Faster Cycle Detection in the Congested Clique

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 provide a fast distributed algorithm for detecting h-cycles in the Congested Clique model, whose running time decreases as the number of h-cycles in the graph increases. In undirected graphs, constant-round algorithms are known for cycles of even length. Our algorithm greatly improves upon the state of the art for odd values of h. Moreover, our running time applies also to directed graphs, in which case the improvement is for all values of h. Further, our techniques allow us to obtain a triangle detection algorithm in the quantum variant of this model, which is faster than prior work.

A key technical contribution we develop to obtain our fast cycle detection algorithm is a new algorithm for computing the product of many pairs of small matrices in parallel, which may be of independent interest.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Distributed algorithms

Keywords and phrases triangle detection, cycle detection, distributed computing, Congested Clique, quantum computing, Fast matrix multiplication, Fast rectangular matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.12

Related Version *Full Version*: `https://www.arxiv.org/abs/2408.15132`

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 feedback.

1 Introduction

Finding small subgraph patterns is a fundamental computational task, with a multitude of applications for uncovering connections between elements in a data set. Research has been thriving, addressing the complexity of different variants of subgraph isomorphism for fixed size subgraph patterns H in a larger host graph G: detecting whether a copy of H exists, listing all of its copies, counting the number of occurrences, and more.

In this paper, we provide a fast distributed algorithm for detecting h-cycles in the Congested Clique model [37], in which n machines communicate by sending $\mathcal{O}(\log n)$-bit messages to each other, in synchronous rounds.

The pioneering work of [19] showed that all copies of any fixed h-vertex graph H in an n node graph can be listed in this model within $\mathcal{O}(n^{1-2/h})$ rounds. This result of course applies also to the detection variant. For the case when H is a cycle, [12] provided an h-cycle detection algorithm running in $2^{\mathcal{O}(h)}n^{\rho}$ rounds, for both undirected and directed graphs (henceforth digraphs). Here, ρ is the exponent of distributed fast matrix multiplication (FMM) in the Congested Clique model, i.e., the value such that $\mathcal{O}(n^{\rho})$ rounds are sufficient for multiplying two $n \times n$ matrices. The value of ρ is currently known to be at most $1 - 2/\omega$ where ω is the centralized fast matrix multiplication exponent, and since $\omega \leq 2.371552$ [2],

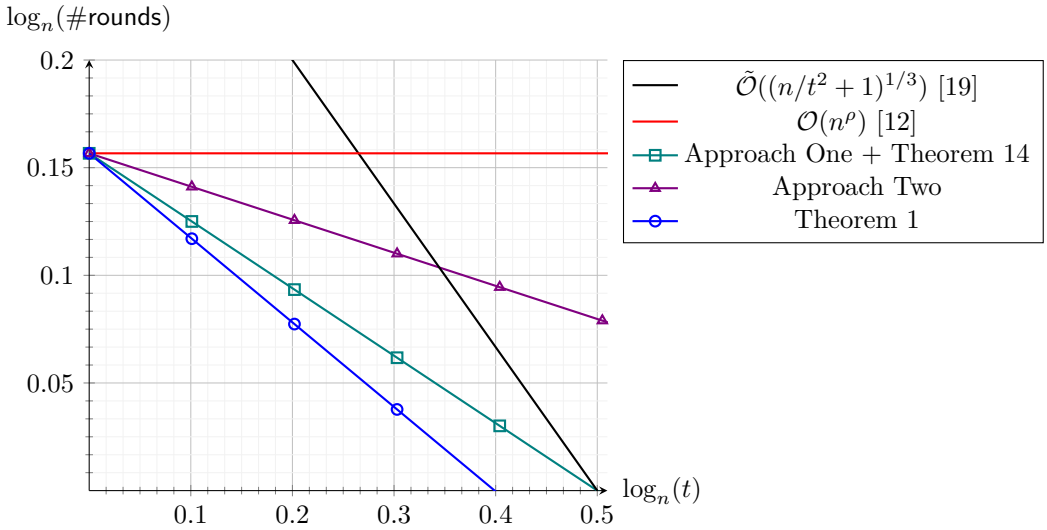

Figure 1 An illustrative comparison between our results and prior work, for the case of triangles. For each algorithm, we plot the base-n logarithm of the number of rounds as a function of the base-n logarithm of the number of triangles.

we get a bound for ρ of 0.15667. In the case of 4-cycles in undirected graphs, [12] obtained a constant-round detection algorithm, and this result was later generalized by [10] to hold for detection of any even-length cycle in undirected graphs.

This leaves the complexity of odd-cycle detection as an open question, as well as the detection of cycles of any length in digraphs. For triangles, [19] showed a detection algorithm that completes within $\tilde{\mathcal{O}}(n^{1/3}/(t^{2/3}+1))$ rounds, w.h.p.[1], where t is the number of triangles. Since [20] hints that lower bounds for H detection in this model are not within reach, it remains open whether the above is optimal.

Question: *For a given graph H, is there a faster H-detection algorithm when the number of instances of H in the input graph is large?*

We answer this question in the affirmative, providing a fast h-cycle detection algorithm whose complexity decreases as the number t of instances of H grows. Our algorithm has the same running time for detecting h-cycles in graphs as well as in digraphs. For triangles, the complexity of our algorithm greatly improves upon that of [19]. For larger odd cycles in graphs, as well as cycles of any length in digraphs, to the best of our knowledge, this is the first improvement over [12].

An important insight of our main technical contribution is to identify a new refined parameter as a key player for detection: the number of vertices x that participate in an h-cycle. Below, we elaborate on our result and technical approach.

1.1 Our Contributions and Technical Overview

To frame our technical contributions, we first briefly overview the two previous approaches for the case of *triangles*. In [19], an $\tilde{\mathcal{O}}(n^{1/3}/(t^{2/3}+1))$-round algorithm is presented, where t is the number of triangles. The algorithm samples n induced subgraphs, and for each sample

[1] High probability in this paper refers to a probability that is at least $1-1/n^c$ for some constant $c \geq 1$.

it checks for a triangle by letting a dedicated vertex collect the edges of the sample. In [12], a $2^{\mathcal{O}(h)}n^{\rho}$-round algorithm is presented which employs fast matrix multiplication over the entire graph.

Warm-up. As a warm-up, consider the following combination of these approaches to get the best of both worlds, leading to an algorithm that completes in $\tilde{\mathcal{O}}(n^{\rho}/(t^{2\rho}+1))$ rounds, which is already an improvement over the prior state of the art (recall that $\rho < 1/3$). To obtain this, we sample only t^2 induced subgraphs to which each vertex is added with probability $1/t$. Using the second moment method, we can show that at least one sampled subgraph contains a triangle with probability $\Omega(1)$. To check if each subgraph contains a triangle we use matrix multiplication, and so no vertex has to collect the edges of an entire sample. To compute the product of t^2 square matrices of size n/t, we develop a new algorithm, which computes the product of s pairs of square matrices of size k in $\mathcal{O}(n^{\rho-2} \cdot k^2 \cdot s^{1-\rho})$ rounds. This algorithm may be of independent interest.

Another natural approach to consider is one that samples a subset of vertices and checks whether any of these vertices participates in a triangle, rather than attempting to sample a complete triangle. Here, we can sample each vertex with probability $1/t^{1/3}$ and check whether it is in a triangle by invoking rectangular matrix multiplication. While this too improves upon the state of the art for some graphs, it is always slower than our first approach. Note that trying to reduce the running time by sampling with a probability that is smaller than $1/t^{1/3}$ would reach a dead-end since it is not likely to hit any vertex in a triangle in case all t triangles are induced by a clique of $t^{1/3}$ vertices. See Figure 1 for a comparison of the two approaches, as well as the previous algorithms, and our new algorithm.

A caveat is that the first approach does not extend for h-cycles, as the number of samples it needs to perform increases with h. For example, for C_4 detection, if we sample uniformly random induced subgraph with n/t vertices, it contains a copy of C_4 with probability at least roughly $1/t^3$ which means we have to sample at least t^3 subgraphs to ensure that we find a copy of C_4 with a constant probability. This is slower than the previous best known algorithm of [12] which takes $2^{\mathcal{O}(h)}n^{\rho}$ rounds. In other words, the first approach is slower as h is larger.

Our contribution. Our key insight is that we can further boost these two approaches such that they complement each other, in the following sense. For a fixed value of t, the first approach is better when the number of vertices that participate in a triangle, which we denote by x, is small, while the second approach is better when x is large. This refinement of considering the parameter x along with t allows us to bring these two approaches a big leap forward by obtaining a faster algorithm for triangle detection, as well as an algorithm for h-cycle detection for longer cycles, in both graphs and digraphs.

Our first algorithm, which we refer to as Find-Cycle, follows the first approach. It samples $s = x^3/t$ subsets of vertices $(U_1, \ldots, U_s)$, by adding each vertex to U_i independently with probability $1/x$. The algorithm then checks for every $i \in [s]$ if $G[U_i]$ contains a triangle. This involves computing the product of s pairs of square matrices of size n/x each, which we do in $\tilde{\mathcal{O}}(n^{\rho-2} \cdot (n/x)^2 \cdot s^{1-\rho}) = \tilde{\mathcal{O}}(n^{\rho} \cdot (1/x)^2 \cdot (x^3/t)^{1-\rho})$ rounds. Using the second moment method (which is very similar to Chebyshev's inequality) we show that at least one of the s induced subgraphs contains a triangle with a constant probability.

Our second algorithm, which we refer to as Find-Vertex-In-Cycle, follows the second approach. It samples a subset of vertices S, by adding each vertex to S independently with probability $1/x$. The algorithm then checks if one of the vertices from S participates in a

triangle by computing the product of a rectangular matrix of size $n/x \times n$ and a square matrix of size n. Interestingly, the algorithm Find-Vertex-In-Cycle also achieves the same round complexity, as a function of x, for h-cycle detection, for $h = \mathcal{O}(1)$.

Our final algorithm alternates between the two algorithms until one of them terminates. Among all n vertices with t triangles, the final algorithm is the slowest when the two algorithms have the same round complexity, which happens when $x^{3-1.82408} = \Theta(t)$.

The following states the running time of our fast h-cycle detection algorithm, and is proven in Appendix A.

▶ **Theorem 1** (h-Cycle Detection). *Let G be a (directed) graph with t copies of h-cycles. There is a randomized Congested Clique algorithm for h-cycle detection, which takes $\tilde{\mathcal{O}}(h^{\mathcal{O}(h)} \cdot n^{0.1567}/(t^{\frac{0.4617}{h-1.82408}} + 1))$ rounds w.h.p.*

Here, the constants $0.1567, 0.4617$ arise from the complexity of rectangular multiplication. We are able to show that the product of a rectangular matrix of size $k \times n$ and a square matrix can be computed in $O(n^{0.1567}/k^{0.4617})$ rounds, using the formula of [27] and an adaptation of the code of [5]. To get a flavor of the above complexity, note that a crucial implication of Theorem 1 is that we detect a triangle in $\tilde{\mathcal{O}}(1)$ rounds for graphs with at least $t = \Omega(n^{0.3992})$ triangles, improving upon the previously known threshold of $t = \Omega(n^{1/2})$ from [19].

Many Matrix Multiplications in Parallel. To implement our Find-Cycle algorithm, we need to compute the product of many small random square matrices, which are submatrices of the adjacency matrix of the input graph. We state this informally in the following theorem.

▶ **Theorem 2** (Informal). *Let k, s be two integers such that $k \in [\sqrt{n}, n]$, and $s \leq (n/k)^2$. Then, in the Congested Clique model, the n vertices can compute the product of s pairs of square matrices of size k in $\mathcal{O}(n^{\rho-2} \cdot k^2 \cdot s^{1-\rho})$ rounds, given that the input is distributed among the vertices in a "balanced" manner.*

The formal definitions and the proof of Theorem 2 appear in Section 3, as well as the definitions and claims we need for the proof of Theorem 1.

The conceptual contribution of Theorem 2 is as follows. On one hand, it is known that n vertices can compute the product of n pairs of matrices of size $\sqrt{n}$ in a constant number of rounds, given that the input is balanced, by letting the i-th vertex collect all the entries of the i-th pair of matrices and computing their product. On the other hand, n vertices can also compute the product of one square matrix of size n in $\mathcal{O}(n^\rho)$ rounds, as shown in [12]. Theorem 2 gives a smooth trade-off between these two extremes.

Note that [27] provides an algorithm for computing the product of s pairs of square matrices of size n in $\mathcal{O}(n^\rho \cdot s^{1-\rho})$ rounds for $s \leq n$. The paper also provides an algorithm for computing the product of s pairs of *rectangular* matrices of sizes $n \times m$ and $m \times n$, where the bound on the round complexity is more involved and is not given by an analytic expression, see Section 3.3 for a discussion. Moreover, we need to compute the product of square matrices of size smaller than n, which is not covered by the above algorithm.

Before providing intuition about our proof of Theorem 2, we explain what we mean by a balanced input and how the output should be distributed. Given a set of s pairs of square matrices $\mathcal{Q} = \{(S_i, T_i)\}_{i \in [s]}$ of size k each, where $sk^2 \leq n^2$, we think of the input as a "flat" array of sk^2 entries. The input is distributed as follows. The input given to the first vertex is the first n entries in this array. The second vertex gets the next n entries and so on. For the output $\{(P_i)\}_{i \in [s]}$, where $P_i = S_i \cdot T_i$ for $i \in [s]$, we again transform the set of output matrices into a flat array and let each vertex learn distinct consecutive n entries from it. Note

that as long as each vertex holds unique n entries from the input, and every vertex knows which entries from the input every other vertex holds, then the input can be redistributed in $\mathcal{O}(1)$ rounds, using Lemma 8. We therefore define a balanced input as such.

▶ **Definition 3** (Balanced Input). *An input for n vertices is* balanced *if it is partitioned between the vertices such that each vertex holds at most n (unique) entries from the input, and every vertex knows which entries from the input are held by every other vertex.*

Now we can give the intuition behind the proof of Theorem 2. We partition the n vertices into s sets of size n/s each. For every $i \in [s]$ we call the i-th set in the partition the i-th *team*. The i-th team is responsible for computing the product (S_i, T_i) (this partitioning method is similar to that of [27]). After partitioning, the problem boils down to computing one product of square matrices of size k using n/s vertices with bandwidth of size $s \log n$, which we solve by extending the work of [12], which considers only the product of square matrices of size equal to the number of vertices.

Recall that our main motivation for this tool of Theorem 2 is to implement the algorithm Find-Cycle. That is, given a graph G with n vertices, we sample s subsets of vertices $(U_1, \ldots, U_s)$, where each vertex joins each set independently with probability p. Each set U_i defines an induced subgraph $G[U_i]$ with an adjacency matrix A_i. We need to compute $(A_i)^h$ for every $i \in [s]$, and we denote the set $\mathcal{Q} = \{(A_i, A_i)\}_{i \in [s]}$ as the input, where we assume $s \leq 1/p^2$. In order to implement the algorithm Find-Cycle using Theorem 2, we need to show that $\mathcal{Q}$ is a balanced input. However, this does not precisely hold, but we can show a sufficient guarantee of $\mathcal{Q}$ being "almost" balanced w.h.p., in which the requirements in Definition 3 are weakened. These weaker conditions still allow us to quickly redistributed the elements into a balanced input in $\mathcal{O}(\log n)$ rounds w.h.p.

To conclude, we obtain a fast algorithm for h-cycle detection, which beats the previous state-of-the art for odd-length cycles, as well as directed cycles. The algorithm is faster as the number of copies of h-cycles increases, where the key parameter for the algorithm and the analysis is the number of vertices in the graph that participate in an h-cycle.

Triangle Detection in the Quantum Congested Clique Model. Our new matrix multiplication tool turns out to be helpful for additional tasks. In the quantum setting, we obtain the following in the Quantum Congested Clique model, which is similar to the Congested Clique model, but the vertices exchange messages of $\mathcal{O}(\log n)$ qubits in each round instead of standard bits.

▶ **Theorem 4.** *There exists a Quantum Congested Clique $\tilde{\mathcal{O}}((n/(t^2+1))^{3\rho/4})$-round algorithm for triangle detection, with a success probability of at least $1/2$.*

Due to space considerations, the proof of Theorem 4 is deferred to the full version of the paper. Theorem 4 obtains a $\tilde{\mathcal{O}}(n^{3\rho/4})$-round algorithm that remains effective even when $t = 0$. This is faster than the previous state-of-the-art algorithm from [12], which takes $\mathcal{O}(n^\rho)$ rounds and does not leverage the additional capabilities that the Quantum Congested Clique model offers. For subgraphs other than triangles, there are detection algorithms in the Quantum Congested Clique which use the extra power of the model. For example, [9] provides an algorithm for larger clique detection. Moreover, in the quantum Congest model, there are algorithms for clique and cycle detection [32, 26], which are faster than their Congest counterparts.

Our algorithm uses a Grover search [30], which is a quantum algorithm to find an element in an unsorted list of size L, while accessing only $\sqrt{|L|}$ entries from the list. Generally, given a function $f : X \to \{0, 1\}$ and a universe X, Grover search is a quantum algorithm which

finds an $x \in X$ such that $f(x) = 1$ (assuming such x exists), by querying f at most $\sqrt{|X|}$ times w.h.p. In [35] a distributed implementation of Grover search was provided, for the quantum Congest model, which was later extended to the Quantum Congested Clique model in [31, 9].

We provide an overview of our algorithm, which has two steps. In the first step, we sample $s = 1/t^2$ random induced subgraphs $(U_1, \ldots, U_s)$, where each vertex joins every set independently with probability $1/t$. We partition the vertices into s sets, each of n/s vertices, which we call teams. For $i \in [s]$, the i-th team uses Grover search to detect a triangle in the subgraph $G[U_i]$, as follows. It samples $\ell = 8 \log n/q^3$ subsets of vertices $(W_1, \ldots, W_\ell)$, where each vertex from U_i joins each set independently with probability $1/q$. The universe for the search is the set $X \triangleq \{G[W_i]\}_{i \in [\ell]}$, and the boolean function g is defined as $g(G[W_i]) = 1$ if $G[W_i]$ contains a triangle, and 0 otherwise.

The correctness of the algorithm follows because if the graph G has t triangles, then there exists an index $i \in [s]$ such that $G[U_i]$ contains a triangle with probability at least $1/10$. The i-th team finds this triangle using $\sqrt{1/q^3}$ evaluations of g w.h.p. The final detail of the algorithm is to set q such that each evaluation of g takes $\mathcal{O}(1)$ rounds, which optimizes the round complexity of this approach.

Our above fast triangle detection algorithm for the Quantum Congested Clique model is given in the full version of the paper, as well as a discussion on why extending our approach to h-cycle detection is not straightforward.

Additional Related Work. In this Congested Clique model, matrix multiplication was first studied by [20]. After the aforementioned works of [12, 27], the work of [13] showed an algorithm whose running time improves with the sparsity of the input matrices, and [8] showed algorithms for sparse matrix multiplication which also enjoy the sparsity of the output or when only a sparse piece of the output is needed.

The task of listing subgraphs has also received great attention in the Congested Clique model. Here, each vertex needs to output a list of copies of the subgraph H, such that the union of the lists is exactly the set of all copies of H in the graph. As mentioned, [19] give an algorithm for listing all h-vertex graphs within $O(n^{1-2/h})$ rounds. For triangles, this is known to be tight by [32, 38]. This optimality extends to larger cliques due to [25]. Listing triangles in sparse graphs can be done faster, as first shown by [38] with a randomized algorithm, and then followed up by [13] with a deterministic algorithm. Afterward, [11] showed faster sparse listing for larger cliques, which also has a deterministic algorithm due to [10].

We mention that in the closely-related Congest model, in which the communication graph is the input graph itself, rather than being a complete network, the state of affairs is in stark contrast to the Congested Clique model. For listing cliques, optimal algorithms are known due to [16, 7], with deterministic solutions in [17, 14, 15]. The underlying approach, initiated by [16], is to construct an expander decomposition, which partitions the vertices into components with good expansion (low mixing time). At a very high level, the vertices of each component list cliques for which some edges are inside the component, and then recurse over the remaining edges. However, for an algorithmic approach of the Congested Clique model to have a fast implementation in the Congest model, also when using the known routing procedures [28, 29], the algorithm has to adhere to certain conditions. In other words, it is not the case that any algorithm in the Congested Clique model can be executed efficiently by the components of the expander decomposition in the Congest model.

Specifically, we stress that for the detection variant in Congest, the state of the art even just for triangles is the same as for listing. That is, even the $O(n^\rho)$ algorithm of [12] does not have an implementation in the Congest model, and it is unknown how to detect triangles

in less than the time it takes for listing them (interestingly, the only lower bounds that are known are that a single round does not suffice [1, 25]). For larger h-cycles, detection for odd values of h is known to have a complexity of $\tilde{\Theta}(n)$ [21]. Much work is invest in studying the complexity of detecting even cycles [21, 34, 10, 22, 9, 39, 26], with the state of the art being a recent result showing that h-cycles can be detected in $\tilde{O}(n^{1-2/h})$ rounds for even values of h [26].

Investigations into the subgraph detection problem have also been conducted in additional models such as the quantum Congest and Quantum Congested Clique models, where vertices exchange qubits instead of standard bits. In [33], a quantum Congest $\tilde{\mathcal{O}}(n^{1/4})$-round algorithm for triangle detection was presented, which outperforms the $\tilde{\mathcal{O}}(n^{1/3})$-round Congest algorithm. This approach was further improved in [9] by developing an $\tilde{\mathcal{O}}(n^{1/5})$ rounds quantum algorithm. Additional upper and lower bounds for cycle detection in the quantum Congest model were presented in [39, 26]. In [9], an Quantum Congested Clique algorithm for p-clique detection, for $p \geq 4$, was presented, which achieves an $\tilde{\mathcal{O}}(n^{1-2/(p-1)})$-round complexity, which is faster than the classical Congested Clique algorithm. Further research in the quantum distributed models includes both upper and lower bounds for various problems [23, 31, 39].

There is extensive research about subgraph finding in additional models of distributed computing. All of these important works are a bit more far from our work here, and hence we refer the reader to the survey of [6], which contains a recent overview of subgraph finding algorithms for distributed settings.

2 Preliminaries

We use $[n]$ to denote the set $\{1, \ldots, n\}$. We denote the base graph by G, its vertices by $V(G)$, where unless stated otherwise we assume $V(G) = [n]$. Fix some constant integer $h \geq 0$. Let t denote the number of h-cycles in G, and let $V_{\mathcal{C}_h}(G)$ denote the set of vertices that participate in an h-cycle in G, where we denote by x the size of the set $V_{\mathcal{C}_h}(G)$. This parameter plays a crucial role in the analysis in Appendix A. We establish a connection between the two parameters x and t as follows.

▶ **Definition 5** (δ). *For undirected graph, G with t copies of h-cycle, and $V_{\mathcal{C}_h}(G) = x$, we define δ as such that $x^{h-\delta} = 2h \cdot t$. If G is directed, we define δ as such that $x^{h-\delta} = h \cdot t$.*

We present two claims on δ. We show that $\delta \in [0, h-1]$, and that the term $x^{-\delta}$ is equal to is the probability for h vertices sampled uniformly at random with replacement from $V_{\mathcal{C}_h}(G)$ to form an h-cycle in G.

▷ Claim 6. Let G be a graph with t copies of an h-cycle and x vertices that participate in at least one h-cycle. Sample h vertices $(v_1, \ldots, v_h)$ from $V_{\mathcal{C}_h}(G)$ uniformly at random with replacement from $V_{\mathcal{C}_h}(G)$. Then the probability that they form an h-cycle is exactly $x^{-\delta}$.

▷ Claim 7. It holds that $\delta \in [0, h-1]$.

The proofs of Claims 6 and 7 are deferred to the full version of the paper.

2.1 Additional Tools

▶ **Lemma 8** (Lenzen's Routing Lemma [36]). *The following is equivalent to the Congested Clique model: In every round, each vertex can send (receive) $\{b_i\}_{i\in[n]}$ bits to (from) the i-th vertex, for any sequence $\{b_i\}_{i\in[n]}$ satisfying $\sum_{i=1}^{n} b_i = \mathcal{O}(n \log n)$. In other words, any routing scheme in which no vertex sends or receives more than $\mathcal{O}(n)$ messages can be preformed in $\mathcal{O}(1)$ rounds.*

▶ **Theorem 9** (Chernoff Bound [18, Corollary 1.10.6.]). *Let $X_1, \ldots, X_n$ be independent random variables taking values in $[0,1]$ and $X = \sum_i X_i$. Let $\delta \in [0,1]$. Then,* $\Pr\left[|X - \mathbb{E}[X]| \geq \delta \mathbb{E}[X]|\right] \leq 2\exp\left(-\delta^2 \cdot \mathbb{E}[X]/3\right)$

▶ **Theorem 10** (Reverse Markov's inequality [18, (1.6.4)]). *Let X be a random variable with support contained in $[0, M]$. Then, for $R \in \mathbb{R}$, we have* $\Pr[X > R] \geq \frac{\mathbb{E}[X]-R}{M-R}$.

▶ **Theorem 11** (FMM-based triangle detection [12]). *There is a deterministic algorithm for triangle detection, which takes $\mathcal{O}(n^\rho)$ rounds.*

3 Fast Matrix Multiplication in Congested Clique

3.1 Preliminaries and Balanced Products

In this section, we define the problem of computing s pairs of square matrices of size k each, as well as defining what is a balance input. We assume throughout the paper that the matrices are over a field $\mathbb{F}$, where each element can be represented using $\mathcal{O}(\log n)$ bits. Due to space constraints, we the proofs of this section are omitted, and can be found in the full version of the paper. We introduce the following definitions to specify the required input.

▶ **Definition 12** (The notations $A[i,*]$ and $A[*x*,*y*]$). *Let A be some matrix. We denote the i-th row of A by $A[i,*]$. For a matrix A of dimension $n \times n$ and two indices $x, y \in [\sqrt{k}]$ for $k \leq n$, we also use $A[*x*,*y*]$ to denote a matrix of dimension $n/\sqrt{k} \times n/\sqrt{k}$, which is the following submatrix of A. For every index $v \in [n]$, we split it into three indices $v = v_1v_2v_3$ where $v_1, v_3 \in [n^{1/2} \cdot k^{-1/4}]$, $v_2 \in [\sqrt{k}]$. The expression $*x*$ then refers to all v for which $v_2 = x$.*

The following definition formally defines the problem of multiple matrix multiplications.

▶ **Definition 13** ($\mathsf{Product}(\mathcal{Q})$). *Given set of s pairs of square matrices $\mathcal{Q} = \{(S_i, T_i)\}_{i\in[s]}$ of size $k \times k$. In the $\mathsf{Product}(\mathcal{Q})$ problem, n nodes need to compute the products of those pairs of matrices. The input is distributed as follows. Each vertex $v \in V$ is assigned a label $\ell(v) = (i, x, y)$, where $i \in [s]$, and $x, y \in [\sqrt{n/s}]$. The vertex v gets as input the submatrix $S_i[*x*,*y*], T_i[*x*,*y*]$, and has to learn the entries of the submatrix $P_i[*x*,*y*]$, where $P_i = S_i \cdot T_i$ for $i \in [s]$. We denote the round complexity of this problem by $\mathsf{MM}(k,k,k;s)$.*

Note that every vertex can learn the label of each other vertex in $\mathcal{O}(1)$ rounds. The following theorem is the main theorem for this section, in which we provide an upper bound for the round complexity of the $\mathsf{Product}(\mathcal{Q})$ problem.

▶ **Theorem 14.** *For any two integers k, s where $k \in [\sqrt{n}, n]$ and $s \leq (n/k)^2$, we have that*

$$\mathsf{MM}(k,k,k;s) = \mathcal{O}(n^{\rho-2} \cdot k^2 \cdot s^{1-\rho}).$$

To prove the theorem, we first partition the n nodes into s sets of size n/s each. For every $i \in [s]$ we call the i-th set in the partition the i-th *team*. The i-th team is responsible for computing the i-th product in $\mathcal{Q}$, i.e., the product (S_i, T_i). After partitioning into teams, the problem boils down to computing one product of square matrices of size k using n/s node with bandwidth of size $s \log n$. This extends [12], in which only the product of square matrices of size equal to the number of vertices is considered, and uses [27], in which multiple products are divided into teams. A crucial step in the algorithm for Theorem 14 is to compute the product of a single matrix of size R using n' nodes, which we define next.

▶ **Definition 15** (Single-Product (n', R)). *Let H be a team with n' vertices. Let S, T be two square matrices of dimension R for some $R \in [1, (n')^2]$, and define $P = ST$. Each vertex $v \in H$ is assigned a label $\ell'(v) = xy$, where $x, y \in [\sqrt{n'}]$. The input of each vertex $v \in H$ with label $\ell'(v) = xy$ is $S[*x*, *y*]$ and $T[*x*, *y*]$, and its output should be $P[*x*, *y*]$. We denote this problem by* Product (n', R).

▶ **Proposition 16.** *The* Single-Product (n', R) *problem can be solved in the Congested Clique model with n' vertices in the base graph and bandwidth B, in $\mathcal{O}((n')^\rho \cdot (R/n')^2 \cdot \frac{F}{B})$ rounds, where each entry in R can be represented using $\mathcal{O}(F)$ bits. Using bandwidth B means that in each round, each vertex in the base graph can send B bits to every other vertex.*

3.2 Multiple Products of Random Submatrices

In this subsection, we explain how to use the tools we developed in the previous subsection, to detect an h-cycle in s induced subgraphs sampled uniformly at random. Specifically, we explain how to compute the h-th power of the adjacency matrices of those subgraphs.

Let $\mathcal{U} = (U_1, \ldots, U_s)$ be a set of subsets of vertices, where each subset is a uniformly random set. That is, each vertex joins to the set U_i independently and uniformly at random, with probability p. For each $i \in [s]$, we denote the the adjacency matrix of $G[U_i]$ by A_i, and define $\mathcal{Q} = \{(A_i, A_i)\}_{i \in [s]}$.

We explain how to compute the h-th power of $\{A_i\}_{i \in [s]}$ in parallel by reducing this problem into the Product$(\mathcal{Q})$ problem. In other words, we explain how to redistribute the initial input, into an input for the Product$(\mathcal{Q})$ problem. We show that the reduction takes $\mathcal{O}(\log n)$ rounds (Proposition 20) w.h.p., and provide an algorithm that tests whether the reduction algorithm can be executed in $\mathcal{O}(\log n)$ or not (Claim 23). The testing algorithm takes $\mathcal{O}(1)$ rounds. In case the testing algorithm indicates that we sampled a set $\mathcal{U}$ for which the reduction takes more than $\mathcal{O}(\log n)$ rounds, we discard the current sample set $\mathcal{U}$, and sample a new set. We also prove that w.h.p. we will not have to discard the sampled set (Claim 22).

In Section 3.1, we described an algorithm to compute the product of s pairs of matrices of size k each. Here, we describe an algorithm to compute the product of $1/p^a$ pairs of matrices of size *at most* $4np$ each. The connection between the parameters s, k, p and a is as follows. We set $k = 4np$, and $s = 1/p^a$ where $a \in [0, 2]$. We get that $sk^2 \leq n^2$ as desired. We provide a definition for a set $\mathcal{U}$ for which we can redistribute the input in $\mathcal{O}(\log n)$ rounds. We call such a set a *p-balanced* set.

▶ **Definition 17** (p-Balanced Set). *Given is a parameter p. Let $\mathcal{U}$ be a set of subsets of vertices from $V(G)$. Let a be a constant for which $|\mathcal{U}| = p^{-a}$. We say that $\mathcal{U}$ is a* p-balanced set *if all the following conditions hold:*

1. *$a \in [0, 2]$ (so $|\mathcal{U}| \leq (1/p)^2$).*
2. *$n^{-1/2} \leq p \leq 1$.*
3. *Every vertex $v \in V(G)$ belongs to at most $\lceil |\mathcal{U}| \cdot p \rceil 4 \log n = \lceil p^{1-a} \rceil 4 \log n$ sets in $\mathcal{U}$.*
4. *Every set $U \in \mathcal{U}$ is of size at most $\lceil 4np \rceil$.*

Note that (1) and (2) imply that $|\mathcal{U}| \leq n$.

The next claim proves that if $\mathcal{U} = (U_1, \ldots, U_{p^{-a}})$ is a p-balanced set, then every vertex v can learn the IDs of all vertices in U_j for each set U_j to which v belongs.

▷ Claim 18. Let $\mathcal{U} = (U_1, \ldots, U_{p^{-a}})$ be a p-balanced set, where every vertex knows to which U_j it belongs. There is an $\mathcal{O}(\log n)$-round Congested Clique algorithm that allows each vertex to learn the IDs of all vertices in U_j for each set U_j to which v belongs.

In what follows, we explain how to route the input of a p-balanced set $\mathcal{U}$, after each vertex learns the IDs of all vertices in U_j for each set U_j to which v belongs, to match the input of the Product($\mathcal{Q}$) problem. This routing takes $\mathcal{O}(\log n)$ rounds. Before providing a routing algorithm, we introduce new notation that we need in order to explain how the input is routed.

▶ **Definition 19** (The notation $A[i, U_j]$)**.** *Recall that $V = [n]$, and let $U_j \subset V$, and let i be some vertex in U_j. Let A_j be the corresponding adjacency matrix of U_j. For vertex i in the set U_j we define $A[i, U_j]$ as the submatrix of A, which contains only the i-th row, and all k columns, for $k \in U_j$.*

▶ **Proposition 20** (Redistributing the Input)**.** *Given a parameter p, let $\mathcal{U} \triangleq (U_1, \ldots, U_{p^{-a}})$ be a set of subsets of vertices from $V(G)$ which is a p-balanced set. For each $i \in [p^{-a}]$, let A_i be the adjacency matrix of the induced graph $G[U_i]$. Label each vertex $v \in [n]$ as $\ell(v) \triangleq (x, y, i) \in [\sqrt{np^a}] \times [\sqrt{np^a}] \times [p^{-a}]$. Partition the vertices into p^{-a} teams, each of size $n \cdot p^a$, where the j-th team contains all vertices v with label $\ell(v) = (x, y, i)$ such that $i = j$. Then, in parallel, each vertex v with label $\ell(v) = (x, y, i)$ can learn $A_i[*x*, *y*]$ in $\mathcal{O}(\log n)$ rounds.*

The next corollary address the detection of an h-cycle in one of the sampled graphs. It follows from Proposition 20 and Theorem 14. The algorithmic aspects of this corollary are presented in Appendix A.

▶ **Corollary 21.** *Given r and p, let $\mathcal{U} = (U_1, \ldots, U_r)$ be a set of subsets of vertices from $V(G)$, where $\mathcal{U}$ is a p-balanced set, and every vertex in G knows whether it belongs to U_j for every $j \in [s]$. For each $i \in [p^{-a}]$, let A_i be the adjacency matrix of the induced graph $G[U_i]$. Label each vertex $v \in [n]$ as $\ell(v) \triangleq (x, y, i) \in [\sqrt{np^a}] \times [\sqrt{np^a}] \times [p^{-a}]$. Then, for every integer h, in parallel, each vertex v with label $\ell(v) = (x, y, i)$ can learn $(A_i)^h[*x*, *y*]$ in $\mathcal{O}(\log(h) \cdot n^\rho \cdot p^{2+a(\rho-1)} + \log n)$ rounds.*

The remainder of this subsection shows that a set $\mathcal{U}$ of uniformly random subsets of vertices is a p-balanced set w.h.p. We also provide an algorithm to test whether a set of subsets of vertices is a p-balanced set in a constant number of rounds. We create $s \triangleq p^{-a}$ subsets of vertices, by letting each vertex join each set independently with probability p (each vertex knows p and a). We denote the sets by $\mathcal{U} = (U_1, \ldots, U_s)$, and show that $\mathcal{U}$ is p-balanced w.h.p., and that the vertices can determine whether this is the case in $\mathcal{O}(1)$ rounds. If $\mathcal{U}$ is indeed p-balanced then in $\mathcal{O}(\log n)$ rounds each vertex can learn the IDs of all vertices in each set U_j to which it belongs. This is

▷ Claim 22. $\mathcal{U}$ is balanced with probability at least $1 - 2/n^3$.

▷ Claim 23. There is a Congested Clique algorithm that decides if $\mathcal{U}$ is p-balanced in $\mathcal{O}(1)$ rounds.

3.3 Rectangular Matrices

Here we set the ground for computing the product of two rectangular matrices in Congested Clique. We build on the work of [27], which shows that computing the product of two rectangular matrices S, T of size $n \times n^{\beta_0}$ and $n^{\beta_0} \times n$ respectively takes $\mathcal{O}(n^{o(1)})$ rounds.

▶ **Definition 24** (Rectangular matrix multiplication $\mathsf{RM}(S,T)$)**.** *Given two matrices S,T of dimension $n \times n^z$ and $n^z \times n$ where $z \in [0,1]$, the $\mathsf{RM}(S,T)$ problem is to compute $P = S \cdot T$ in the Congested Clique model with n nodes. The input of each vertex $i \in [n]$ is $S[i,*]$ and $T[*,i]$, and its output should be $P[i,*]$. We abuse the notation and use it also to denote the complexity of the problem by $\mathsf{MM}(n,n,n^z)$ or $\mathsf{RM}(n^z)$.*

▶ Remark 25. For two matrices S,T of size $n^z \times n$ and $n \times n$ (or $n \times n$ and $n \times n^z$), we get the same round complexity [24, Theorem 6]. In this case, we assume the input is of each vertex $i \in [n]$ is $S[*,i]$ and $T[*,i]$, and its output should be $P[i,*]$.

▶ **Definition 26** (The exponent of matrix multiplication)**.** *The exponent of the* sequential *complexity of computing the product of two matrices of dimensions $n \times n^z$ and $n^z \times n$ respectively is denoted by $\omega(z)$. We denote by $\mathcal{O}(n^{\rho(z)})$ the round complexity of computing this product in the Congested Clique model. Let $\alpha_0 = \lim_{\varepsilon \to 0} \sup\{z \mid \omega(z) \leq 2+\varepsilon\}$, and $\beta_0 = \lim_{\varepsilon \to 0} \sup\{z \mid \rho(z) = \varepsilon\}$. Then $\alpha_0 \geq 0.321334$ [40] and $\beta_0 \geq (1+\alpha_0)/2 \geq 0.660667$ [27, 40].*

We would like to upper bound the function $\rho(z)$ by some analytic function, which is easy to work with. To do so, we use the following notation.

▶ **Definition 27** (The notation $\mathtt{B}$, $\mathtt{A}$)**.** *We will use $\mathtt{B},\mathtt{A}$ for two real non-negative constants such that, for every $y \in [0, 1-\beta_0]$ we have $\rho(1-y) \leq \mathtt{B} - \mathtt{A}y$.*

We give two explicit linear functions which bound the function $\rho(1-y)$. First, if the function $\rho(z)$ is convex, then we can set $\mathtt{A} = \rho(1)/(1-\beta_0)$ and $\mathtt{B} = \rho(1)$, by taking the line passing through the points $(\beta_0, \rho(\beta_0))$ and $(1, \rho(1))$. This is of course the "best" (minimizing l_∞ norm) linear function that upper bounds the function $\rho(1-y)$. Yet, proving that $\rho(z)$ is convex is beyond the scope of this paper. Instead, the following claim is an additional explicit linear function we provide, which does not assume that $\rho(z)$ is convex.

▷ Claim 28. We can set $\mathtt{A} = 0.4617$ and $\mathtt{B} = 0.1567$.

The proof of Claim 28 (appears in the full version) is numeric: We build a step function which is always above $\rho(z)$, and then find a line which is above the step function in the desired range. For any choice of $\mathtt{B},\mathtt{A}$ that fits Definition 27 and any $p \in (0,1)$ we have $\mathsf{RM}(np) = \mathcal{O}(n^{\rho(1-\log_n(1/p))}) \leq \mathcal{O}(n^{\mathtt{B}-\mathtt{A}\cdot\log_n(1/p)}) = \mathcal{O}(n^{\mathtt{B}} \cdot p^{\mathtt{A}})$. Thus, by the above discussion and by Claim 28 we get the following.

▶ **Conclusion 1.** *For $p \in (0,1)$ we have $\mathsf{RM}(np) \leq \mathcal{O}(n^{0.1567} \cdot p^{0.4617})$. If $\rho(z)$ is convex, we have $\mathsf{RM}(np) \leq \mathcal{O}(n^{\rho} \cdot p^{\rho/(1-\beta_0)})$.*

References

1 Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Christoph Lenzen. Fooling views: a new lower bound technique for distributed computations under congestion. *Distributed Comput.*, 33(6):545–559, 2020. doi:10.1007/S00446-020-00373-4.

2 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication, 2024. arXiv:2404.16349, doi:10.48550/arXiv.2404.16349.

3 Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2016.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

5 Jan van den Brand. Complexity term balancer. www.ocf.berkeley.edu/~vdbrand/complexity/. Tool to balance complexity terms depending on fast matrix multiplication.

6 Keren Censor-Hillel. Distributed subgraph finding: Progress and challenges (invited talk). In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 3:1–3:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. Up-to-date version on Arxiv, https://doi.org/10.48550/arXiv.2203.06597, 2021. doi:10.4230/LIPIcs.ICALP.2021.3.

7 Keren Censor-Hillel, Yi-Jun Chang, François Le Gall, and Dean Leitersdorf. Tight distributed listing of cliques. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2878–2891. SIAM, 2021. doi:10.1137/1.9781611976465.171.

8 Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. *Distributed Comput.*, 34(6):463–487, 2021. doi:10.1007/s00446-020-00380-5.

9 Keren Censor-Hillel, Orr Fischer, François Le Gall, Dean Leitersdorf, and Rotem Oshman. Quantum distributed algorithms for detection of cliques. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 35:1–35:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ITCS.2022.35.

10 Keren Censor-Hillel, Orr Fischer, Tzlil Gonen, François Le Gall, Dean Leitersdorf, and Rotem Oshman. Fast distributed algorithms for girth, cycles and small subgraphs. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 33:1–33:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.DISC.2020.33.

11 Keren Censor-Hillel, François Le Gall, and Dean Leitersdorf. On distributed listing of cliques. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 474–482, 2020. doi:10.1145/3382734.3405742.

12 Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. *Distributed Comput.*, 32(6):461–478, 2019. doi:10.1007/s00446-016-0270-2.

13 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. doi:10.1016/J.TCS.2019.11.006.

14 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. doi:10.1145/3519270.3538434.

15 Yi-Jun Chang, Shang-En Huang, and Hsin-Hao Su. Deterministic expander routing: Faster and more versatile. *arXiv preprint arXiv:2405.03908*, 2024. doi:10.48550/arXiv.2405.03908.

16 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. doi:10.1145/3446330.

17 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.

18 Benjamin Doerr and Frank Neumann, editors. *Theory of Evolutionary Computation - Recent Developments in Discrete Optimization*. Natural Computing Series. Springer, 2020. doi:10.1007/978-3-030-29414-4.

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 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In Magnús M. Halldórsson and Shlomi Dolev, editors, *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 367–376. ACM, 2014. doi:10.1145/2611462.2611493.

21 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, pages 367–376, 2014. doi:10.1145/2611462.2611493.

22 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.

23 Michael Elkin, Hartmut Klauck, Danupon Nanongkai, and Gopal Pandurangan. Can quantum communication speed up distributed computation? In Magnús M. Halldórsson and Shlomi Dolev, editors, *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 166–175. ACM, 2014. doi:10.1145/2611462.2611488.

24 Michael Elkin and Ofer Neiman. Centralized, parallel, and distributed multi-source shortest paths via hopsets and rectangular matrix multiplication. In *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2022.

25 Orr Fischer, Tzlil Gonen, Fabian Kuhn, and Rotem Oshman. Possibilities and impossibilities for distributed subgraph detection. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 153–162, 2018. doi:10.1145/3210377.3210401.

26 Pierre Fraigniaud, Mael Luce, Frederic Magniez, and Ioan Todinca. Even-cycle detection in the randomized and quantum congest model. *arXiv preprint arXiv:2402.12018*, 2024.

27 François Le Gall. Further algebraic algorithms in the congested clique model and applications to graph-theoretic problems. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 57–70. Springer, 2016. doi:10.1007/978-3-662-53426-7_5.

28 Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed mst and routing in almost mixing time. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 131–140, 2017. doi:10.1145/3087801.3087827.

29 Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In *32nd International Symposium on Distributed Computing*, 2018.

30 Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996. doi:10.1145/237814.237866.

31 Taisuke Izumi and François Le Gall. Quantum distributed algorithm for the all-pairs shortest path problem in the CONGEST-CLIQUE model. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 84–93. ACM, 2019. doi:10.1145/3293611.3331628.

32 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. doi:10.1145/3087801.3087811.

33 Taisuke Izumi, François Le Gall, and Frédéric Magniez. Quantum distributed algorithm for triangle finding in the congest model. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

34 Janne H. Korhonen and Joel Rybicki. Deterministic subgraph detection in broadcast CONGEST. In James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão, editors, *21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal,*

December 18-20, 2017, volume 95 of *LIPIcs*, pages 4:1–4:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.OPODIS.2017.4.

35 François Le Gall and Frédéric Magniez. Sublinear-time quantum computation of the diameter in congest networks. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2018. URL: https://dl.acm.org/citation.cfm?id=3212744.

36 Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 42–50, 2013. doi:10.1145/2484239.2501983.

37 Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in o (log log n) communication rounds. *SIAM journal on computing*, 35(1):120–131, 2006. doi:10.1137/S0097539704441848.

38 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. *ACM Transactions on Parallel Computing (TOPC)*, 8(2):1–28, 2021. doi:10.1145/3460900.

39 Joran van Apeldoorn and Tijn de Vos. A framework for distributed quantum queries in the CONGEST model. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 109–119. ACM, 2022. doi:10.1145/3519270.3538413.

40 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *Proc. SODA*, page to appear, 2024.

A h-Cycle Detection

In this section, we prove the following theorem.

▶ **Theorem 1** (h-Cycle Detection). *Let G be a (directed) graph with t copies of h-cycles. There is a randomized Congested Clique algorithm for h-cycle detection, which takes $\tilde{\mathcal{O}}(h^{\mathcal{O}(h)} \cdot n^{0.1567}/(t^{\frac{0.4617}{h-1.82408}}+1))$ rounds w.h.p.*

We do so by presenting two algorithms and analyzing their running time and success probability as a function of the parameters n, t, and x. Figure 2 depicts their running times. All the proofs, and implementation details are omitted due to space constraints and can be found in the full version of the paper.

Before presenting the algorithms, we overview the color-coding technique [4], which is a common method used to find paths or cycles of constant length h. To detect an h-cycle, first color the vertices of the graph using h colors, where each vertex is colored uniformly at random and independently of all other vertices. Then look for a *colorful* h-cycle, which is an h-cycle with exactly one vertex of each color. This provides additional structure, which a detection algorithm can benefit from. However, not every coloring induces a colorful h-cycle, which means that to detect an h-cycle, we might have to repeat this experiment multiple times, until we sample a coloring that induces a colorful h-cycle.

In more detail, given a graph G with n vertices, we sample a uniformly random coloring $\varphi : V \to [h]$, which means that φ colors each vertex uniformly independently at random. We then build the auxiliary graph G_φ which is a directed graph, as follows.

▶ **Definition 29** (G_φ). *Given a graph G, and a coloring $\varphi : V \to [h]$, we define a new directed graph G_φ on the same vertex set, with a set of directed edges $E(G_\varphi) \triangleq \{(u,v) \in E(G) \mid \varphi(v) = (\varphi(u)+1) \mod h\}$. That is, only a subset of the edges is kept, and it consists of the edges from the vertices of color i to the vertices of color $(i+1) \mod h$, for $i \in [h]$.*

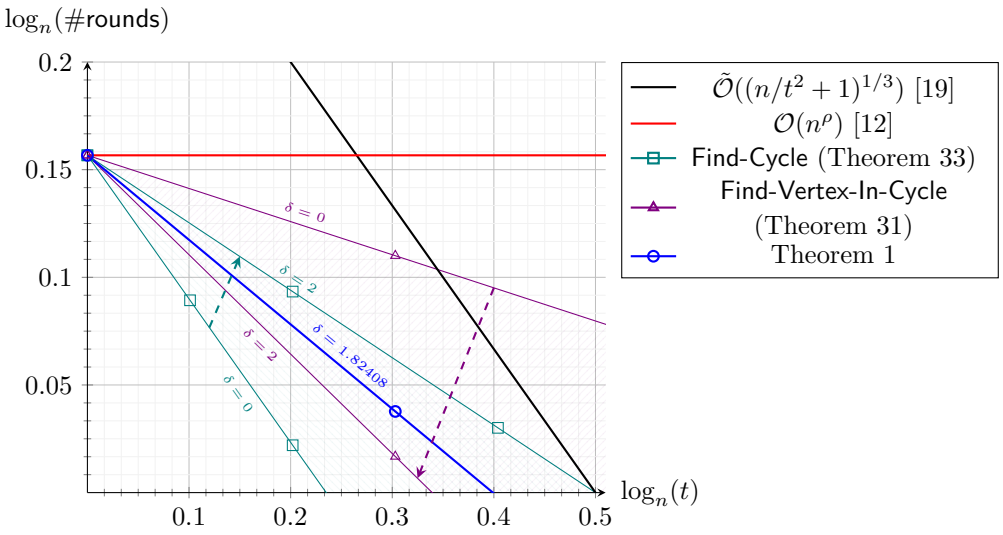

Figure 2 An illustrative comparison between our results and prior work, for the case of triangles. For each algorithm, we plot the base-n logarithm of the number of rounds as a function of the base-n logarithm of the number of triangles. An additional axis represents the value of δ ranging from 0 to 2. For a fixed t, Find-Cycle performs faster as δ decreases, with its round complexity depicted by the area shaded in teal. Conversely, Find-Vertex-In-Cycle performs better as δ increases, and its round complexity is shown by the area shaded in violet.

The graph G_φ has the property that every walk of length smaller than h is a simple path, and every closed walk of length h is a cycle. Here, a *walk* of length h on a (directed) graph is a sequence of vertices $(v_1, v_2, \ldots, v_{h+1})$ not necessarily distinct, such that for $i \in [h]$ we have that (v_i, v_{i+1}) is an edge in G. We say that a walk is a simple path if all the vertices in the walk are distinct. A walk $(v_1, v_2, \ldots, v_h, v_{h+1})$ is *closed* if $v_1 = v_{h+1}$.

We explain how we benefit from the property that every closed walk of length h in G_φ is a cycle. Let A_φ denote the adjacency matrix of G_φ. We can compute the h-th power of the matrix A_φ, and look at the diagonal of the obtained matrix. Then, G_φ is h-cycle free if and only if all the entries on the diagonal are equal to 0. Clearly, if G does not contain an h-cycle, then for any coloring φ, we have that G_φ does not contain an h-cycle. The more interesting property of this random coloring is that if G contains an h-cycle, then the probability that G_φ contains one is at least $1/h^h$, as we prove next.

▷ Claim 30. Let G be a graph with at least one h-cycle. Let $\varphi : V \to [h]$ be some uniformly random coloring. Then G_φ contains an h-cycle with probability at least $\frac{1}{h^h}$.

A.1 The Algorithm Find-Vertex-In-Cycle

We explain how to detect an h-cycle in time $\mathcal{O}(\mathsf{MM}(n, n, n/x)\cdot\log^2 n)$ w.h.p., with a one-sided error, as stated in the next theorem.

▶ **Theorem 31.** *There exists a randomized Congested Clique algorithm to detect an h-cycle in time $\tilde{\mathcal{O}}(\mathsf{MM}\left(n, n, \frac{n}{x}\right))$ w.h.p., with a one-sided error.*

Let G be a graph with n vertices and t copies of an h-cycle, for a fixed constant h. For a graph H, we denote by $V_{\mathcal{C}_h}(H)$ the set of vertices that participate in an h-cycle in H. Let $x = |V_{\mathcal{C}_h}(G)|$. We prove Theorem 31 by analyzing the following random process.

Find-Vertex-In-Cycle. The input of the algorithm is a graph G and some value $p \in [0,1]$. The output is "True" if at least one h-cycle is detected, and "False" otherwise. The algorithm works as follows. The algorithm samples a coloring $\varphi : V(G) \to [h]$, uniformly at random, and uses it to define a new auxiliary graph G_φ, as explained in Definition 29. Let V_i denote the set of vertices in G_φ that are assigned the color i, for $i \in [h]$. The algorithm then samples a subset of vertices from V_1, by sampling each vertex independently with probability p. Let U_1 denote the set obtained. Define F_φ as the induced subgraph of G_φ with the vertex set $U_1 \cup \bigcup_{i=2}^{h} V_i$. Let A_{F_φ} denote the adjacency matrix of the graph F_φ. Next, the algorithm exactly counts the number of h-cycles in F_φ using rectangular matrix multiplication. That is, it computes the trace of the h-th power of A_{F_φ}, and outputs "True" if it is not zero, and "False" otherwise. Clearly, this can be computed by first computing the h-th power of A_{F_φ}, and then computing its trace, which takes $\mathcal{O}(\mathsf{MM}(n,n,n))$ rounds. However, a faster well-known way to compute this trace, without computing the h-th power of A_{F_φ}, is as follows. Compute the following product:

$$A_{F_\varphi}[U_1, V_2] \cdot A_{F_\varphi}[V_2, V_3] \cdots A_{F_\varphi}[V_{h-1}, V_h] \cdot A_{F_\varphi}[V_h, U_1] \ ,$$

where for $S, T \subseteq V$ the matrix $A_{F_\varphi}[S,T]$ denotes the rectangular matrix with $|S|$ rows and $|T|$ columns, every for every $s \in S$ and $T \in t$ we have that $(A_{F_\varphi}[S,T])_{s,t} = 1$ if $(s,t) \in E(F_\varphi)$ and 0 otherwise. This matrix is also called the biadjacency matrix. The order in which the multiplications are computed affects the round complexity. The algorithm computes this product by sequentially multiplying a rectangular matrix of size at most $4np \times n$ and a matrix of size at most $n \times n$, to get a new matrix of size $4np \times n$. In other words, the algorithm first computes the product $A_{F_\varphi}[U_1, V_2] \cdot A_{F_\varphi}[V_2, V_3]$, to obtain some matrix B_2, and then computes the product $B_2 \cdot A_{F_\varphi}[V_3, V_4]$. In this way, the algorithm does not multiply two square matrices of size n, and can benefit from the fact that it only computes the product of one smaller rectangular matrix with a square one. This completes the description of the algorithm.

Clearly, the algorithm never outputs "True" if the graph G is h-cycle free. In what follows, we give a lower bound on the probability that it outputs "True" when the graph has h-cycles.

▷ Claim 32. If the sampling probability of vertices from V_1 into U_1 satisfies $p \geq \frac{4h^h}{x}$, then the algorithm outputs "True" with probability at least $\frac{1}{4h^h}$.

The implementation of the algorithm in the Congested Clique model appears in the full version of the paper.

A.2 The Algorithm Find-Cycle

In the next two subsections, we explain how to prove the following theorem.

▶ **Theorem 33.** *There exists a randomized Congested Clique algorithm to detect an h-cycle in time $\tilde{\mathcal{O}}(\mathsf{MM}\left(\frac{n}{x}, \frac{n}{x}, \frac{n}{x}; x^\delta\right))$ w.h.p., with one-sided error.*

Recall that G is a graph with n vertices and t copies of an h-cycle for $h = \mathcal{O}(1)$. For a graph H, we denote by $V_{\mathcal{C}_h}(H)$ the set of vertices that participates in an h-cycle in H. Let $x = |V_{\mathcal{C}_h}(G)|$. We also use δ for the solution for $x^{h-\delta} = 2ht$ satisfies that $\delta \in [0, h-1]$.

▶ Remark 34. Recall that $\mathsf{MM}\left(\frac{n}{x}, \frac{n}{x}, \frac{n}{x}; x^\delta\right) = \mathcal{O}(n^\rho \cdot x^{-(2+\delta(\rho-1))})$, by Corollary 21.

We prove Theorem 33 by analyzing the following random process.

Find-Cycle. The input of the algorithm is a graph G A graph G, a value $p \in [0,1]$, and a value $a \in [0,2]$. The output is "True" if at least one h-cycle is detected, and "False" otherwise. The algorithm works as follows.

1. Sample uniformly at random a coloring $\varphi : V \to [h]$.
2. Sample $r \leftarrow 8(4h)^{h+2} \cdot p^{-a}$ subsets of vertices $\mathcal{U} = (U_1, \ldots, U_r)$, where each vertex joins U_i independently with probability p for $i \in [r]$.
3. For every $U \in \mathcal{U}$, define two graphs. The first one is the induced graph $F = G[U]$, and the second one is the colored directed graph F_φ, obtained from applying φ on F. Denote the adjacency matrix of F_φ by M_U.
4. For $U \in \mathcal{U}$, compute the trace of the h-th power of the matrix M_U, and output "True" if for at least one set U, this trace is not zero. Otherwise, output "False".

Fix some set $U \in \mathcal{U}$, and a random coloring φ, and let $F = G[U]$, and $F_\varphi = (G[U])_\varphi$. We prove that for $p \geq 1/x$, the subgraph F_φ contains an h-cycle with probability $\Omega\left(x^{-\delta}\right)$. For that, it suffices to prove that if $p \geq 1/x$ then F contains an h-cycle with probability at least $\frac{1}{x^\delta \cdot (4h)^{h+1}}$: We proved in Claim 30 that if F contains an h-cycle then F_φ contains an h-cycle with probability at least $\frac{1}{h^h}$. In the next proposition, we prove that the subgraph F contains an h-cycle with probability at least $\frac{1}{x^\delta \cdot (4h)^{h+2}}$, if $p \geq \frac{1}{x}$.

▶ **Proposition 35.** *If $p \geq \frac{1}{x}$, then F contains an h-cycle with probability at least $\frac{1}{x^\delta \cdot (4h)^{h+2}}$.*

To prove the above, we use the second moment method [3, Theorem 4.3.1]. The proof of the proposition, as well as the implementation of the algorithm in the Congested Clique model, is deferred to the full version of the paper.

A.3 Wrap-Up: Fast Cycle Detection

In this subsection, we wrap up to prove our fast algorithm for h-cycle detection, when $h = \mathcal{O}(1)$, in both undirected and directed graphs. Our algorithm is the fastest for odd cycle detection when the number of cycles is super polylogarithmic, and for h-cycle detection in directed graphs, when the number of h-cycles is super polylogarithmic. For graphs with small t, our algorithm has the same running time as the fastest algorithm for multiplying two matrices of size $n \times n$, and our running time is never worse than it up to polylogarithmic factors.

▶ **Theorem 1** (h-Cycle Detection). *Let G be a (directed) graph with t copies of h-cycles. There is a randomized Congested Clique algorithm for h-cycle detection, which takes $\tilde{\mathcal{O}}(h^{\mathcal{O}(h)} \cdot n^{0.1567}/(t^{\frac{0.4617}{h-1.82408}} + 1))$ rounds w.h.p.*

Let $\mathcal{R}(G)$ denote the round complexity of Theorem 1. Let $\mathcal{R}_1(G), \mathcal{R}_2(G)$ denote the round complexity of the algorithms in Theorem 33 and Theorem 31 respectively. To prove Theorem 1, we show that for every graph G with n vertices and t copies of an h-cycle, we have $\min\{\mathcal{R}_1(G), \mathcal{R}_2(G)\} \leq \mathcal{R}(G)$. To show that, we use a case analysis. Recall that x denotes the number of vertices in G that participate in an h-cycle, and that $x^{h-\delta} = 2ht$. We show that if $\delta \geq 1.82408$, then $\mathcal{R}_2(G) \leq \mathcal{R}(G)$, and if $\delta \leq 1.82408$, then $\mathcal{R}_1(G) \leq \mathcal{R}(G)$.

The theorem then follows, as we can run the algorithms Find-Cycle and Find-Vertex-In-Cycle one step at a time, until one of them detects a triangle.

Proof of Theorem 1.

The Case $\delta \geq 1.82408$. The execution of the algorithm Find-Vertex-In-Cycle takes $\mathsf{MM}\left(\frac{n}{x}, n, n\right)$ rounds, where $\mathsf{MM}\left(\frac{n}{x}, n, n\right) \leq n^{\mathtt{B}} \cdot x^{-\mathtt{A}}$ by Definition 27. Since we assumed that $\delta \geq 1.82408$, we have $x = (2ht)^{1/(h-\delta)} \geq t^{1/(h-1.82408)}$. We get that $\mathcal{R}_2(G) \leq n^{\mathtt{B}} \cdot t^{-\frac{\mathtt{A}}{2-1.82408}}$. By plugging in $\mathtt{A} = 0.4617, \mathtt{B} = 0.1567$, (see Claim 28) we get that the round complexity is bounded by $n^{0.1567} \cdot t^{-\frac{0.4617}{2-1.82408}}$, which completes the proof of this case.

The Case $\delta \leq 1.82408$. The execution of the algorithm Find-Vertex-In-Cycle takes $\mathsf{MM}\left(\frac{n}{x}, \frac{n}{x}, \frac{n}{x}; x^{\delta}\right)$ rounds, where

$$\begin{aligned}
\mathsf{MM}\left(\frac{n}{x}, \frac{n}{x}, \frac{n}{x}; x^{\delta}\right) &= \mathcal{O}(n^{\rho}/x^{2+\delta(\rho-1)}) \\
&= \mathcal{O}(n^{\rho}/t^{\frac{2+\delta(\rho-1)}{h-\delta}}) \\
&\leq \mathcal{O}(n^{\rho}/t^{\frac{2+1.82408(\rho-1)}{h-1.82408}}) \\
&\leq \mathcal{O}(n^{0.1567} \cdot t^{\frac{0.4617}{h-1.82408}})
\end{aligned}$$

The first equality follows from Theorem 14. The penultimate inequality follows since the function $\delta \mapsto \frac{2+\delta(\rho-1)}{h-\delta}$ is monotonically decreasing in the range $\delta \in [0, 1.82408]$. The last inequality follows by setting $\rho \leftarrow 0.1567$, which completes the proof. ◀