda \leq 1$ **then return** \hat{t}_Λ ;
- 5 $H \leftarrow G[V - V_\Lambda]$;
- 6 $p \leftarrow 1/2$; \triangleright We keep p instead of $1/2$ for readability
- 7 $F \leftarrow H[p]$; $\triangleright F$ is a random subgraph of H
- 8 $\hat{t}_F \leftarrow \text{Template}_{\varepsilon'}(F, \Lambda \cdot p^h)$;
- 9 **return** $\hat{t}_\Lambda + \hat{t}_F/p^h$;

The depth of the recursion in $\text{Template}_{\varepsilon'}(G, \Lambda)$ is at most $\log_{1/p^h}(\Lambda) + 1$. Since we will only call this algorithm with $\Lambda \leq |V(G)|^h$, we can conclude that the depth of the recursion is at most $\log n + 1$. The guarantees for the template are given in the following lemma.

37:10 Fast Approximate Counting of Cycles

► **Lemma 4** (Guarantees for the Template Algorithm). *For every $\varepsilon \in (0, 1/2]$ and every $\Lambda \geq \tau_G \cdot \frac{\varepsilon^2}{8Q}$, we have $\Pr \left[\text{Template}_{\varepsilon'}(G, \Lambda) = t_G(1 \pm \frac{\varepsilon}{2}) \pm \Lambda \cdot \frac{\log^4(n)}{2\varepsilon} \right] \geq \frac{2}{3}$, where $\varepsilon' = \frac{\varepsilon}{4 \log n}$.*

■ **Algorithm 2** Doubling-Template(G, ε).

Input: A graph $G = (V, E)$ with t_G copies of H and a precision parameter $\varepsilon \leq 1/2$.
Output: \hat{t} , which is a $(1 \pm \varepsilon)$ approximation for t w.h.p.

- 1 $\varepsilon' \leftarrow \frac{\varepsilon}{4 \log n}$, $W \leftarrow n^h$, $Q \leftarrow 8 \log^4(n)$, $\Lambda \leftarrow W \cdot \varepsilon^2 / Q$
- 2 **for** $i = 0$ **to** $i = h \log n$ **do**
- 3 $\hat{t}_i \leftarrow \text{Median} [\text{Template}_{\varepsilon'}(G, \Lambda/2^i), 400 \log n]$; ▷ \hat{t}_i is the median of
 $400 \log n$ independent executions of $\text{Template}_{\varepsilon'}(G, \Lambda/2^i)$.
- 4 **if** $\hat{t}_i \geq W/2^i$ **then return** \hat{t}_i ;
- 5 **return** *Exact deterministic count of t_G .*

► **Lemma 5** (Guarantees for the Doubling-Template Algorithm). *Let G be a graph with n vertices and t_G copies of H . Fix some $\varepsilon > 0$ that may depend on n . Let \hat{t} denote the output of Doubling-Template(G, ε) (specified in Algorithm 2). Then, $\Pr [\hat{t} = t(1 \pm \varepsilon)] \geq 1 - 1/n^2$.*

The main result that we prove in this section is Lemma 5. We prove it using Lemma 4. First, we use amplification, to show that the event specified in Lemma 4 occurs with high probability, and not only with probability at least $2/3$. The rest assumes that this event always occurs. We then use case analysis on $\Lambda/2^i$.

1. For i such that $\Lambda/2^i \geq 4t_G$, we show that $\hat{t}_i < \Lambda/2^i$ w.h.p., meaning the doubling algorithm does not stop for such i w.h.p., and makes another iteration with a refined initial heaviness threshold.
2. For i such that $\Lambda/2^i \leq 4t_G$, we show that $\hat{t}_i = t_G(1 \pm \varepsilon)$ w.h.p.
3. For i such that $\Lambda/2^i \leq t_G/2$, we show that $\hat{t}_i \geq \Lambda/2^i$ w.h.p., which means the algorithm stops as soon as $\Lambda/2^i \leq t_G/2$ w.h.p.

To summarize, the doubling algorithm always stops (by the third property). It does not stop when $\Lambda/2^i \geq 4t_G$ w.h.p. Therefore, when it does stop we have that $\Lambda/2^i \leq 4t_G$, and then it obtains a $(1 \pm \varepsilon)$ approximation for t_G .

To prove Lemma 4, we state and prove a more refined version of the guarantees of the template algorithm. We need the following definition to restate it.

► **Definition 6** ($\hat{D}(\Lambda)$). *We define the depth of the call $\text{Template}_{\varepsilon'}(F, \Lambda)$ as $\hat{D}(\Lambda) = \max \left\{ 0, \left\lceil \log_{1/p^h}(\Lambda) \right\rceil \right\}$.*

We assume $\Lambda \leq (h!) \cdot \binom{n}{h} \leq n^h$, and therefore that $\hat{D}(\Lambda) \leq \log n + 1$.

► **Lemma 7** (Induction Hypothesis). *Given a graph G and ε , we set $p = 1/2$, and $\varepsilon' = \frac{\varepsilon}{4 \log n}$. Then, for any $\Lambda \geq \tau_G \cdot \frac{\varepsilon^2}{8Q}$, and any $K = o(n)$ that could depend on $n, \hat{D}(\Lambda)$ and p , we have*

$$\Pr \left[\text{Template}_{\varepsilon'}(G, \Lambda) = t_G(1 \pm \varepsilon')^{\hat{D}(\Lambda)} \pm 2\hat{D}(\Lambda) K \cdot \Lambda / \varepsilon' \right] \geq 1 - 4h\hat{D}(\Lambda) / (Kp^h).$$

Proof Sketch. We prove using induction on the depth of the recursion. We skip the proof of the base case and state the induction hypothesis. The full proof appears in the full version.

Step. We first define some notation and events. We have three graphs $F \subseteq H \subseteq G$, where G is the input graph, H is an induced subgraph of G without the Λ -heavy vertices, and F is a random induced graph of H obtained by keeping each vertex of H independently with probability p . Let \hat{t}_G denote the output of $\text{Template}_{\varepsilon'}(G, \Lambda)$, and let \hat{t}_F denote the output of $\text{Template}_{\varepsilon'}(F, \Lambda \cdot p^h)$. We use t_Λ to denote $t_G(V_\Lambda)$, and let \hat{t}_Λ denote the output of $\text{Count-Heavy}_{\varepsilon'}(G, V_\Lambda)$. Note that $\hat{D}(\Lambda) \geq \hat{D}(\Lambda p^h) + 1$ (unless $\hat{D}(\Lambda) = 0$, in which case $\hat{D}(\Lambda) = \hat{D}(\Lambda p^h) = 0$).

Intuition. We compute two values, \hat{t}_Λ and \hat{t}_F . We then output $\hat{t} = \hat{t}_\Lambda + \hat{t}_F/p^h$ as an approximation for t_G . By the black box guarantees, we have that \hat{t}_Λ is a good approximation for t_Λ . What is left is to show that \hat{t}_F/p^h is a good approximation for t_H . We split this into two parts. First, we show that t_F/p^h is “close” to the value of t_H . Next, we use the induction hypothesis, to show that \hat{t}_F is a good approximation of t_F . We need to show that the “composition” of these approximations is also good. Let $\mathcal{E}_{\text{Find-Heavy}}$ denote the event that all calls made to the Find-Heavy black box produce a valid output. That is, the event that V_Λ contains a superset of the Λ -heavy vertices, without any $\Lambda/(\log n)^{h^2}$ n -light vertices. **We prove the correctness of the algorithm under the assumption that $\mathcal{E}_{\text{Find-Heavy}}$ occurs.** We define:

1. $\mathcal{E}_1 \triangleq \{\hat{t}_\Lambda = t_\Lambda(1 \pm \varepsilon')\}$. The heavy copies of H are approximated correctly.
2. $\mathcal{E}_2 \triangleq \{\hat{t}_F = t_F(1 \pm \varepsilon')^{\hat{D}(\Lambda p^h)} \pm 2\hat{D}(\Lambda p^h) \cdot K \cdot (\Lambda \cdot p^h)/\varepsilon'\}$. This is the event in the induction hypothesis.
3. $\mathcal{E}_3 \triangleq \{t_F/p^h = t_H(1 \pm \varepsilon') \pm K \cdot \Lambda/\varepsilon'\}$. This is a concentration bound on t_F .
4. $\mathcal{E}_4 \triangleq \{\hat{t}_F/p^h = t_H(1 \pm \varepsilon')^{\hat{D}(\Lambda)} \pm 2\hat{D}(\Lambda) K \cdot \Lambda/\varepsilon'\}$. This event contains $\mathcal{E}_2 \cap \mathcal{E}_3$.
5. $\mathcal{E}_5 \triangleq \{\hat{t}_G = t_G(1 \pm \varepsilon')^{\hat{D}(\Lambda)} \pm 2\hat{D}(\Lambda) K \cdot \Lambda/\varepsilon'\}$. This is the event specified in Lemma 7.

Lemma 7 requires us to show that $\Pr[\mathcal{E}_5] \geq 1 - \frac{4h\hat{D}(\Lambda)}{Kp^h}$. We show this by proving that $\mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3 \subseteq \mathcal{E}_5$, which implies that it is sufficient to prove that $\Pr[\mathcal{E}_{\text{Find-Heavy}} \cap \mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3] \geq 1 - \frac{4h\hat{D}(\Lambda)}{Kp^h}$. To show the latter, we show that $\Pr[\mathcal{E}_{\text{Find-Heavy}} \cap \mathcal{E}_1 \cap \mathcal{E}_2] \geq 1 - \frac{4h\hat{D}(\Lambda p^h) + 2}{Kp^h}$ and that $\Pr[\mathcal{E}_3] \geq 1 - \frac{h+1}{Kp^h}$. Summing up the two error probabilities in the above expressions gives the desired result. \blacktriangleleft

3 Application: Approximating the Number of h -Cycles

In this section, we analyze the running time of the recursive and doubling algorithm, when the counted subgraph is an h -cycle for $h = \mathcal{O}(1)$. For this task, we assume the black boxes can be implemented in specific runtime, as stated in the following lemma which is proven in the next sections.

► **Lemma 8.** *Let G be a graph with n vertices, let H be an h -cycle for some $h = \mathcal{O}(1)$, and let ε' be some parameter. Then, each call to $\text{Find-Heavy}(G, \Lambda)$ can be implemented in time $\tilde{\mathcal{O}}(\text{MM}(n, n, \frac{n}{\Lambda^{1/(h-2)}}))$ and each call to $\text{Count-Heavy}_{\varepsilon'}(G, V_\Lambda, a_\Lambda, b_\Lambda)$ can be implemented in time $\tilde{\mathcal{O}}(\text{MM}(n, n, 1) \cdot \frac{b_\Lambda}{a_\Lambda \cdot \varepsilon}) = \tilde{\mathcal{O}}(n^2 \cdot \frac{b_\Lambda}{a_\Lambda \cdot \varepsilon})$.*

► **Theorem 1 (Approximating the Number of h -Cycles).** *Let G be a given graph with n vertices and let $h \geq 3$ be a fixed integer. There is a randomized algorithm that outputs an approximation \hat{t} for the number t of h -cycles in G such that $\Pr[(1 - \varepsilon)t \leq \hat{t} \leq (1 + \varepsilon)t] \geq 1 - 1/n^2$, for any constant $\varepsilon > 0$. The running time is bounded by $\tilde{\mathcal{O}}(\text{MM}(n, n/t^{1/(h-2)}, n))$, the fastest running time to multiply an $n \times n/t^{1/(h-2)}$ matrix by an $n/t^{1/(h-2)} \times n$ matrix.*

Proof Sketch. Consider the Doubling-Template (G, ε) . Its correctness follows from Lemma 5. We analyze its running time. We prove that given Lemma 8, the running time of the recursive algorithm $\text{Template}_{\varepsilon'}(G, \Lambda)$ is bounded by $\tilde{\mathcal{O}}\left(\text{MM}\left(n, n, \frac{n}{\Lambda^{1/(h-2)}}\right) + \frac{n^2}{\varepsilon^3}\right)$ with probability at least $1 - \exp(-\log^2 n)$. Then, we show that the complexity of the doubling algorithm is bounded by $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^3} \cdot \text{MM}\left(n, n, \frac{n}{\Lambda^{1/(h-2)}}\right)\right)$ with probability at least $1 - \frac{1}{n^3}$. The full details appear in the full version. Here we give a sketch of why the running time of $\text{Template}_{\varepsilon'}(G, \Lambda)$ is bounded by $\tilde{\mathcal{O}}\left(\text{MM}\left(n, n, \frac{n}{\Lambda^{1/(h-2)}}\right) + \frac{n^2}{\varepsilon^3}\right)$ with probability at least $1 - \exp(-\log^2 n)$.

We unroll the recursion of the algorithm: it makes a single call to **Find-Heavy**, then a single call to **Count-Heavy** $_{\varepsilon'}$, and then a recursive call. The algorithm's runtime can be analyzed by bounding $\{\text{Find-Heavy}(G_k, \Lambda_k)\}_{k=0}^r$, and $\{\text{Count-Heavy}_{\varepsilon'}(G_k, V_{\Lambda_k}, a_{\Lambda_k}, b_{\Lambda_k})\}_{k=0}^r$, where r is the recursion depth and G_k denotes the input graph for the k -th call of the recursive algorithm, where $G_0 = G$. Let $\Lambda_k = \Lambda_0 \cdot p^{kh}$ denote the heaviness threshold, and let V_{Λ_k} denote the set of Λ_k -heavy vertices in the k -th iteration. We bound the running time of the k -th call to each of the black boxes. We prove in the full version that the total running time for all calls to the **Count-Heavy** $_{\varepsilon'}$ black-box is $\tilde{\mathcal{O}}(n^2/\varepsilon^3)$. Next, we bound the running time of the k -th call to the **Find-Heavy** black-box. Note that for any k , the call $\text{Find-Heavy}(G_k, \Lambda_k)$ takes $\tilde{\mathcal{O}}(\text{MM}(|V(G_k)|, |V(G_k)|, |V(G_k)|/\Lambda_k^{1/(h-2)}))$ by Lemma 8. We use Chernoff's inequality to show that the number of vertices in G_k is $\tilde{\mathcal{O}}(\max\{1, np^k\})$, thus we can replace $|V(G_k)|$ by np^k in the above expression.

The crux is that the running time of the first call to **Find-Heavy** also applies to subsequent calls. This is because $\text{MM}(np^k, np^k, np^k/(\Lambda \cdot p^{hk})^{1/(h-2)}) \leq \text{MM}(n, n, \frac{n}{\Lambda^{1/(h-2)}} \cdot p^{k \cdot (3-h/(h-2))}) \leq \text{MM}(n, n, n/\Lambda^{1/(h-2)})$, where the first inequality is a simple observation that we prove in the full version, and the second inequality follows since $3 - h/(h-2) = 2\frac{h-3}{h-2}$ is non-negative for $h \geq 3$ and therefore $p^{k \cdot (3-h/(h-2))} \leq 1$. We conclude that the running time of each call in $\{\text{Find-Heavy}(G_k, \Lambda_k)\}_{k=0}^r$ is at most $r \cdot \mathcal{O}(\text{MM}(n, n, n/\Lambda^{1/(h-2)}))$. As $r \leq \log n$, we get that all calls take a total of $\tilde{\mathcal{O}}(\text{MM}(n, n, n/\Lambda^{1/(h-2)}))$ time.

This completes the proof, as all calls to the **Find-Heavy** black box and the **Count-Heavy** $_{\varepsilon'}$ black box take at most $\tilde{\mathcal{O}}(\text{MM}(n, n, n/\Lambda^{1/(h-2)}) + n^2/\varepsilon^3)$ time in total. \blacktriangleleft

4 Implementing the Black Box **Count-Heavy** $_{\varepsilon}$

In this section, we implement **Count-Heavy** $_{\varepsilon}$. We prove the second part of Lemma 8, stated next.

► **Theorem 9.** *There is an algorithm that implements the **Count-Heavy** $_{\varepsilon}(G, S, a, b)$ black box, when \mathbb{H} is an h -cycles, for $h = \mathcal{O}(1)$, in time $\tilde{\mathcal{O}}(n^2 \cdot \frac{b}{a \cdot \varepsilon})$.*

For this entire section, the graph G is fixed, and the set S is a fixed subset of vertices, where for every $v \in S$ we have $t_G(v) \in [a, b]$. We emphasize that a is only a lower bound on $\min_{v \in S} t_G(v)$ and b is only an upper bound on $\max_{v \in S} t_G(v)$. Denote $N_k \triangleq |S|/k$. As explained in the introduction, it is insufficient to sample a few vertices from S , estimate $t_G(v)$ for each one, and apply a concentration bound to compute $t_G(S)$. Formally, this approach fails because $\sum_{v \in S} t_G(v) \neq t_G(S)$ due to possible double counting. We overcome this issue by sampling a small subset of vertices $S' \subseteq S$, and then, for every $v \in S'$ we approximate the number of copies of \mathbb{H} which intersect v and exactly i additional vertices from S for $0 \leq i \leq h-1$. This will allow us to estimate the number of multiple countings and therefore get an estimation of $t_G(S)$.

That is, the key ingredient of our approach for approximating \hat{t} as required by Count-Heavy $_{\varepsilon}$ is to approximate the number of cycles that intersect S in exactly k vertices. To this end, we define the following.

► **Definition 10.** Let \mathcal{C} denote the set of copies of \mathbf{H} in G . For $U \subseteq V$, define $\mathcal{C}(U) \triangleq \{C \in \mathcal{C} \mid C \cap U \neq \emptyset\}$. Denote $t \triangleq |\mathcal{C}|$ and $t(U) \triangleq |\mathcal{C}(U)|$. In general, we replace the symbol \mathcal{C} by t , to denote the cardinality of a set. Let \mathcal{C}^k denote the set of copies $C \in \mathcal{C}$ with $|C \cap S| = k$. Let $\mathcal{C}^k(v) \triangleq \mathcal{C}^k \cap \mathcal{C}(v)$. Let $t^k = |\mathcal{C}^k|$.

The following lemma shows that we can efficiently approximate t^k .

► **Lemma 11** (Algorithm Approx_{t^k}). *There exists a randomized algorithm Approx_{t^k} with the following characteristics. The input is a graph G , a set S , a precision parameter δ , a parameter $k \in [h]$, and a tuple (a, b) , such that for every $v \in S$ we have $t_G(v) \in [a, b]$. The algorithm produces an output \hat{t}^k which satisfies $\Pr[\hat{t}^k = t^k \pm t_G(S) \cdot \delta] \geq 1 - \frac{1}{n^4}$. The running time of the algorithm is bounded by $\tilde{O}(n^2 \cdot \frac{b}{a} \cdot \frac{1}{\delta})$.*

Approximating t^k directly leads to Theorem 9 because $\sum_{k \in [h]} t^k = t_G(S)$. A formal proof appears in the full version. To approximate t^k as required by Lemma 11, we find a value whose expectation is t^k and whose variance is at most $\mathcal{O}((N_k \cdot b)^2)$ (recall that $N_k = |S|/k$). We can do this efficiently, as follows.

► **Lemma 12** (Algorithm $\text{Approx}_{\mathbb{E}[t^k]}$). *There is a randomized algorithm $\text{Approx}_{\mathbb{E}[t^k]}$ whose input is G, S, δ and k . Note that unlike Approx_{t^k} , the algorithm $\text{Approx}_{\mathbb{E}[t^k]}$ does not require a and b as part of its input parameters. $\text{Approx}_{\mathbb{E}[t^k]}$ computes a value X such that $\mathbb{E}[X] = t^k$ and $\text{Var}[X] \leq C \cdot (N_k \cdot b)^2$, where C is a constant. The running time of the algorithm is bounded by $\tilde{O}(n^2)$.*

The reason that Lemma 12 is helpful is that we can run the algorithm it provides r times and take the median of means of these invocations. A formal proof of Lemma 11 appears in the full version. To get a sample X with $\mathbb{E}[X] = t^k$ and $\text{Var}[X] \leq C \cdot (b \cdot N_k)^2$ as needed by Lemma 12, we find samples Y_v with $\mathbb{E}[Y_v] = t^k(v)$ and $\text{Var}[Y_v] \leq C \cdot b^2$. We can do this efficiently, as follows.

► **Lemma 13.** *There is a randomized algorithm $\text{Approx}_{\mathbb{E}[\text{vertex-}t^k]}$ whose input is G, S, δ, k and a vertex $v \in S$. Unlike $\text{Approx}_{\mathbb{E}[t^k]}$, $\text{Approx}_{\mathbb{E}[\text{vertex-}t^k]}$ additionally takes a vertex $v \in S$ as input. $\text{Approx}_{\mathbb{E}[\text{vertex-}t^k]}$ computes a value Y_v such that $\mathbb{E}[Y_v] = t^k(v)$ and $\text{Var}[Y_v] \leq C \cdot b^2$, where C is a constant. The running time of the algorithm is bounded by $\tilde{O}(n^2)$.*

The reason that Lemma 13 is helpful is that we can sample a vertex v uniformly at random from S , and get, using Lemma 13, an unbiased estimator for the number of copies of \mathbf{H} which contains v , and intersect S exactly k times, i.e., $t^k(v)$. By the law of total expectation, we get that the expected value of this quantity, is equal to $\frac{1}{|S|} \sum_{u \in S} \mathbb{E}[t^k(u)] = t^k/N_k$. Therefore, we get an unbiased estimator for t^k up to a known value N_k . A formal proof of Lemma 12 appears in the full version.

To prove Lemma 13, we utilize the color-coding technique introduced by [2]. The high level approach of the technique is to randomly color vertices with h colors and detect *colorful* h -cycles that are ordered by, say, increasing colors. This additional structure allows for faster detection, at the cost of some probability of missing h -cycles that are colored out of order, which is overcome by repeated experiments.

A pertinent question arises: why are the algorithms $\text{Approx}_{\mathbb{E}[t^k]}$, $\text{Approx}_{\mathbb{E}[\text{vertex-}t^k]}$ necessary? Why not just choose a random coloring, compute the number of colorful copies of \mathbb{H} intersecting a set S exactly k times, and apply Chebyshev's inequality to conclude that repeating this process $\tilde{O}(n^2 \cdot \frac{b}{a\delta})$ times suffices for a good approximation of t^k ? The answer lies in the execution time of the matrix multiplication algorithm for counting colorful copies, which is dominated by the sizes of the largest, second largest, and smallest color classes. Roughly speaking, the smaller the product of these sizes, the faster the algorithm runs. Under random coloring, color classes each have a size of $\Omega(n)$ with high probability. Conversely, $\text{Approx}_{\mathbb{E}[\text{vertex-}t^k]}$ produces a color class containing just a single vertex, which significantly improves the running time in the worst case, compared to the approach which does not use the algorithms $\text{Approx}_{\mathbb{E}[t^k]}$, $\text{Approx}_{\mathbb{E}[\text{vertex-}t^k]}$. We need the following definitions to explain how color-coding works, and how the algorithm $\text{Approx}_{\mathbb{E}[\text{vertex-}t^k]}$ uses rectangular matrix multiplication.

► **Definition 14.** Fix some coloring $\varphi : V \rightarrow [\ell]$ for some $\ell \in \mathbb{N}$ (ℓ will usually be h). Let \mathcal{C}_φ denote the set of all copies $C \in \mathcal{C}$, such that $\varphi(C) = [\ell]$. If $C \in \mathcal{C}_\varphi$, we say that C is φ -colorful. Also define $\mathcal{C}_\varphi(v) = \mathcal{C}_\varphi \cap \mathcal{C}(v)$, $t_\varphi \triangleq |\mathcal{C}_\varphi|$, and $t_\varphi(v) \triangleq |\mathcal{C}_\varphi(v)|$. Let $\mathcal{C}_\varphi^k \triangleq \mathcal{C}^k \cap \mathcal{C}_\varphi$. That is, \mathcal{C}_φ^k is the set of copies of \mathbb{H} in G , where each such copy is colorful w.r.t. φ , and additionally intersects the set S exactly k times. Let $t_\varphi^k = |\mathcal{C}_\varphi^k|$.

Let A, B be two finite sets. We say that a function $\varphi : A \rightarrow B$ is a random coloring, if the value of each $a \in A$ is set to some value $b \in B$, where b is chosen uniformly at random from B and independently of values chosen for other elements in A .

Let $\varphi : V \rightarrow [h]$. For $i \in [h]$, we denote by $\varphi^{-1}(i)$ the set of all vertices v with $\varphi(v) = i$, and call this set the i -th color class. Assume without loss of generality that the color classes are sorted according their cardinalities, in a non-decreasing order. That is, for every $i < h$ we have $|\varphi^{-1}(i)| \geq |\varphi^{-1}(i+1)|$.

The last part of the above definition is used to quantify the complexity of computing t_φ^k as a function of the sizes of the color classes that the coloring φ induces, as follows.

► **Lemma 15.** Let $(\varphi_1, \varphi_2, \varphi_h)$ denote the cardinality of the largest, second largest, and smallest color classes, respectively. For any fixed $k \in [h]$, there is a deterministic algorithm for computing $\{t_\varphi^k(v)\}_{v \in V}$ in time $\mathcal{O}((h!)^2 \cdot h^2 \cdot \text{MM}(\varphi_1, \varphi_2, \varphi_h))$.

Next, we explain how to implement the algorithm $\text{Approx}_{\mathbb{E}[\text{vertex-}t^k]}$ given that we can compute the number t_φ^k of colorful copies of \mathbb{H} . The algorithm works as follows. It colors each vertex with a random color from the set $[h-1]$. It then recolors the input vertex by a new color h . Let φ denote this coloring. The algorithm then computes t_φ^k and outputs t_φ^k/q for some constant q such that $\mathbb{E}[t_\varphi/q] = t^k(v)$. We prove in the full version that the expectation and variance of this output satisfy the claimed requirements.

We are left with proving Lemma 15, which is the final step in the implementation of the algorithm $\text{Approx}_{\mathbb{E}[\text{vertex-}t^k]}$. To prove it, we reduce the problem of computing t_φ^k to the problem of computing t_φ on an auxiliary graph, in which every h -cycle is colorful and also intersects the set S exactly k times. We construct the auxiliary graph by randomly coloring the vertices with h colors, then selecting k color classes and keeping only vertices from S in them, while discarding the rest of the vertices in those classes. For the remaining $h-k$ color classes, we retain only vertices that are not part of S . We get a graph in which each color class is either contained in S or disjoint from S . This reduces the problem of computing t_φ^k on the auxiliary graph, to computing t_φ on it. The next claim addresses the running time of computing t_φ (on the auxiliary graph) instead of computing t_φ^k , and is the final missing piece for the proof of Theorem 9.

▷ **Claim 16.** Let G be a graph and let $\sigma \triangleq (U_1, \dots, U_h)$ be an (ordered) sequence of disjoint subsets of vertices of $V(G)$. Let t_G^σ denote the number of copies $C \in \mathcal{C}_G$ where $C = (v_1, v_2, \dots, v_h)$ and $v_i \in U_i$ for every $i \in [h]$. Let $U_{(1)}, U_{(2)}, U_{(h)}$ denote the cardinality of the largest, second largest, and smallest subset, respectively. Then, there is a deterministic algorithm that outputs $\{t_G^\sigma(v)\}_{v \in V}$ in time $\mathcal{O}(h^2 \cdot \text{MM}(U_{(1)}, U_{(2)}, U_{(h)}))$.

Proof of Claim 16. We assume without loss of generality that $|U_1| = U_{(h)}$, i.e., that U_1 is the smallest set. We first explain how to compute $\{t_G^\sigma(v)\}_{v \in U_1}$, and then we generalize this for U_j for any $j \in [h]$.

Let A denote the adjacency matrix of G . For $X, Y \subseteq V(G)$ let $A[X, Y]$ denote the submatrix containing all rows v for $v \in X$ and all columns u for $u \in Y$. Combinatorially, define a new directed graph H' with vertex set $X \cup Y$, and a directed edge (x, y) between a pair of vertices $x \in X$ and $y \in Y$ if and only if (x, y) is an edge in G . Note that $A[X, Y]$ is exactly the adjacency matrix of the new graph H' . Define $B_0 \triangleq I_{|U_1|}$, and for $0 \leq i < h$, define $A_i^\sigma \triangleq A[U_i, U_{i+1}]$ and $B_i^\sigma \triangleq B_{i-1} \cdot A_i$. We compute B_i^σ for $0 \leq i < h$. Note that $B_i^\sigma[x, y]$ denotes the number of paths with i edges between a vertex $x \in U_1$, and a vertex $y \in U_{i+1}$, which are of the form $(x, u_2, u_3, \dots, u_{i-1}, y)$ where $u_j \in U_j$ for $2 \leq j < i$. Let $A_h \triangleq A[U_h, U_1]$. After computing B_h , we compute $M = B_h^\sigma \cdot A_h^\sigma$ and return all entries on the diagonal of M . Note that for $v \in U_1$, we have that $M[v, v] = t_G^\sigma(v)$.

The generalization to other values $j \in [h]$ has only a small modification and appears in the full version.

Running Time. In the i -th iteration, for $1 \leq i \leq h$, we compute the product of the matrices B_{i-1} with the matrix A_i . Let a', b' denote the dimensions of A_i . The dimensions of B_{i-1} are $U_{(h)}, a'$, and therefore the running time is $\text{MM}(a', b', U_{(h)})$. Without loss of generality, we can assume $b' \leq a'$, because $\text{MM}(a', b', X) = \text{MM}(X, b', a')$ for any a', b', X . We also have $a' \leq U_{(1)}$ and $b' \leq U_{(2)}$. This bounds the time for the i -th iteration by $\mathcal{O}(h^2 \cdot \text{MM}(a', b', U_{(h)})) \leq \mathcal{O}(h^2 \cdot \text{MM}(U_{(1)}, U_{(2)}, U_{(h)}))$, which proves the claim. ◁

5 Implementing the Find-Heavy Black Box

In this section, we prove the first part of Lemma 8, as is specified in the following theorem.

► **Theorem 17.** *There is an algorithm that implements the Find-Heavy(G, Λ) black box when H is an h -cycle with $h = \mathcal{O}(1)$, in time $\tilde{\mathcal{O}}(\text{MM}(n, n, n/\Lambda^{1/(h-2)}))$.*

An algorithm for Theorem 17 is given a graph G and a heaviness threshold Λ , and needs to output a superset of the Λ -heavy vertices, which contains no $\Lambda/(\log n)^{h^2}$ -light vertices. Our algorithm works as follows. The algorithm selects a vector $P = (p_1, \dots, p_h)$, where $p_i \in [0, 1]$ for $i \in [h]$, which we explain shortly how to select. The algorithm then samples a uniform coloring φ for the vertices, and keeps each vertex of the i -th color class with probability p_i . We emphasize that not all vertices are kept with the same probability. Let F denote the obtained graph. If v is in at least one φ -colorful cycle in F , we say that v is P -discovered. We can find all P -discovered vertices over F using Lemma 15. For every vertex $v \in V$, let $v[P]$ denote the probability that v is P -discovered. The randomness is taken over the choice of the coloring and the sampling of vertices. We call this experiment the P -discovery experiment. We repeat this P -discovery experiment k times. If v is P -discovered more than $k\tau$ times, where τ is a threshold we set later, then P adds v into the set of heavy vertices. In this case, we say that v is P -added to the set of heavy vertices.

37:16 Fast Approximate Counting of Cycles

The final step of our algorithm is choosing a vector P , or rather a set of vectors \mathcal{P} , and then performing the P -discovery experiment with each vector in the set k times. Before specifying how we should choose \mathcal{P} , k , and τ , we state the properties we hope to achieve.

(1) Each vector $P \in \mathcal{P}$ induces a graph F , such that invoking Lemma 15 for computing the set of P -discovered vertices, takes $\tilde{\mathcal{O}}(\text{MM}(n, n, n/\Lambda^{1/(h-2)}))$ time. (2) Each vertex with $t_G(v) \geq \Lambda$ has at least one $P \in \mathcal{P}$ that P -adds v to the set of heavy vertices, w.h.p. (3) Any vertex v with $t_G(v) \leq \Lambda/(\log n)^{h^2}$ is w.h.p. not P -added to the heavy vertex set for any $P \in \mathcal{P}$. (4) The set \mathcal{P} has only $\tilde{\mathcal{O}}(1)$ vectors, allowing us to avoid repeating this experiment too many times.

The set of all vectors we will use is as follows. We take a (finite) subset of the all vectors $(p_1, \dots, p_h) \in [0, 1]^h$ which satisfy $\prod_{i \in [h]} p_i \leq \tilde{\mathcal{O}}(1/\Lambda)$. That is, all such vectors for which $p_i \in \{2^{-j} \mid 0 \leq j \leq \log(\Lambda) + 1\}$ for $i \in [h]$. We denote this set of vectors by $\text{Product}_h(\Lambda)$. Note that $|\text{Product}_h(\Lambda)| \leq (\log(\Lambda) + 2)^h = \tilde{\mathcal{O}}(1)$, where the last inequality follows because $\Lambda \leq n^h$, since no vertex in G participates in more than n^h copies of any h -vertex graph (that is, if the input was $\Lambda > n^h$, the algorithm could simply output an empty set). This means that $\log(\Lambda) \leq h \log n = \tilde{\mathcal{O}}(1)$. This proves Property (4) above.

The rest of the section proves that this set satisfies Properties (1)–(3). We first prove the first property, stating that for each $P \in \text{Product}_h(\Lambda)$, the P -discovery experiment can be implemented in the desired time.

► **Lemma 18.** *Let G be a graph with n vertices and let Λ be some positive number. Let $P = (p_1, \dots, p_h)$ be a vector in $[0, 1]^h$ with $\prod_{i=1}^h p_i = \tilde{\mathcal{O}}(\frac{1}{\Lambda})$. Let F be the (random) graph obtained in a P -discovery experiment. Then, we can find all the P -discovered vertices in time $\tilde{\mathcal{O}}(\text{MM}(n, n, n \cdot \Lambda^{-1/(h-2)}))$, w.h.p.*

Proof Sketch. The proof is included in the full version, and we provide the proof sketch here. Fix some $P \in [0, 1]^h$ where $P = (p_1, \dots, p_h)$, and $\prod_{i=1}^h p_i \leq 1/X$, for $X \geq 1$. Assume without loss of generality that $p_i \geq p_j$ for $i > j$. This implies that p_1 is the largest coordinate. Let F be the random graph, and let F_1, F_2, F_h denote the sizes of the first, second, and h color classes in F . First, note that we can use Lemma 15 to compute the set of discovered vertices in time $\mathcal{O}(\text{MM}(F_1, F_2, F_h))$. We analyze the running time of the algorithm specified in Lemma 15 on the random graph F . Using a standard Chernoff's inequality, we can get that $\mathcal{O}(\text{MM}(F_1, F_2, F_h)) = \tilde{\mathcal{O}}(\text{MM}(np_1, np_2, np_h))$ w.h.p. The crux of the algorithm is the following inequalities $\text{MM}(np_1, np_2, np_h) \leq \text{MM}(n, n, n \cdot (p_1 p_2 p_h)) \leq \text{MM}(n, n, n/X^{1/(h-2)})$, which completes the proof by setting $X \leftarrow \Lambda$. The first inequality is proved in the full version. The second inequality reduces to solving the following optimization task. Maximize $p_1 \cdot p_2 \cdot p_h$, under the constraints $p_i \geq p_{i+1}$, and $\prod_{i \in [h]} p_i = 1/X$. The proof is also in the full version, where we show that an optimal value is obtained when $p_1 = p_2 = 1$ and $p_i = X^{1/(h-2)}$, which completes the proof sketch. ◀

It remains to prove Properties (2) – (3). For this, we prove that each $\Lambda \cdot (2h \log(n))^{h+1}$ -heavy vertex v , has a vector $P \in \text{Product}_h(\Lambda)$ for which $v[P] = \Omega(1)$. On the other hand, for every $\Lambda/\log n$ -light vertex v , and any vector $P \in \text{Product}_h(\Lambda)$, we have $v[P] = \mathcal{O}(1/\log n)$, and therefore, we can distinguish between the two. The full details appear in the full version.

The upper bound on $v[P]$ for light vertex uses Markov's inequality. For the rest of this section we prove the statement on heavy vertices, by induction on h , for which the base case is $h = 3$. The intuition for the proof is as follows. Consider the base case of triangles. Let V_i denote the i -th color class. Consider finding the heavy vertices in the first color class. Consider all vectors $P_i = (1, 2^{-i}, 2^i/\Lambda)$, for $0 \leq i \leq \log(\Lambda) + 1$. These vectors form a subset of $\text{Product}_h(\Lambda)$. Fix some Λ -heavy vertex $v \in V_1$. We want to prove that for at least one

$i \in [\log(\Lambda)]$ we have $v[P_i] \geq \Omega(1)$. Our choice for the vectors is designed to deal with the following two extreme cases. The first case is that every $C \in \mathcal{C}_\varphi(v)$ intersects one specific vertex $u \in V_2$. For this case, the vector $P_0 = (1, 1, \frac{1}{\Lambda})$ is the right choice for discovering v because it maximizes the probability we hit u and a common neighbor of v and u , under the constraint that the sampling probability $p_1 \cdot p_2 \cdot p_3 \leq 1/\Lambda$. The second case is that among each of $V_2 \cap N(v)$ and $V_3 \cap N(v)$, there are $\sqrt{\Lambda}$ vertices that are connected as a complete bipartite graph (contained in $V_2 \times V_3$). For this case, the vector P_i for $i = \log(\Lambda)/2$ is the right choice for discovering v , because $P_i = (1, \frac{1}{\sqrt{\Lambda}}, \frac{1}{\sqrt{\Lambda}})$. For a larger h , the “hard” case is the following generalization of the above. From each color class V_i for $i \in \{3, \dots, h\}$ we take $k = \Lambda^{1/(h-2)}$ vertices and connect them by a complete $(h-2)$ -partite graph. We then take $v \in V_1$ and $u \in V_2$, we add an edge between v and u , and we connect v to all the aforementioned k vertices from V_h and we connect u to all the aforementioned k vertices from V_3 . The vector $(1, 1, \frac{1}{k}, \dots, \frac{1}{k})$ is the right choice for this case.

Roughly speaking, we show the set $\text{Product}_h(\Lambda)$ gradually shifts from handling one extreme case to another, and therefore “covers” all cases in between, which are the different ways to split vertices in h -cycles into h pieces whose product of sizes is Λ .

We need a final technical step before presenting the proof. First, we construct an h -partite graph which is easier to work with. For this, we sample uniform h coloring of the vertices of G . For $i \in [h]$ let V_i denote the set of vertices colored in the i -th color. For every $i \in [h]$ we keep only edges between the vertices of V_i and $V_{i+1 \bmod h}$, and direct those edges from V_i to V_{i+1} . Let G_φ denote the obtained directed graph. We emphasize that $t_{G_\varphi}(v) \leq t_\varphi(v)$, as the latter counts all colorful cycles in which v participates in, whereas the former counts only colorful cycles with edges between V_i and V_{i+1} in which v participates in. We prove that a heavy vertex will be P discovered in G_φ , with probability at least $\Omega(1)$, by some $P \in \text{Product}_h(\Lambda)$, whereas light vertices will only be discovered with probability $\mathcal{O}(1/\log n)$. In other words, since the gap between the heavy and light vertices is sufficiently large, we can still distinguish between the two in G_φ .

The rest of this section is dedicated to proving the following proposition.

► **Proposition 19.** *Fix a vertex v , and an h -coloring of the vertices of G . Suppose $t_{G_\varphi}(v) \geq \Lambda \cdot (2h \log(n))^{(h-1)^2}$. Then, there exists a vector $P \in \text{Product}_h(\Lambda)$ such that the probability that v gets P -discovered in G_φ is at least $(1 - 1/e)^{h-1}$.*

Due to space considerations, we prove here only the base case where the induction step is deferred to the full version.

Proof of Proposition 19. We prove this by induction on h . We start with the base case, that is, $h = 3$. We fix some h -coloring φ . Recall that we denote the i -th color class by V_i . Consider a vertex $v \in V_1$ with $t_{G_\varphi(\pi)}(v) \geq \Lambda \cdot (2h \log(n))^4$. To simplify the notation, we use $G' = G_\varphi$. We will prove that there exists a vector $P \in \text{Product}_h(\Lambda)$, such that the probability that the vertex v is P -discovered over G' is at least $(1 - 1/e)^2$. Let $K_0 = \log(\Lambda) + 1$. We consider a subset of vectors in $\text{Product}_h(\Lambda)$ of the form $P_i = (1, p_i, q_i)$, for $i \in \{0, 1, \dots, K_0\}$, where $p_i \triangleq 2^{-i}$, $q_i \triangleq \frac{2^i}{\Lambda}$. We partition V_2 into classes as follows. For $k \in \{1, 2, \dots, K_0\}$, let $Q_k = \{u \in V_2 \mid |\mathcal{C}_{G'}(v) \cap \mathcal{C}_{G'}(u)| \in [1/q_k, 2/q_k]\}$, $Q_0 = \{u \in V_2 \mid |\mathcal{C}_{G'}(v) \cap \mathcal{C}_{G'}(u)| \geq 1/q_0\}$. In words, $u \in Q_k$ if and only if the number of 3-cycles in G' that contain both v and u is at least $1/q_k$ and less than $2/q_k$. The set Q_0 consists of all vertices $u \in V_2$, such that the number of 3-cycles in G' that contain both v and u is at least $1/q_0 = \Lambda$. We claim that there exists $k \in \{0, 1, \dots, K_0\}$ such that $|Q_k| \geq 1/p_k$. The proof appears in the full version. We next show that for such k , we have $v[P_k] \geq (1 - 1/e)^{h-1}$ which completes the proof of the base case. Fix $k \in \{0, 1, \dots, K_0\}$ where $|Q_k| \geq 1/p_k$. For any non-empty subset $S \subseteq Q_k$,

let $\mathcal{E}_1(S)$ denote the event that $\{Q_k \cap V_2[p_k] = S\}$. That is, $\mathcal{E}_1(S)$ denote the event that during the P_k -discovery experiment, the set of vertices that were sampled from $V_2 \cap Q_k$ is exactly the set S . Let $R(S)$ denote the subset of vertices in V_3 which participate in a 3-cycle (in G') that contains v and some additional vertex from S . Let $\mathcal{E}_2(S)$ denote the event that $\{V_3[q_k] \cap R(S) \neq \emptyset\}$. That is, $\mathcal{E}_2(S)$ denotes the event that during the P_k -discovery experiment, the set of vertices that were sampled from $V_3 \cap R(S)$ is not empty. Next, we show that

$$v[P_k] \geq \sum_{S: \emptyset \subsetneq S \subseteq Q_k} \Pr[\mathcal{E}_1(S) \cap \mathcal{E}_2(S)] \geq (1 - 1/e)^2. \quad (1)$$

For every fixed $S \subseteq Q_k$, which is not empty, the events $\mathcal{E}_1(S)$ and $\mathcal{E}_2(S)$ are independent, since $\mathcal{E}_1(S)$ addresses sampling vertices from V_2 , while $\mathcal{E}_2(S)$ addresses sampling vertices from V_3 , where the two samples are independent of each other. Also note that $\mathcal{E}_1(S) \cap \mathcal{E}_2(S)$ is contained in the event that v is P_k -discovered, and since the events $\{\mathcal{E}_1(S)\}_{S \subseteq Q_k}$ are disjoint, so are the events $\{\mathcal{E}_1(S) \cap \mathcal{E}_2(S)\}_{S \subseteq Q_k}$. Therefore, the event that v is P_k -discovered, contains the union of the following disjoint events $\{\bigcup_{S: \emptyset \subsetneq S \subseteq Q_k} \mathcal{E}_1(S) \cap \mathcal{E}_2(S)\}$. We get

$$\begin{aligned} v[P_k] &\geq \Pr\left[\bigcup_{S: \emptyset \subsetneq S \subseteq Q_k} \mathcal{E}_1(S) \cap \mathcal{E}_2(S)\right] = \sum_{S: \emptyset \subsetneq S \subseteq Q_k} \Pr[\mathcal{E}_1(S) \cap \mathcal{E}_2(S)] \\ &= \sum_{S: \emptyset \subsetneq S \subseteq Q_k} \Pr[\mathcal{E}_1(S)] \Pr[\mathcal{E}_2(S)]. \end{aligned}$$

To complete the proof, we need to show that (1) $\sum_{S: \emptyset \subsetneq S \subseteq Q_k} \Pr[\mathcal{E}_1(S)] \geq 1 - 1/e$, and (2) that for every $S \subseteq Q_k$, which is not empty, we have $\Pr[\mathcal{E}_2(S)] \geq 1 - 1/e$. The first claim follows as

$$\sum_{S: \emptyset \subsetneq S \subseteq Q_k} \Pr[\mathcal{E}_1(S)] = \Pr[Q_k[p_k] \neq \emptyset] = 1 - (1 - p_k)^{|Q_k|} \geq 1 - 1/e,$$

where the inequalities hold for the following reasons. The first two equalities follows from definition, and the last inequality follows from the assumption that $|Q_k| \geq 1/p_k$.

The second claim follows as every non-empty subset $S \subseteq Q_k$ satisfies $|R(S)| \geq 1/q_k$. To see this, fix some vertex $u \in S$. We have $R(u) \subseteq R(S)$, and $|R(u)| \geq 1/q_k$ because $u \in Q_k$. Therefore, for any such S , we have

$$\Pr[\mathcal{E}_2(S)] = \Pr[R(S)[q_k] \neq \emptyset] = 1 - (1 - q_k)^{|R(S)|} \geq 1 - (1 - q_k)^{1/q_k} \geq 1 - 1/e.$$

This completes the proof of Equation (1). The rest of the proof appears in the full version. ◀

References

- 1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. *SIAM J. Comput.*, 47(6):2527–2555, 2018.
- 2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.
- 3 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.
- 4 Arturs Backurs and Christos Tzamos. Improving viterbi is hard: Better runtimes imply faster clique algorithms. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 311–321, 2017.
- 5 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 623–632. ACM/SIAM, 2002.

- 6 Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Trans. Knowl. Discov. Data*, 4(3):13:1–13:28, 2010.
- 7 Amartya Shankha Biswas, Talya Eden, Quanquan C. Liu, Ronitt Rubinfeld, and Slobodan Mitrovic. Massively parallel algorithms for small subgraph counting. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPICs*, pages 39:1–39:28. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.APPROX/RANDOM.2022.39.
- 8 Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2014. doi:10.1007/978-3-662-43948-7_19.
- 9 Karl Bringmann and Philip Wellnitz. Clique-based lower bounds for parsing tree-adjoint grammars. In *Proceedings of the 28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 12:1–12:14, 2017.
- 10 Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In Stijn Vansummeren, editor, *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 253–262. ACM, 2006.
- 11 Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. Sparse matrix multiplication and triangle listing in the congested clique model. *Theoretical Computer Science*, 809:45–60, 2020.
- 12 Keren Censor-Hillel, Dean Leitersdorf, and David Vulakh. Deterministic near-optimal distributed listing of cliques. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 271–280, 2022.
- 13 Yi-Jun Chang, Seth Pettie, Thatchaphol Saranurak, and Hengjie Zhang. Near-optimal distributed triangle enumeration via expander decompositions. *Journal of the ACM (JACM)*, 68(3):1–36, 2021.
- 14 Yi-Jun Chang and Thatchaphol Saranurak. Deterministic distributed expander decomposition and routing with applications in distributed derandomization. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 377–388. IEEE, 2020. doi:10.1109/FOCS46700.2020.00043.
- 15 Artur Czumaj and Christian Konrad. Detecting cliques in congest networks. *Distributed Computing*, 33(6):533–543, 2020.
- 16 Mina Dalirrooyfard, Surya Mathialagan, Virginia Vassilevska Williams, and Yinzhan Xu. Listing cliques from smaller cliques. *CoRR*, 2023. arXiv:2307.15871.
- 17 Mina Dalirrooyfard, Thuy-Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: Hardness for all induced patterns and faster noninduced cycles. *SIAM J. Comput.*, 50(5):1627–1662, 2021.
- 18 Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colorful decision oracle. *SIAM J. Comput.*, 51(4):849–899, 2022.
- 19 Danny Dolev, Christoph Lenzen, and Shir Peled. “tri, tri again”: finding triangles and small subgraphs in a distributed setting. In *Distributed Computing: 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings 26*, pages 195–209. Springer, 2012.
- 20 Talya Eden, Nimrod Fiat, Orr Fischer, Fabian Kuhn, and Rotem Oshman. Sublinear-time distributed algorithms for detecting small cliques and even cycles. *Distributed Computing*, pages 1–28, 2022.
- 21 Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.

- 22 Talya Eden, Dana Ron, and Will Rosenbaum. Almost optimal bounds for sublinear-time sampling of k -cliques in bounded arboricity graphs. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 56:1–56:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 23 Talya Eden, Dana Ron, and C. Seshadhri. On approximating the number of k -cliques in sublinear time. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 722–734. ACM, 2018.
- 24 Mira Gonen, Dana Ron, and Yuval Shavitt. Counting stars and other small subgraphs in sublinear-time. *SIAM J. Discret. Math.*, 25(3):1365–1411, 2011.
- 25 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 1–10, 1977.
- 26 Taisuke Izumi and François Le Gall. Triangle finding and listing in congest networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 381–389, 2017.
- 27 Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In Lusheng Wang, editor, *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 710–716. Springer, 2005.
- 28 Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 598–609. Springer, 2012.
- 29 Tamara G. Kolda, Ali Pinar, Todd D. Plantenga, C. Seshadhri, and Christine Task. Counting triangles in massive graphs with mapreduce. *SIAM J. Sci. Comput.*, 36(5), 2014.
- 30 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Comment. Math. Univ. Carol.*, 26(2):415–419, 1985.
- 31 Mihai Pătrăușcu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610. ACM, 2010.
- 32 Kanat Tangwongsan, A. Pavan, and Srikanta Tirthapura. Parallel triangle counting in massive streaming graphs. In Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi, editors, *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 – November 1, 2013*, pages 781–786. ACM, 2013.
- 33 Jakub Tětek. Approximate triangle counting via sampling and fast matrix multiplication. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 34 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.
- 35 Virginia Vassilevska Williams, Yinzhao Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *Proc. SODA*, page to appear, 2024.
- 36 Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM J. Discret. Math.*, 10(2):209–222, 1997.
- 37 Raphael Yuster and Uri Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 254–260. SIAM, 2004.