Fast Distributed Algorithms for Girth, Cycles and Small Subgraphs

Keren Censor-Hillel
Technion – Israel Institute of Technology, Haifa, Israel
ckeren@cs.technion.ac.il

Orr Fischer
Tel-Aviv University, Israel
orrfischer@mail.tau.ac.il

Tzlil Gonen
Tel-Aviv University, Israel
tzlilgon@gmail.com

François Le Gall
Nagoya University, Japan
legall@math.nagoya-u.ac.jp

Dean Leitersdorf
Technion – Israel Institute of Technology, Haifa, Israel
dean.leitersdorf@gmail.com

Rotem Oshman
Tel-Aviv University, Israel
roshman@tau.ac.il

Abstract

In this paper we give fast distributed graph algorithms for detecting and listing small subgraphs, and for computing or approximating the girth. Our algorithms improve upon the state of the art by polynomial factors, and for girth, we obtain a constant-time algorithm for additive +1 approximation in Congested Clique, and the first parametrized algorithm for exact computation in Congest.

In the Congested Clique model, we first develop a technique for learning small neighborhoods, and apply it to obtain an $O(1)$-round algorithm that computes the girth with only an additive +1 error. Next, we introduce a new technique (the partition tree technique) allowing for efficiently listing all copies of any subgraph, which is deterministic and improves upon the state-of the-art for non-dense graphs. We give two concrete applications of the partition tree technique: First we show that for constant k, it is possible to solve C_{2k}-detection in $O(1)$ rounds in the Congested Clique, improving on prior work, which used fast matrix multiplication and thus had polynomial round complexity. Second, we show that in triangle-free graphs, the girth can be exactly computed in time polynomially faster than the best known bounds for general graphs. We remark that no analogous result is currently known for sequential algorithms.

In the Congest model, we describe a new approach for finding cycles, and instantiate it in two ways: first, we show a fast parametrized algorithm for girth with round complexity $\tilde{O}(\min\{g \cdot n^{1-1/\Theta(g)}, n\})$ for any girth g; and second, we show how to find small even-length cycles C_{2k} for $k = 3,4,5$ in $O(n^{1-1/k})$ rounds. This is a polynomial improvement upon the previous running times; for example, our C_6-detection algorithm runs in $O(n^{2/3})$ rounds, compared to $O(n^{3/4})$ in prior work. Finally, using our improved C_6-freeness algorithm, and the barrier on proving lower bounds on triangle-freeness of Eden et al., we show that improving the current $\tilde{\Omega}(\sqrt{n})$ lower bound for C_6-freeness of Korhonen et al. by *any* polynomial factor would imply strong circuit complexity lower bounds.

2012 ACM Subject Classification Networks → Network algorithms; Theory of computation → Distributed algorithms

Keywords and phrases distributed graph algorithms, cycles, girth, Congested Clique, CONGEST

Digital Object Identifier 10.4230/LIPIcs.DISC.2020.33

Funding This project was partially supported by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement no. 755839, by the JSPS KAKENHI grants JP16H01705, JP19H04066 JP20H04139 and JP20H00579 and by the MEXT Q-LEAP grant JP-MXS0120319794.

1 Introduction

A fundamental problem in many computational settings is that of finding cycles and other small subgraphs within a given graph. This paper focuses on finding subgraphs in distributed networks that communicate through limited bandwidth. The motivation for this is two-fold: first, for some subgraphs H there exist distributed algorithms that perform better on H-free graphs, such as distributed cut and coloring algorithms in triangle-free graphs [15, 25]. The second reason for which we are interested in these problems is that while solving them only requires obtaining *local* knowledge, about small non-distant neighborhoods, the bandwidth restrictions impose a major hurdle for collecting this information. This induces a rich landscape of complexities for subgraph-related problems. We contribute to the effort of characterizing the complexities of subgraph-related problems by providing new techniques, from which we derive fast algorithms for such problems in the two key distributed bandwidth restricted models, namely, CONGEST and CONGESTED CLIQUE.

In the CONGESTED CLIQUE model, n synchronous nodes can send messages of $O(\log n)$ bits in an all-to-all fashion. The input graph is an arbitrary n vertex graph, partitioned such that every node receives the edges of a single vertex as input. Our main contribution in this model is an algorithm for obtaining a $+1$ approximation for the girth in a *constant* number of rounds, where the girth of a graph is the length of its shortest cycle.

▶ **Theorem 1.** *Given a graph G with an unknown girth g, there exists a deterministic $O(1)$ round algorithm in the CONGESTED CLIQUE model which outputs an integer a, such that $g \in \{a, a+1\}$.*

For comparison, note that the current state-of-the-art algorithm computes the exact girth in $O(n^{0.158})$ rounds [4]. To obtain our $+1$ approximation algorithm, we devise two main new methods, which we describe here in a nutshell. The first is an algorithm in which each node learns its entire neighborhood up to a radius which is a constant approximation of the girth. To this end, we prove that we can quickly list all paths of sufficient length, as well as efficiently distribute them to the nodes that need to learn them. The second method that we introduce is a way to double the radius of the neighborhoods that all the nodes know, by having each node acquire the knowledge held by the farthest nodes in its currently-known neighborhood. Crucially, both of these procedures can be done in $O(1)$ rounds, and could be useful for additional applications.

Our second contribution in the CONGESTED CLIQUE model is a *partition tree* technique which allows for efficiently detecting or listing all copies of any subgraph with at most $\log n$ nodes, in a deterministic manner. In particular, our main application of the partition tree technique is to obtain the following subgraph listing algorithm, which improves upon the state-of-the-art for non-dense graphs.

▶ **Theorem 2.** *Given a graph G with n nodes and m edges and a graph H with $p \leq \log n$ nodes and k edges, let $\tilde{m} = \max\{m, n^{1+1/p}\}$. There exists a deterministic CONGESTED CLIQUE algorithm that terminates in $O(\frac{k\tilde{m}}{n^{1+2/p}} + p)$ rounds and lists all instances of H in G.*

We give two concrete applications of this result. The first is fast detection of even cycles.

▶ **Corollary 3.** *Given a graph G and an integer $k \leq (\log n)/2$, there exists a deterministic $O(k^2)$-round algorithm in the CONGESTED CLIQUE model for detecting cycles of length $2k$.*

Note that for constant k the above algorithm completes within $O(1)$ rounds. Prior work for cycle detection in the CONGESTED CLIQUE model used fast matrix multiplication (FMM) and thus had polynomial round complexity, apart from detecting 4-cycles which was shown to have a constant-round algorithm [4]. The second implication of the partition-tree technique is a fast algorithm for computing the *exact* girth in triangle-free graphs. Prior algorithms for girth in the Congested Clique model are based on fast matrix multiplication (FMM), a technique that can be no faster than checking for triangle-freeness.

▶ **Corollary 4.** *Given a triangle-free graph G with an unknown girth g, there exists a deterministic $\tilde{O}(n^{1/10})$-round algorithm in the Congested Clique model which outputs g.*

This result leverages the fact that graphs without small cycles become increasingly sparse, and the algorithm of Theorem 2 is efficient on sparse graphs. We remark that, interestingly, no analogous result going below the complexity of FMM for girth in triangle-free graphs is known for sequential algorithms, since the best known sequential algorithms for cycle detection in sparse graphs (see [2]) are not fast enough. We also note that given further lower bounds on the girth beyond triangle-freeness, the runtime of our algorithm improves even further; for instance, if the graph does not contain any k-cycle for $k \in \{3, 4, 5\}$, then our algorithm computes the exact girth in $\tilde{O}(n^{1/21})$ rounds. We refer to Proposition 13 in Section 4 for a more precise statement.

In the Congest model, n synchronous nodes can send messages of $O(\log n)$ bits to their neighbors only, and the input graph is the communication graph. In this model, we develop a new approach for finding cycles of a given size. A key step that is present in all known sublinear-round algorithms for finding cycles in Congest is the elimination of *high-degree vertices*: we check whether there is a cycle that includes a high-degree node, and if we conclude that there is no such cycle, we can remove the high-degree nodes from the graph. The remaining graph is much easier to handle, since it has low degree. In prior work, the high-degree vertices were eliminated by sequentially enumerating over them and starting a short BFS from each one. Here we introduce a different method for finding cycles that include a high-degree node: intuitively, we show that if we start from a *neighbor* of a small even cycle, we can quickly find the cycle itself. Since high-degree nodes have many neighbors, if we sample a uniformly random node in the graph, we are somewhat likely to hit a neighbor of the high-degree node, and from there we can find the cycle in constant rounds.

We apply this technique to give a fast algorithm that detects small even cycles, and a fast parameterized algorithm for computing the *exact* girth. Specifically, we obtain the following:

▶ **Theorem 5.** *Given a graph G, there exists a randomized algorithm in the Congest model for detection of $2k$-cycles in $O(n^{1-1/k})$ rounds, for $k = 3, 4, 5$.*

This significantly improves upon the running time of $O(n^{1-1/\Theta(k^2)})$ of the previous state-of-the-art [10]: for cycles of length 6,8 or 10, the previous algorithm had running time $O(n^{3/4})$, $O(n^{5/6})$ or $O(n^{10/11})$, respectively. We believe that going below round complexity of $O(n^{1-1/k})$ for C_{2k}-detection in the Congest model would require a breakthrough beyond currently known techniques, with potential ramifications also for the Congested Clique model.

For exact girth, previously, an $O(n)$-round algorithm for exact girth was known, based on computing all-pairs shortest paths [16]. Our result is as follows:

▶ **Theorem 6.** *Given a graph G with an unknown girth g, there exists a randomized $O(\min\{g \cdot n^{1-1/\Theta(g)}, n\})$-round algorithm in the Congest model which outputs g.*

"Outputs" here means that the first node that halts outputs the girth. Other nodes of the graph may halt later, and output larger values. This is unavoidable, unless we introduce a term in the running time that depends on the diameter of the graph.

Our final result is an *obstacle* on proving lower bounds for C_6-freeness in CONGEST. In [19] it was shown that the C_{2k}-freeness problem is subject to a lower bound of $\widetilde{\Omega}(\sqrt{n})$, for any k. For C_6-freeness, the best known algorithm is our new algorithm here, which runs in $\tilde{O}(n^{2/3})$ rounds, and there are reasons to believe that this may be optimal. Unfortunately, we show that proving a lower bound of the form $\Omega(n^{1/2+\alpha})$, for any constant $\alpha > 0$, would imply breakthrough results in circuit complexity. This result uses ideas from our improved C_6-freeness algorithm, and the barrier on proving lower bounds on triangle-freeness from [10, 11].

Related work. The problem of subgraph-freeness, and in particular cycle detection, has been extensively studied in the CONGESTED CLIQUE and CONGEST models. While there are only a few papers which study girth computation, related problems such as diameter computation or shortest paths were also extensively studied in these models.

In the first work to consider girth computation in the sequential setting, Itai and Rodeh [17] gave algorithms with running time $O(mn)$ and $O(n^2)$ for computing exact girth and $+1$ approximation of the girth, respectively, using a BFS approach, and an $O(n^\omega)$ algorithm for exact girth using an algebric method, where ω is the exponent of matrix multiplication. Later, various trade-offs between running time and additive or multiplicative approximations for girth were obtained (e.g [22, 26–28]).

In the CONGESTED CLIQUE model, an $O(n^{0.158})$ round algorithm for exact girth and a $2^{O(k)}n^{0.158}$ round algorithm for C_k-detection for any k was shown by [4] based on matrix multiplication techniques. These algebraic techniques were later extended by [3, 5, 20].

For general subgraphs, a CONGESTED CLIQUE algorithm for listing all instances of a subgraph H of size k in $\widetilde{O}(n^{1-2/k})$ rounds was shown in [8], which was shown to be tight for triangles [18, 23] and later for cliques of size $k > 3$ as well [12]. In this work we give a "sparsity-aware" version of this result, which has improved performance as the graph becomes sparser. Previously, distributed "sparsity-aware" algorithm were studied in the context of distributed sparse matrix multiplication [3, 5] and in the context of the k-machine model [23].

Frischknecht et al. [13] was the first work to consider girth computation in CONGEST, and showed that at least $\widetilde{\Omega}(\sqrt{n})$ rounds are required in order to obtain a $(2-\epsilon)$-approximation of the girth. Peleg et al. [24] showed an algorithm computing a $(2-1/g)$-approximation of the girth with round complexity $O(D + \sqrt{gn}\log n)$, where g is the girth of the graph. Holzer et al. [16] showed an algorithm for exact girth computation in $O(n)$ rounds, based on an exact all-pairs shortest path algorithm, and an algorithm for computing an $(1+\epsilon)$-approximation of the girth in $O(\min\{n/g + D\log(D/g), n\})$ rounds.

In the cycle-freeness problem in the CONGEST setting, Drucker et al. [9] showed a near tight lower bound of $\widetilde{\Omega}(n)$ for constant sized odd-length cycles, as well as a lower bound of $\widetilde{\Omega}(n^{1/k})$ for C_{2k}-freeness, which was later improved to $\widetilde{\Omega}(\sqrt{n})$ by Korhonen et al. [19]. A CONGEST randomized algorithm for listing all triangles with round complexity $\widetilde{O}(n^{1/3})$ was shown in [7], which improved the previous $\widetilde{O}(n^{1/2})$-round algorithm of [6] and the $\widetilde{O}(n^{3/4})$-round algorithm of [18]. The first sublinear-time algorithm for C_{2k}-freeness for $k \geq 3$ was given in [12] running in $\widetilde{O}(n^{1-1/(k^2-k)})$ rounds, and was later improved in [10] to round complexity $\widetilde{O}(n^{1-2/(k^2-k+2)})$ for odd k and $\widetilde{O}(n^{1-2/(k^2-2k+4)})$ for even k.

2 Preliminaries

Definitions. Given a graph H, the H-*listing* problem is a problem in which each node may output a set of H-copies, and the goal of the network is that w.h.p. the union over the sets of outputted H-copies by the nodes is exactly the set of H-copies in G.

The *Túran number* of a graph H, denoted $\mathrm{ex}(n, H)$, is the maximum number of edges m such that there exists a graph G on n vertices and m edges which contains no copy of H.

▶ **Lemma 7** (Túran number of C_{2k} [14]). *For $k \in \mathbb{N}$, if G is C_{2k}-free, then G contains at most $17kn^{1+1/k}$ edges.*

▶ **Lemma 8** (Túran number for girth [14]). *If G is C_i-free for all $3 \leq i \leq 2k$, then G contains at most $n^{1+1/k} + n$ edges.*

Let $N_i(v)$ denote the *graph* defined by the nodes of hop-distance at most i from v, that is, it includes all such nodes and all the *edges* incident to nodes with hop-distance at most $i-1$ from v. For sets $A, B \subseteq V$, denote by $E(A, B) = \{(a, b) \in E \mid a \in A \wedge b \in B\}$ the set of edges between A and B.

Load-Balanced Routing in the Congested Clique Model. We introduce a useful routing procedure which extends that of [21], and it is used throughout our results. The routing procedure of [21] routes a set of messages where each node needs to send and receive at most $O(n)$ messages, in $O(1)$ rounds. The following shows that it possible to replace the constraint where each node needs to *send* at most $O(n)$ messages with one stating that the messages each node desires to send are based on at most $O(n \log n)$ bits. This allows us to route messages even when the some nodes are each a source of $\omega(n)$ messages.

▶ **Lemma 9** (Load Balanced Routing). *Any routing instance $\mathcal{M}$, in which every node v is the target of up to $O(n)$ messages, and v locally computes the messages it desires to send from at most $|R(v)| = O(n \log n)$ bits, can performed in $O(1)$ rounds.*

The proof of Lemma 9 is deferred to the full paper. Notice that this lemma implies, in a straightforward manner, the following Corollary 10 which we refer to extensively.

▶ **Corollary 10.** *In the deterministic* Congested Clique *model, given that each node originally begins with $O(n \log n)$ bits of input, and at most $O(1)$ rounds have passed since the initiation of the algorithm, then any routing instance $\mathcal{M}$, in which every node v is the target of up to $O(n \cdot x)$ messages, can performed in $O(x)$ rounds.*

While this is a weaker statement than that of Lemma 9, it is convenient to use when showing constant-time algorithms in the Congested Clique model, as it completely circumvents the need for a bound on the number of messages each node desires to send.

3 Deterministic $O(1)$ Round Algorithm for +1 Girth Approximation in the Congested Clique

In this section we prove Theorem 1: we construct a deterministic $O(1)$ round algorithm for +1 girth approximation in the Congested Clique model. The algorithm is composed of two phases, each a novel technique on its own, and through their combination, we achieve the desired result. The first procedure is based on a *subgraph enumeration* approach and allows each node to learn its $\lfloor \frac{g}{20} \rfloor$ hop-neighborhood in $O(1)$ rounds. Formally, it is shown in the following Theorem 11.

▶ **Theorem 11.** *Given a graph G, with n nodes, and an unknown girth $g < \log n$, there exists an $O(1)$ round algorithm in the* Congested Clique *model, which either outputs g, or, ensures that every node knows its $\lfloor g/20 \rfloor$ hop-neighborhood.*

The latter procedure is based on a *BFS-like* approach and allows each node to double the hop-distance of the neighborhood which it knows in $O(1)$ rounds, *at least* until the first cycle is encountered. This is stated formally in Theorem 12

▶ **Theorem 12.** *Let G be a graph with n nodes, an unknown girth $g < \log n$, and with minimum degree $\delta \geq 2$. Assume that for a given integer parameter $a > 0$, every $v \in V$ knows the edges of $N_a(v)$, and that $N_a(v)$ is a tree. There exists an algorithm which completes in $O(1)$ rounds of the* Congested Clique *model, and either reports g exactly, or, reaches one of the following two states: (1) Every $v \in V$ knows $N_{2a}(v)$,* or *(2) For some value $b \geq \lceil \frac{g}{2} \rceil - 1$ which is agreed upon by all nodes of the network, every $v \in V$ knows all of $N_b(v)$. All nodes know whether g was reported exactly, and if not, which state was reached.*

Thus, by invoking the first algorithm once, and then the latter for a constant number of times, we achieve an $O(1)$ round algorithm for the approximation problem. The reason we achieve a $+1$ approximation, and not an exact result, is due to the fact that the second algorithm stops right before detecting the shortest cycle in the graph and cannot differentiate whether it is of odd or even length. In Section 3.1, we formally prove Theorem 1, when $g < \log n$ and the minimum degree is $\delta \geq 2$, using Theorems 11, 12. The constraints $g < \log n$ and $\delta \geq 2$ can be quickly overcome by eliminating some trivial, degenerate cases, and these details are deferred to the full version. Finally, in Sections 3.2, 3.3, we proceed to our fundamental technical contributions by showing the proofs of Theorems 11, 12.

3.1 Proving Theorem 1

Here, we prove Theorem 1, in case that $g < \log n$ and the minimum degree is $\delta \geq 2$.

Proof of Theorem 1. We first invoke the algorithm from Theorem 11, in $O(1)$ rounds. Either g was outputted, or, every node learned its $\lfloor g/20 \rfloor$ hop-neighborhood.

Next, we invoke the algorithm from Theorem 12. The nodes now learned new, larger neighborhoods - regardless of whether the algorithm halted in State 1 (every $v \in V$ knows $N_{2a}(v)$) or State 2 (for some $b \geq \lceil \frac{g}{2} \rceil - 1$, every $v \in V$ knows all of $N_b(v)$). If any node sees a cycle, then it broadcasts the length of the shortest cycle which it sees and all the nodes terminate and output the minimum of the values which were broadcast in the network. It is clear that, in this case, the exact value of g is outputted, since all nodes know the neighborhoods surrounding them of same radius, and thus if any node saw a cycle, one node must have seen the shortest cycle in the graph.

Finally, in the case that no cycle was seen so far, we differentiate between the states at which the algorithm from Theorem 12 can halt at. If it halts at State 1, then every node learned twice the radius of the neighborhood it already knew. In such a case, we invoke Theorem 12 again and repeat. Notice that we can do this at most $O(1)$ times, before either seeing a cycle or halting at State 2, due to the fact that the nodes originally know their $\lfloor g/20 \rfloor$ hop-neighborhoods. In the case that we eventually halt at State 2, and no cycles were seen by any node so far, all the nodes output that the girth is either $2b + 1$ or $2b + 2$, where b is the radius of the neighborhoods which they learned. It is clear that if all nodes learned their $b \geq \lceil \frac{g}{2} \rceil - 1$ hop-neighborhoods, and none saw a cycle, then it must be that $b = \lceil \frac{g}{2} \rceil - 1$ and thus either $g = 2b + 1$ or $g = 2b + 2$. ◀

3.2 Phase I: Initial Neighborhood Learning

The key procedure of this phase (formally stated above as Theorem 11) consists of two major steps. In the first step, either each path of length $\lfloor \frac{g}{10} \rfloor$ in G is detected by at least one node, or g is outputted. This step can be seen as an edge-partition variant of the listing algorithm in [8]. The second step uses a load-balancing routine in order to redistribute the information computed in the first step so that each node v learn its $\lfloor \frac{g}{20} \rfloor$ hop-neighborhood.

Step 1: Path Listing. We next list all paths of length $\lfloor \frac{g}{10} \rfloor$, or output g, in $O(1)$ rounds.

First, each node sends its degree to the rest of the network, and each then locally calculates the number of edges in the graph $m = \sum_v deg(v)/2$. Let $k \in \mathbb{N}$ be the largest integer such that $m \leq n^{1+1/k} + n$. Then, each node v is assigned a hard-coded range of $deg(v)$ indices in $\{1, \dots, m\}$, and locally numbers its edges using these indices.

If $k \leq 4$, then by Lemma 8 the girth is of size at most 10, and thus, trivially, paths of length $\lfloor \frac{g}{10} \rfloor \leq 1$ are known and we can halt. Thus, from here on, we may assume that $k \geq 5$.

Let P be a partition of the set $\{1, \dots, m\}$ into $\lceil kn^{2/k}/(20e) \rceil$ consecutive segments of size $O\left(\frac{m}{\lceil kn^{2/k}/(20e) \rceil} + 1\right)$, and let K be a family containing all the possible choices of $\lfloor k/4 \rfloor$ segments from P (in this context, e denotes the mathematical constant). It holds that

$$|K| = \binom{\lceil kn^{2/k}/(20e) \rceil}{\lfloor k/4 \rfloor} \leq \left(\frac{e \lceil kn^{2/k}/(20e) \rceil}{\lfloor k/4 \rfloor} \right)^{\lfloor k/4 \rfloor} \leq \left(\frac{n^{2/k}}{2} + 1 \right)^{\lfloor k/4 \rfloor} \leq n^{\frac{2}{k} \lfloor k/4 \rfloor} \leq n,$$

where the first inequality holds due to the well-known combinatorial statement that $\binom{n}{k} \leq (\frac{ne}{k})^k$, for all $n \in \mathbb{N}, 1 \leq k \leq n$, and in the other inequalities, the fact that $5 \leq k < \log n$ is used. Thus, it is possible to associate each $k_i \in K$ with a unique node v_i. Each k_i is a set of $\lfloor k/4 \rfloor$ sets of $O\left(\frac{m}{\lceil kn^{2/k}/(20e) \rceil} + 1\right)$ edges, and so let E_i denote the edges in the sets contained in k_i. Notice that

$$|E_i| \leq \lfloor k/4 \rfloor \left(\frac{m}{\lceil kn^{2/k}/(20e) \rceil} + 1 \right) \leq (k/4) \frac{20en^{1+1/k}}{kn^{2/k}} + k = 5en^{1-1/k} + k \leq n,$$

and therefore, by Corollary 10, it is possible for each v_i to learn all of the edges in E_i in $O(1)$ rounds of communication.

Finally, every node v_i broadcasts the shortest cycle which it witnesses in E_i. Notice that every path, p, of length at most $\lfloor k/4 \rfloor$, is fully contained inside some E_j, due to the construction of P, and therefore the corresponding node, v_j, which now knows all of E_j, will witness p. Thus, if $g \leq \lfloor k/4 \rfloor$, some node will witness the shortest cycle in the graph and be able to broadcast its length, g. Otherwise, notice that since $m \not\leq n^{1+1/(k+1)} + n$, Lemma 8 implies that the graph is not C_i-free for all $i \leq 2(k+1)$, and thus $g \leq 2(k+1)$. Thus all paths of length at most $\lfloor k/4 \rfloor \geq \lfloor g/8 - 1/4 \rfloor \geq \lfloor g/10 \rfloor$ have been listed. Notice that $g/8 - 1/4 \geq g/10$ whenever $g \geq 10$, and this can be assumed, since otherwise, trivially, paths of length $\lfloor \frac{g}{10} \rfloor < 1$ are known.

If at least one node v_i informs about a cycle in E_i, the minimum number sent by a node is outputted as g, and the algorithm terminates. Otherwise, it proceeds to the second step.

Step 2: Neighborhood Learning. We desire to redistribute some of the information learned in the previous step so that each node will know its $\lfloor \frac{g}{20} \rfloor$ hop-neighborhood.

Notice that all paths of length at most $\lfloor g/10 \rfloor$ have been listed. Therefore, also all paths of length at most $\lfloor \frac{g}{20} \rfloor$ have been listed. We strive to redistribute this information so that each node v knows all paths of length at most $\lfloor \frac{g}{20} \rfloor$ which start at v, and thus v knows its entire $\lfloor \frac{g}{20} \rfloor$ hop-neighborhood. Notice that we would like for each v to know both the nodes in its $\lfloor \frac{g}{20} \rfloor$ hop-neighborhood, as well as the edges between them.

We begin by ensuring that each v knows every node u in its $\lfloor \frac{g}{20} \rfloor$ hop-neighborhood. Let $v \in V$ and u be some node in its $\lfloor \frac{g}{20} \rfloor$ hop-neighborhood. Since $\lfloor \frac{g}{20} \rfloor < g/2$, then the $\lfloor \frac{g}{20} \rfloor$ hop-neighborhood of v is a tree. Therefore, there exists exactly one path, $p_{v,u}$, of length at most $\lfloor \frac{g}{20} \rfloor$ between v and u. In the previous step, we ensured that at least one node w is aware of $p_{v,u}$. Specifically, notice that it might be the case that many nodes know about $p_{v,u}$, due to the last step, yet, every node w which knows of this path also knows all the other nodes w' which learned this path through their $E_{w'}$. Thus, it is possible to choose, in a hard-coded manner, a single node w which will be responsible for informing v that $p_{v,u}$ exists. Having done that, node w desires to convey to v the message that u is in its $\lfloor \frac{g}{20} \rfloor$ hop-neighborhood, in addition to the hop-distance between v and u – that is, the length of $p_{v,u}$. Notice that for each such u, node v is destined to receive exactly one message, and therefore every node in the graph is the target of $O(n)$ messages. This shows that Corollary 10 may be invoked in order to deliver all these messages in $O(1)$ rounds.

Now, we desire to inform every v of the edges in its $\lfloor \frac{g}{20} \rfloor$ hop-neighborhood. Node v now knows all the nodes u in this neighborhood, as well as the hop-distance to each of them. Node v sends a message to each such u which is at most $\lfloor \frac{g}{20} \rfloor - 1$ hops away from it, and requests that u send to v *all* its incident edges in the graph. Notice that all these edges are exactly all the edges contained in the $\lfloor \frac{g}{20} \rfloor$ hop-neighborhood of v, and since this neighborhood is a tree, v is the target of at most $O(n)$ messages. As before, this shows that Corollary 10 may be invoked in order to deliver all these messages in $O(1)$ rounds.

3.3 Phase II: Neighborhood Doubling

The key procedure in this phase (formally stated above as Theorem 12) is an $O(1)$ round algorithm which doubles the radius of the hop-neighborhood known to each node, until the nodes know a neighborhood large enough in order to approximate the girth up to an additive value of 1. The algorithm works along the following lines. Denote by $F_a(v)$, the nodes which are exactly at distance a from v – we refer to these as the *front-line* nodes. Each nodes v initially knows $N_a(v)$, and at once attempt to learn all of $\bigcup_{u \in F_a(v)}(N_a(u) \setminus N_a(v))$, in an efficient manner. If this step succeeds, then all the nodes reach State 1, and halt. Otherwise, they coordinate to increase the radii of the neighborhoods which they know by as much as possible in $O(1)$ rounds, and ultimately arrive at State 2, and halt.

Halting at State 1. Let $v \in V$ and $u \in F_a(v)$. Node u aims to send to node v the edges in $N_a(u) \setminus N_a(v)$. Notice that node u can locally compute these edges as follows. It observes the first node w on the path between v, u. Since $N_a(u)$ is a tree, for every node $w' \in N_a(u)$, there is exactly one simple path, $p_{u,w'}$, which u sees to w'. Notice that $w' \in N_a(v)$ if and only if $p_{u,w'}$ passes through w. Thus, node u knows exactly which edges it desires to send to node v. However, before sending them, it first sends to node v the value $|N_a(u) \setminus N_a(v)|$.

We now shift back to the perspective of node v. It computes and broadcasts an upper bound on $|\bigcup_{u \in F_a(v)}(N_a(u) \setminus N_a(v))|$, by calculating $\sum_{u \in F_a(v)} |N_a(u) \setminus N_a(v)|$. If all nodes broadcast values which are at most $n - 1$, then by Corollary 10, it is possible in $O(1)$ rounds to perform all the routing requests and have each node double the radius of the neighborhood which it knows. At this point, the nodes collectively reach State 1 and halt.

Otherwise, at least one node reported a value greater than or equal to n. This implies that for some node v, there is a cycle in $N_{2a}(v)$, since at least two nodes $u, u' \in F_a(v)$ have simple paths in their a hop-neighborhoods to the same node w. In this case, the nodes proceed to a second part of the algorithm, which eventually leads to halting at State 2.

Halting at State 2. Our goal, at this stage, is to determine the largest possible value $i' \in \{1, \ldots, a-1\}$, such that for every node v, $\sum_{u \in F_a(v)} |N_{i'}(u) \setminus N_a(v)| < n$. Once this is achieved, then the algorithm can complete in a similar manner to that above. To see this, assume that we have this maximal value i'. Therefore, all nodes v can learn $N_{a+i'}(v)$ in $O(1)$ rounds, similarly to above. If any cycle is seen, then g is outputted and the algorithm halts. Otherwise, due to the definition of i', there must exist some node v' such that $\sum_{u \in F_a(v')} |N_{i'+1}(u) \setminus N_a(v')| \geq n$. This implies that there is a cycle in $N_{a+i'+1}(v')$, and therefore $2a + 2i' < g \leq 2a + 2i' + 2$. As such, $a + i' = (2a' + 2i' + 2)/2 - 1 \geq \lceil g/2 \rceil - 1$, and we may halt at State 2.

We now show how to find i'. This is trivially possible to accomplish in $O(a)$ rounds – each node u simply sends to v the values $\{|N_1(u) \setminus N_a(v)|, \dots, |N_{a-1}(u) \setminus N_a(v)|\}$, node v locally computes the $a-1$ different sums, and broadcasts them. However, this does not suffice for our goal of an $O(1)$ round algorithm, as a can be logarithmic in n. Instead, let every node v broadcast $|F_a(v)|$, and denote by v' the node with maximal $|F_a(v')|$, and write $d = \lfloor n/|F_a(v')| \rfloor$. For every node v and $u \in F_a(v)$, node u sends to v the values $\{|N_1(u) \setminus N_a(v)|, \dots, |N_d(u) \setminus N_a(v)|\}$, node v computes the d different sums of these values from all $u \in F_a(v)$, and broadcast them. Notice that this takes $O(1)$ rounds, using Corollary 10 as each node wants to receive at most $O(n)$ messages. Notice that it is now possible in $O(1)$ rounds to compute $\min\{i', d\}$ – either $i' \leq d$, and thus $\min\{i', d\} = i'$ and we can compute it, or, $\min\{i', d\} = d$. If we show that $g \leq 2a + 2d$, then if all v learn $N_{a+\min\{i',d\}}(v)$, this would suffice in order to either find the exact girth or halt at State 2, as required.

We claim that $g \leq 2a + 2d$. To see this, assume that $g > 2a + 2d$. Since $g > 2a + 2d$, there are no cycles in $N_{a+d}(v')$. Combining this with the fact that we assume the minimal degree in G to be at least 2, we can see that for all $j \neq j' \in \{1, \dots, d\}$, it holds that $|F_{a+j}(v')| \geq |F_{a+j-1}(v')|$, and $F_{a+j}(v') \cap F_{a+j'}(v') = \emptyset$. Thus, in $N_{a+d}(v')$ there are at least $(d+1) \cdot |F_a(v')| = (\lfloor n/|F_a(v')| \rfloor + 1) \cdot |F_a(v')| > n$ nodes, a clear impossibility. As we have arrived at a contradiction, we get that $g \leq 2a + 2d$, as required.

4 Subgraph Listing in the Congested Clique Model

We show an efficient "sparsity-aware" algorithm to *list* subgraphs in the Congested Clique model. Our main result is the following theorem, which is proven in the following sections.

▶ **Theorem 2.** *Given a graph G with n nodes and m edges and a graph H with $p \leq \log n$ nodes and k edges, let $\tilde{m} = \max\{m, n^{1+1/p}\}$. There exists a deterministic Congested Clique algorithm that terminates in $O(\frac{k\tilde{m}}{n^{1+2/p}} + p)$ rounds and lists all instances of H in G.*

As mentioned in the introduction, we can combine this result with known bounds on the number of edges in graphs without specific subgraphs, to achieve fast subgraph *detection* results. First, by combining Theorem 2 with Lemma 7, we immediately get Corollary 3: If the graph contains more than $17kn^{1+1/k}$ edges (which can be checked in a single round), then by Lemma 7 we can safely output that there must exist a cycle of length $2k$. Otherwise, plugging $p = 2k$ and $m = 17kn^{1+1/k}$ in Theorem 2 gives that we can detect (and even list, in this case) the existence of a cycle of length $2k$ within $O(k^2)$ rounds. Next, by combining Theorem 2 with Lemma 8, we can get the following result:

▶ **Proposition 13.** *Given a graph G with n nodes, m edges and an unknown girth g such that $g > \ell$ for some known ℓ, and defining $f(x) = n^{1/\lfloor (x-1)/2 \rfloor - 2/(2 \cdot \lfloor (x-1)/2 \rfloor + 1)}$, there is a deterministic $\tilde{O}(\min\{f(g), f(2 \cdot \lfloor (\ell+1)/2 \rfloor + 1)\})$ round algorithm in the Congested Clique model which outputs g.*

Proposition 13 first shows that the exact girth can be computed in $\tilde{O}(f(g))$ rounds – a polynomial improvement over the state-of-the-art for all graphs with $g \geq 5$. Moreover, if it is known that the graph has girth greater than ℓ,[1] then the round complexity is additionally guaranteed to be $\tilde{O}(f(2 \cdot \lfloor(\ell+1)/2\rfloor + 1))$. For instance, for any odd value $\ell = 2r-1$ we get the upper bound $\tilde{O}(n^{1/r-2/(2r+1)})$. Taking $r = 2$ gives Corollary 4 stated in the introduction, which improves upon the state-of-the-art for triangle free graphs.

We note that more claims can be shown using bounds for the Túran numbers of various other graphs – for example, for detection of $K_{s,t}$ (complete bipartite graph with s nodes on one side and t on the other) for certain values of s, t.

4.1 Partition trees

We introduce the notion of *partition trees*, as a fundamental tool for subgraph listing in the *deterministic* Congested Clique model. Partition trees are a deterministic load-balancing mechanism that evenly divides the work of checking whether any copies of a subgraph are present. In prior work, randomized load-balancing was used for this purpose, but this incurs logarithmic factors which we cannot tolerate here. Throughout this section, given a subgraph with p nodes, we frequently refer to the value $x = n^{1/p}$. We assume that x is an integer, because $p \leq \log n$ implies $x \geq 2$, and so it is possible to round x to an integer without affecting the round complexity or correctness.

We start with Definition 14, which defines a p-partition tree, which is a tree structure in which every node represents a partition of the graph G. Then, in Definition 15, we define an H-partition tree, in which we require certain conditions on the number of edges between parts of a p-partition, based on the subgraph H of p nodes which we will want to list.

► **Definition 14** (p-partition tree, Figure 1). *Let $G = (V, E)$ be a graph with n nodes and m edges, and let $p \leq \log n$. A p-*partition tree* $T = T_{G,p}$ is a tree of p layers (depth $p-1$), where each non-leaf node has at most $x = n^{1/p}$ children. Each node in the tree is associated with a partition of V consisting of at most x parts.*

We inductively denote all partitions associated with nodes in T as follows. The partition associated with the root r of T is called the root partition*, and is denoted by $P_\emptyset$. Given a node with a partition denoted by $P_{(\ell_1,\dots,\ell_{i-1})}$, the partition associated with its jth child, for $0 \leq j \leq x-1$, is denoted $P_{(\ell_1,\dots,\ell_{i-1},j)}$.*

The at most x parts of each partition $P_{(\ell_1,\dots,\ell_i)}$ are denoted by $U_{(\ell_1,\dots,\ell_i),j}$, for $0 \leq j \leq x-1$. For each $0 \leq j \leq x-1$, the part $U_{(\ell_1,\dots,\ell_{i-1}),\ell_i}$ is called the parent *of the part $U_{(\ell_1,\dots,\ell_{i-1},\ell_i),j}$, also denoted as $U_{(\ell_1,\dots,\ell_{i-1}),\ell_i} = \texttt{parent}(U_{(\ell_1,\dots,\ell_{i-1},\ell_i),j})$.*

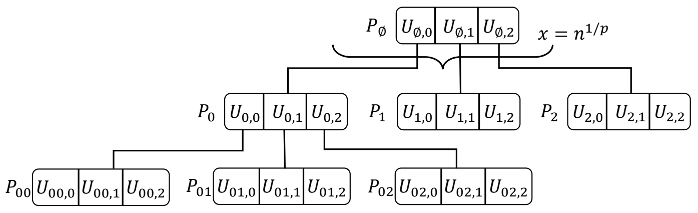


Figure 1 A partial illustration of a partition tree with $p, x = 3$.

► **Definition 15** (H-partition tree). *Let $G = (V, E)$ be a graph with n nodes and m edges, and let H be a graph with $p \leq \log n$ nodes, $\{z_0, \dots, z_{p-1}\}$, and denote $d_i = |\{\{z_i, z_t\} \in E_H \mid t < i\}|$*

[1] It is possible to phrase a slightly stronger result which does not require a lower bound on the girth, but rather that for a specific k, which depends on the sparsity of the graph, there will not be any cycles of length $2k+1$.

for each $0 \le i \le p-1$, $x = n^{1/p}$ *and* $\tilde{m} = \max\{m, nx\}$. *A* H-partition tree $T = T_{G,H}$ *is a* p*-partition tree with the following additional constraints, for some constants* c_1, c_2. .

1. *for every part* $U = U_{(\ell_1,\ldots,\ell_{i-1},\ell_i),j}$*, it holds that* $|E(U,V)| \le c_1 m/x + n$*, and*
2. *for every part* $U_i = U_{(\ell_1,\ldots,\ell_{i-1},\ell_i),j}$*, and all of its ancestor parts* $U_t = \texttt{parent}(U_{t+1})$ *for* $t = i-1, \ldots 0$*, it holds that* $\sum_{t<i,\{z_i,z_t\}\in E_H} |E(U,U_t)| \le c_2 d_i \tilde{m}/x^2 + n$*,*

Notice that in Definition 15, we define $\tilde{m}$ as an upper bound on m, the number of edges in the input graph. This is done as if the graph is *too* sparse, we use a slightly higher bound on the number of edges in order to make decisions regarding the constraints on the partitions. We note that $\tilde{m}$ is purely a technicality – we do not *require* that there be at least this many edges in the graph. In the following two theorems we show that we can construct an H-partition tree and use it to efficiently perform H-listing.

▶ **Theorem 16.** *Let* $G = (V,E)$ *be a graph with* n *nodes, and let* H *be a graph with* $p \le \log n$ *nodes. There exists a deterministic* Congested Clique *algorithm that completes in* $O(1)$ *rounds and constructs an* H*-partition tree* T*, such that* T *is known to all nodes of* G *– that is, all nodes know all the partitions making up* T*.*

Given an H-partition tree, we can list all instances of H in G.

▶ **Theorem 17.** *Let* $G = (V,E)$ *be a graph with* n *nodes, let* H *be a graph with* $p \le \log n$ *nodes and* k *edges, and denote* $x = n^{1/p}$ *and* $\tilde{m} = \max\{m, nx\}$*. There exists a deterministic* Congested Clique *algorithm that completes in* $O(\frac{k\tilde{m}}{n^{1+2/p}} + p)$ *rounds and lists all instances of* H *in* G*, given an* H*-partition tree* T *that is known to all nodes.*

Thus, Theorems 16 and 17 directly imply Theorem 2.

Proof sketch of Theorem 16. We construct a set of preliminary partitions in $O(1)$ rounds, and show that it is possible to construct the entire partition tree using only this set of partitions. By ensuring that these partitions are globally known, each node computes the entire tree locally. The root partition, denoted as R, only needs to adhere to the condition on the number of edges that touch each part, which is easily obtained by grouping nodes in any arbitrary order until the total number of edges exceeds the threshold.

Then, for every set of $1 \le \ell \le p-1$ parts of R, denoted $\{Q_{j_0}, \ldots, Q_{j_{\ell-1}}\}$, we construct a partition $M_{\{j_0,\ldots,j_{\ell-1}\}}$ of G with at most x parts. That is, we construct $(x/2+1)^{p-1} \le x^{p-1} = n/x$ additional different partitions. When constructing the partition $M_{\{j_0,\ldots,j_{\ell-1}\}}$, we maintain three conditions. Primarily, we maintain Condition 1; that is, for every part $N = N_{\{j_0,\ldots,j_{\ell-1}\},k}$ it holds that $|E(N,V)| \le c_1 m/x + n$. Furthermore, similarly to Condition 2, we ensure that for each part, $N = N_{\{j_0,\ldots,j_{\ell-1}\},k}$, $\sum_{0\le i<\ell} |E(N,Q_{j_i})| \le c_2 \ell \tilde{m}/x^2 + n$. Lastly, we ensure that $M_{\{j_0,\ldots,j_{\ell-1}\}}$ is a refinement of R.

To build the partitions and to be able to make each partition known to all nodes, we assign each partition with a set of x nodes that are called *the builder nodes*. Since we have n/x partitions, we can do this in a mutually disjoint manner. Finally, we show that each node can internally construct the entire partition tree from the set of all partitions that we created, because for every node in the partition tree we can always find the needed partition among the partitions M that we created. To see why, note first that all partitions satisfy Condition 1. Second, every partition in the tree has to satisfy Condition 2. Intuitively, this holds since the corresponding parts in the condition are parts in the root partition R, because we made sure that each partition is a refinement of R. Thus, for every set of at most p ancestor parts, we can always find among the partitions M that we created, a partition which corresponds to that set of ancestor parts with respect to Condition 2. ◀

Proof sketch of Theorem 17. Denote by $\{z_0, \dots, z_{p-1}\}$ the nodes of H, and denote $d_i = |\{\{z_i, z_t\} \in E_H \mid t < i\}|$ for each $0 \le i \le p-1$. We assign each leaf of the H-partition tree T to x different nodes. Note that there are x^{p-1} leaves, which is at most n/x due to our choice of $x = n^{1/p}$. We abuse the notation and denote a node in T with the same notation as we use for the partition that is associated with it. Each leaf $P_{(\ell_1,\dots,\ell_{p-1})}$ is thus assigned to x different nodes, and each part $U_{(\ell_1,\dots,\ell_{p-1}),j}$ in each leaf partition is assigned to a different node. For each node $v \in V$, we denote by $U_{v,p-1}$ the part of the leaf partition that it is assigned to. Then, inductively, for every $i = p-2, \dots 0$, we denote $U_{v,i} = \texttt{parent}(U_{v,i+1})$. Note that for all $v \in V$ we have that $U_{v,0}$ is a part in the root partition.

We now let each node $v \in V$ learn all edges in $\bigcup_{t<i \text{ s.t. } \{z_i,z_t\}\in E_H} E(U_{v,i}, U_{v,t})$ and list all the instances of H it sees. We prove that all instances of H in G are indeed listed by this approach, and that learning the required edges by all nodes takes $O(\frac{k\tilde{m}}{n^{1+2/p}} + p)$ rounds. ◀

5 Detecting Even Cycles and Computing the Girth in Congest

In this section we sketch our Congest algorithms for finding small even cycles and for computing the girth. Both algorithms use the same high-level idea for quickly finding a cycle of a given length; we begin with a high-level overview of this technique, and then give more details for the girth algorithm and the even-cycle detection algorithm.

5.1 Finding Cycles

Suppose we want to check if the graph contains a copy of the ℓ-cycle, C_ℓ. To simplify the exposition, let us assume that ℓ is even, $\ell = 2k$ (the odd case is handled in our exact girth algorithm, below). We partition copies of C_{2k} in the graph into two types:

Light cycles. A cycle $u_0, \dots, u_{2k-1}$ is called *light* if $\deg(u_i) \le n^{1/k}$ for each $i \in [2k]$. To find light $2k$-cycles, we "deactivate" all nodes with degree greater than $n^{1/k}$, and consider the subgraph G' induced by the nodes with degree at most $n^{1/k}$. In G', we simultaneously start a depth-k BFS from all nodes, by having each node of G' send out a BFS token that is forwarded up to k hops by all the nodes of G' that receive it. When all BFS instances are completed, node u_k receives the BFS token of node u_0 from two of its neighbors, and announces that it has found a $2k$-cycle. All BFS instances complete in $O(k \cdot n^{(k-1)/k})$ rounds, as the i-neighborhood of any node in G' is of size at most $n^{i/k}$ for each $i \in [k]$.

We must be careful to avoid "false positives", which might occur if u_0's token travels to u_k by paths that are not vertex-disjoint; in this case, even though u_k receives u_0's token from two neighbors, the graph might not contain a $2k$-cycle – it contains some smaller cycle. In our C_{2k} algorithm, since we are interested in constant (and very small) k, we resolve this issue using color coding [1], which incurs a multiplicative factor of $k^{O(k)}$ in the running time. In our exact girth algorithm, we cannot afford to use color coding, because we must consider cycles of super-constant length; however, here we can argue that by the time we search for ℓ-cycles, we have already ruled out the existence of ℓ'-cycles for all $\ell' < \ell$, and therefore no "false positives" can occur.

Heavy cycles. A cycle $u_0, \dots, u_{2k-1}$ is called *heavy* if it contains some node u_i with $\deg(u_i) > n^{1/k}$. Assume that u_0 is a node with the highest degree in the cycle, so that $\deg(u_0) > n^{1/k}$. To detect a heavy cycle, we exploit the fact that a randomly-sampled node

s in the graph will be a *neighbor* of u_0 with probability $\Omega(n^{1-1/k})$.[2] If we start a depth-k BFS from every neighbor of s, that would include u_0, and we could then detect the cycle by having node u_k receive u_0's token from its two neighbors u_{k-1}, u_{k+1}. There are two issues with this approach:

- As in the case of light cycles, a cycle of length $< 2k$ could cause a "false positive". We avoid this the same way we did above, using color-coding (for C_{2k}-detection) or the fact that there are no smaller cycles (for computing the girth).
- Starting a BFS from every neighbor of s could be too expensive: s could have high degree, and starting a BFS from many nodes at the same time could cause high congestion. We resolve this issue by:
  (I) First checking whether or not s itself participates in a $2k$-cycle, by starting a k-round BFS from s (using color-coding if necessary); and
  (II) Proving that if s does not participate in a $2k$-cycle, then w.h.p., it suffices for each cycle node to forward a *constant* number of BFS tokens, in order for u_k to receive the BFS token of u_0 from both u_{k-1} and u_{k+1}. For the girth this is straightforward, but for C_{2k}-detection it is more involved.

Together, we see that each time we sample a uniformly-random graph node, we can check in constant time whether it participates in or is adjacent to a $2k$-cycle. Since $\deg(u_0) > n^{1/k}$, after $O(n^{1-1/k})$ iterations we are guaranteed to find the cycle with high probability. We now give more details for each algorithm.

5.2 Computing the Girth

To compute the girth of the graph, we consider each length $\ell = 1, 2, \ldots, \log n$ sequentially, and search for ℓ-cycles in the graph using the framework described above. Each iteration takes $\ell \cdot n^{1-1/\lfloor \ell/2 \rfloor}$ rounds. By itself, this approach would have superlinear running time in graphs with high girth. In order to cap the running time at $O(n)$ rounds we run in parallel the linear-time all-pairs shortest path (APSP) algorithm of [16]. If the APSP algorithm terminates first, we use the results of APSP to compute the exact girth, and halt. Altogether, if the graph has girth g, then we terminate after $O(\min\{g \cdot n^{1-1/\lfloor g/2 \rfloor}, n\})$ rounds.

Let us instantiate the cycle-finding framework above for the case where we know the graph does not contain ℓ'-cycles for any $\ell' < \ell$, and want to check if the graph contains an ℓ-cycle. Each time we start a BFS, we let it proceed to depth $\lceil \ell/2 \rceil$, with the BFS token storing how many hops (edges) it has traveled; a node u detects an ℓ-cycle if it receives the BFS token of some node v from two distinct neighbors $w, w' \in N(u)$, with the token having travelled $\lceil \ell/2 \rceil$ hops to reach w and $\lfloor \ell/2 \rfloor$ hops to reach w'. We must address the following concerns:

"False positives". Suppose that node $u_{\lceil \ell/2 \rceil}$ receives the BFS token of node u_0 from its two neighbors $u_{\lceil \ell/2 \rceil - 1}$ and $u_{\lceil \ell/2 \rceil + 1}$, with the token having traveled $\lceil \ell/2 \rceil$ hops to reach $u_{\lceil \ell/2 \rceil - 1}$, and $\lfloor \ell/2 \rfloor$ hops to reach $u_{\lceil \ell/2 \rceil + 1}$. Let $u_0 = v_0, v_1, \ldots, v_{\lceil \ell/2 \rceil} = u_{\lceil \ell/2 \rceil}$ and $u_0 = w_0, w_1, \ldots, w_{\lfloor \ell/2 \rfloor} = u_{\lceil \ell/2 \rceil}$ be the paths along which u_0's token traveled to $u_{\lceil \ell/2 \rceil}$, and assume for the sake of contradiction that these paths contain some shared node $v_i = w_j$,

[2] We cannot actually sample one random node in the CONGEST model, but we can simulate this by assigning each node a random priority from the range $[n^3]$, so that w.h.p. the priorities are unique. Then we have the smallest-priority node "kill off" all its competitors in a neighborhood whose size matches the running time of the algorithm. This has the same effect as sampling one uniform node.

where $i, j > 0$ and either $i < \lceil \ell/2 \rceil$ or $j < \lfloor \ell/2 \rfloor$ (or both). Then, taking a lexicographically-smallest pair (i, j) such that $v_i = w_j$, we see that the graph contains a simple cycle $u_0, v_1, \ldots, v_i = w_j, w_{j-1}, \ldots, u_0$ of length at most $i + j < \ell$, contradicting our assumption that the graph is free of cycles of length less than ℓ.

Thus, the paths through which u_0's token reaches $u_{\lceil \ell/2 \rceil}$ must be node-disjoint (except their shared endpoints), which means there is a simple ℓ-cycle in the graph.

Congestion. suppose we have sampled a neighbor $s \in N(u_0)$, and verified that s does not participate in an ℓ-cycle. We now initiate a BFS from every neighbor of s (including u_0), and claim that u_0's BFS token is received by each cycle node $u_1, \ldots, u_{\ell-1}$ before any other BFS tokens are received: thus, it suffices to have each node forward one token only, and discard all tokens received afterwards. This claim is easily proven by induction on the distance of u_i from u_0: essentially, if some neighbor u_0' of s is able to "overtake" u_0's BFS token and arrive at some cycle node u_i first (or at the same time), and if u_i is the nearest such node to u_0, then we get a cycle of length at most ℓ that passes through s. But we already verified that s does not participate in an ℓ-cycle, and that the graph contains no smaller cycles, so this cannot happen.

5.3 $2k$-Cycle Detection

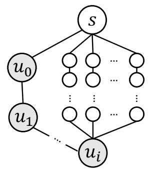
The details and especially the correctness proof of our $2k$-cycle detection algorithm are more complex, so we give here a high-level overview of the ideas, and defer the details to the full version. For the sake of symmetry, we represent the nodes of a $2k$-cycle as $u_0, u_1, \ldots, u_k = u_{-k}, u_{-(k-1)}, \ldots, u_{-1}$, so that the two paths leading from u_0 to u_k are given by $u_0, u_1, \ldots, u_k$ and $u_0, u_{-1}, \ldots, u_{-k} = u_k$. We use the same high-level framework for cycle-detection described above, but add the following ingredients.

Color coding [1]: to avoid small cycles leading to "false positives", we assign to each node $u \in V$ a random color, $c(u) \in \{-(k-1), \ldots, k\}$. Whenever we carry out a BFS, only nodes with color 0 may initiate a BFS, and a node u with color $c(u) = b \cdot i$, where $b \in \{-1, +1\}$ and $i \in \{0, \ldots, k-1\}$ is only allowed to forward BFS tokens to nodes with color $b \cdot (i+1)$. This ensures that if node u_k receives the BFS token of u_0 from its neighbors u_{k-1} and $u_{-(k-1)}$, then the token was carried along two paths that are node-disjoint, implying the presence of a simple $2k$-cycle. Since nodes choose their colors independently, the probability that a given cycle is "colored correctly" (i.e., $c(u_i) = i$ for each i) is $(1/2k)^{2k}$, a constant. We repeat each step of the algorithm sufficiently many times to ensure that if a $2k$-cycle exists, it is colored correctly at least once w.h.p, and then our color-coded BFS finds it.

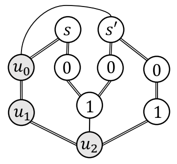
Limiting congestion: When we search for heavy cycles, we sample a uniformly random node s, check if it is part of a $2k$-cycle, and if not, we start a color-coded BFS from each 0-colored neighbor of s. There can be many such neighbors, potentially leading to congestion; however, we show that if the cycle is colored correctly, it suffices for each node with color $i \in [2k]$ to forward a constant number $T_k(i)$ of BFS tokens.

Our main concern is that the node s that we sampled is "bad", in the sense that it has many short node-disjoint paths to some cycle node u_i. If we sample such a "bad neighbor" of u_0, its 0-colored neighbors could initiate many BFS instances, which would then reach u_i and cause congestion. See Figure 2a for an illustration.

To bound the probability that we hit a "bad neighbor", we first rule out any neighbor of u_0 that itself participates in a $2k$-cycle. Next, we argue that if s has many node-disjoint paths to some cycle node u_i, such that the path nodes are colored $0, 1, \ldots, i$ (so that a



(a) A "bad neighbor" $s \in N(u_0)$.



(b) "Shared paths": the edges of the 10-cycle are indicated by double lines.

Figure 2 Illustrations for the proof sketch of the $2k$-cycle algorithm.

BFS can be initiated by the first path node and flow across the path), then we can charge these paths against the degree of u_i, as each path ends at a different neighbor of u_i. Since $\deg(u_0) \geq \deg(u_i)$, this means only a small constant fraction of u_0's neighbors have many such disjoint paths. When we sample a random node, we are unlikely to hit a "bad neighbor". (We are not worried about non-disjoint paths, as they do not contribute any "new" BFS tokens; see full version for details).

The problem with this argument is that if different neighbors of u_0 *share* paths to u_i, we might be overcounting when we charge each path against the degree of u_i. Our solution is to show that there is "not too much" sharing, otherwise a $2k$-cycle appears – and since we only consider neighbors of u_0 that are not on a $2k$-cycle, we know that this is impossible.

In Figure 2b, we show an example of one situation that must be ruled out (among others): consider $k = 5$ (i.e., 10-cycles), and suppose that two distinct neighbors $s, s' \in N(u_0)$ each have at least 10 node-disjoint paths with the "right colors", 0-1, to u_2. Suppose further that one of these paths is *shared*, as shown in the figure. In addition, node s' has at least one additional path (the rightmost path in the figure), which must exist because s' has at least 10 node-disjoint paths to u_2 (so at least one of these paths avoids all the other nodes shown in the figure). We see that there is a 10-cycle involving nodes s and s'; since we only consider neighbors of u_0 that do not themselves participate in a 10-cycle, this situation cannot arise.

6 Sketch of the $\Omega(n^{1/2+\alpha})$-Round Barrier on Lower Bounds for C_6-Detection in Congest

We prove that any lower bound on C_6-detection of the form $\Omega(n^{1/2+\alpha})$, where $\alpha > 0$, would imply new and powerful circuit lower bounds (namely, a superlinear lower bound on the number of wires in a circuit of polynomial depth). To show this, we first show that C_6-detection reduces to finding a directed triangle; this means that lower bounds on C_6-detection imply lower bounds on finding directed triangles. Next, we follow the same proof from [10] (which was given there for *undirected* triangles) to show that a lower bound of the form $\Omega(n^\alpha)$ on finding directed triangles would imply new circuit lower bounds.

The main idea of the reduction from 6-cycles to directed triangles is as follows. First, we eliminate all vertices with degree $\geq \sqrt{n}$, by calling the heavy-cycle subroutine of our new C_6-detection algorithm for $O(\sqrt{n}\log n)$ iterations. Each iteration requires constant time, and after $O(\sqrt{n}\log n)$ iterations, if there is a cycle containing a node with degree $> \sqrt{n}$, we will find one w.h.p.: w.h.p at least one iteration samples a node that is a neighbor of the cycle node that has degree $> \sqrt{n}$. Following this step, we "erase" all vertices with degree $\geq \sqrt{n}$ from the graph, together with all their edges. Let G' be the resulting graph.

Next, each vertex of G' chooses a random color $c(u) \in [6]$, and we simulate an algorithm for finding directed triangles on the directed graph $H = (V, E_H)$, where $(u,v) \in E_H$ iff $c(v) = c(u) + 2 \bmod 6$ and there is a path u, w, v in G' with $c(w) = c(v) + 1 \bmod 6$. It holds that any directed triangle in H corresponds to a well-colored simple 6-cycle in G', and vice-versa. Since G' can simulate one round in H using $\sqrt{n}$ rounds (as G' has maximum degree $\sqrt{n}$), if we can find a directed triangle in $O(n^{\alpha})$ rounds in H, then we can find a 6-cycle in $O(n^{1/2+\alpha})$ rounds in G'. Thus, intractability results for 6-cycles imply intractability for directed triangles, and following [10,11] we show this would imply new circuit lower bounds.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

3 Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC 2019)*, pages 74–83, 2019.

4 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.

5 Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. Sparse matrix multiplication and triangle listing in the congested clique model. *Theor. Comput. Sci.*, 809:45–60, 2020. doi:10.1016/j.tcs.2019.11.006.

6 Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 821–840, 2019.

7 Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC 2019)*, pages 66–73, 2019.

8 Danny Dolev, Christoph Lenzen, and Shir Peled. "tri, tri again": Finding triangles and small subgraphs in a distributed setting. In *Proceedings of the 26th International Symposium on Distributed Computing (DISC 2012)*, pages 195–209, 2012.

9 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 (PODC 2014)*, pages 367–376, 2014.

10 Talya Eden, Nimrod Fiat, Orr Fischer, Fabian Kuhn, and Rotem Oshman. Sublinear-time distributed algorithms for detecting small cliques and even cycles. In *Proceedings of the 33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146 of *LIPIcs*, pages 15:1–15:16, 2019.

11 Talya Eden, Nimrod Fiat, Orr Fischer, Fabian Kuhn, and Rotem Oshman. A note on the hardness of proving distributed lower bounds for triangle-freeness. https://www.cs.tau.ac.il/~roshman/papers/EFFKO19_triangle_note.pdf, 2020.

12 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 (SPAA 2018)*, pages 153–162, 2018. doi:10.1145/3210377.3210401.

13 Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1150–1162, 2012.

14 Zoltán Füredi and Miklós Simonovits. *The History of Degenerate (Bipartite) Extremal Graph Problems*, pages 169–264. Springer Berlin Heidelberg, 2013.

15 Juho Hirvonen, Joel Rybicki, Stefan Schmid, and Jukka Suomela. Large cuts with local algorithms on triangle-free graphs. *Electr. J. Comb.*, 24(4):P4.21, 2017. URL: `http://www.combinatorics.org/ojs/index.php/eljc/article/view/v24i4p21`.

16 Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing (PODC 2012)*, pages 355–364, 2012.

17 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978. `doi:10.1137/0207033`.

18 Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In *Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing (PODC 2017)*, pages 381–389, 2017.

19 Janne H. Korhonen and Joel Rybicki. Deterministic subgraph detection in broadcast CONGEST. In *Proceedings of the 21st International Conference on Principles of Distributed Systems (OPODIS 2017)*, pages 4:1–4:16, 2017.

20 François Le Gall. Further algebraic algorithms in the congested clique model and applications to graph-theoretic problems. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC 2016)*, pages 57–70, 2016.

21 Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing (PODC 2013)*, pages 42–50, 2013.

22 Andrzej Lingas and Eva-Marta Lundell. Efficient approximation algorithms for shortest cycles in undirected graphs. *Inf. Process. Lett.*, 109(10):493–498, 2009.

23 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA 2018)*, pages 405–414, 2018.

24 David Peleg, Liam Roditty, and Elad Tal. Distributed algorithms for network diameter and girth. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP 2012)*, pages 660–672, 2012.

25 Seth Pettie and Hsin-Hao Su. Distributed coloring algorithms for triangle-free graphs. *Information and Computation*, 243:263–280, 2015.

26 Liam Roditty and Roei Tov. Approximating the girth. *ACM Trans. Algorithms*, 9(2):15:1–15:13, 2013.

27 Liam Roditty and Virginia Vassilevska Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS 2011)*, pages 180–189, 2011.

28 Liam Roditty and Virginia Vassilevska Williams. Subquadratic time approximation algorithms for the girth. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 833–845, 2012.