Sublinear-time distributed algorithms for detecting small cliques and even cycles

Talya Eden1 · Nimrod Fiat2 · Orr Fischer2 · Fabian Kuhn3 · Rotem Oshman2

Received: 5 June 2020 / Accepted: 9 August 2021 / Published online: 26 November 2021
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2021

Abstract
In this paper we give sublinear-time distributed algorithms in the CONGEST model for finding or listing cliques and even-length cycles. We show for the first time that all copies of 4-cliques and 5-cliques in the network graph can be detected and listed in sublinear time, \(O(n^{5/6+o(1)})\) rounds and \(O(n^{73/75+o(1)})\) rounds, respectively. For even-length cycles, \(C_{2k}\), we give an improved sublinear-time algorithm, which exploits a new connection to extremal combinatorics. For example, for 6-cycles we improve the running time from \(\tilde{O}(n^{5/6})\) to \(\tilde{O}(n^{3/4})\) rounds. We also show two obstacles on proving lower bounds for \(C_{2k}\)-freeness: first, we use the new connection to extremal combinatorics to show that the current lower bound of \(\tilde{\Omega}(\sqrt{n})\) rounds for 6-cycle freeness cannot be improved using partition-based reductions from 2-party communication complexity, the technique by which all known lower bounds on subgraph detection have been proven to date. Second, we show that there is some fixed constant \(\delta \in (0, 1/2)\) such that for any \(k\), a lower bound of \(\Omega(n^{1/2+\delta})\) on \(C_{2k}\)-freeness would imply new lower bounds in circuit complexity. We use the same technique to show a barrier for proving any polynomial lower bound on triangle-freeness. For general subgraphs, it was shown by Fischer et al. that for any fixed \(k\), there exists a subgraph \(H\) of size \(k\) such that \(H\)-freeness requires \(\tilde{\Omega}(n^{2-\Theta(1/k)})\) rounds. It was left as an open problem whether this is tight, or whether some constant-sized subgraph requires truly quadratic time to detect. We show that in fact, for any subgraph \(H\) of constant size \(k\), the \(H\)-freeness problem can be solved in \(O(n^{2-\Theta(1/k)})\) rounds, nearly matching the lower bound.

Keywords Distributed computing · Subgraph freeness · Expander decomposition · CONGEST

1 Introduction
In the subgraph-freeness problem, a network must decide whether its communication graph contains a copy of some fixed subgraph \(H\) or not. If the network is \(H\)-free, then all nodes should accept, but if the graph contains a copy of \(H\), then at least one node should reject. The subgraph-freeness problem has received significant attention in the sequential world, and recently also in the distributed community. Other than being a fundamental graph problem, it has many application in other scientific fields such as biology and social sciences (e.g [39,41,42]).

From the theoretical perspective, distributed subgraph freeness is especially interesting because it is an extremely local problem: to solve \(H\)-freeness for a graph \(H\) of size \(k\),
each node only needs to examine its own $k$-hop neighborhood. However, it is known that in bandwidth-constrained networks (the CONGEST model), subgraph freeness cannot always be solved efficiently [13,14,17,26,29,34]. In fact, it is not even known which classes of subgraphs $H$ admit a sublinear-round distributed algorithm for $H$-freeness: some simple subgraphs, like constant sized odd-length cycles of length at least $5$ are known to require linear time [14], and for any constant $\epsilon > 0$ there are some constant sized subgraphs that require $\Omega(n^{2-\epsilon})$ time [17]. In contrast, for triangles [7] and for constant even-length cycles [17], sublinear-time algorithms are known.

In this work, we seek to improve our understanding of sublinear-time algorithms for two classes of subgraphs: cliques and even-length cycles. We also show that any constant-sized subgraph can be detected in sub-quadratic time.

**Small cliques.** We show for the first time that $4$-cliques and $5$-cliques can be detected in sublinear time; previously, no non-trivial algorithm for $K_4$-freeness or listing was known, and the same is true for $K_5$ (the trivial solution is simply to have each node send its entire neighborhood to all its neighbors, which requires $\Theta(n)$ rounds). In fact, our algorithm is even able to list all copies of $K_4$, $K_5$:

**Theorem 1** The problem of enumerating all $4$-cliques in the graph can be solved in $O(n^{5/6+\omega(1)})$ rounds, and all $5$-cliques can be enumerated in $O(n^{73/75+\omega(1)})$ rounds in CONGEST using randomization.

Our algorithm builds on a recent approach of [7], which decomposes the graph into well-connected clusters and a sparse set of edges, and uses this to give an elegant algorithm that enumerates all triangles. A triangle that has two or three edges in the sparse part of the graph can be found using the sparsity of this edge set, while a triangle that has two edges in a well-connected cluster can be found by the cluster nodes. Unlike triangles, however, a $4$-clique could be “split” between two clusters, with one edge in one cluster, another edge in another cluster, and the remaining edges crossing between the two clusters in the sparse part of the graph. With a $5$-clique the situation becomes even more complex. Thus, listing all $4$-cliques and $5$-cliques requires significant effort and new ideas beyond [7].

**Even-length cycles.** Turning our attention to even-length cycles, $C_{2k}$ for constant $k$, we give an improved sublinear-time algorithm for $C_{2k}$-freeness (it is known that odd-length cycles require linear time [14]). Our improved algorithm exploits a new connection to extremal combinatorics: we show that the Zarankiewicz number of the cycle $C_{2k}$, which is the maximum number of edges in a bipartite graph that does not contain $C_{2k}$, plays a role in testing $C_{2k}$-freeness, even for non-bipartite graphs. This allows us to modify the algorithm from [17] and improve its running time:

**Theorem 2** For any constant integer $k$, $C_{2k}$-freeness can be solved using randomization in at most $O_k(n^{1-2/(k^2-k+2)})$ rounds for odd $k \geq 3$, and at most $O_k(n^{1-2/(k^2-2k+4)})$ rounds for even $k \geq 4$.

For example, for $6$-cycles we improve the running time from $O(n^{5/6})$ (in [17]) to $O(n^{3/4})$ rounds.

We remark that while our $K_4$ and $K_5$ algorithms list all copies of $K_4$, $K_5$ in the graph, our algorithm for even-length cycles is only able to detect whether the graph contains a copy of $C_4$. In general, cliques seem like a more “local” type of subgraph than cycles: the presence of a clique implies that all its nodes can communicate with each other, which is obviously not true for cycles. We formalize this intuition by showing that short cycles really are different from small cliques—it is not possible to enumerate all of them in sublinear time:

**Theorem 3** Any randomized algorithm in CONGEST enumerating all $C_4$-copies in the graph requires $\tilde{O}(n)$ rounds.

Obstacles on proving lower bounds for even-length cycles and triangles. For any $k \geq 2$, the lower bound for $C_{2k}$-freeness has been “stuck” at $\Omega(\sqrt{n})$ for a long time [14,29]. We give two reasons for why improving this lower bound might be hard.

First, using the same connection to Zarankiewicz numbers, we show that reductions to two-party communication complexity of the type used to prove all known lower bounds on $H$-freeness for any subgraph $H$ [11,14,17,29], cannot be used to give a lower bound better than $\tilde{O}(\sqrt{n})$ on $C_6$-freeness. Following [11], which showed a similar result for cliques, we show:

**Theorem 4** Let $(V_A, V_B)$ be a partition of the vertices $V$ of the graph between two players, and assume that each player initially knows the edges incident to any node on its side of the graph $(V_A$ or $V_B$, respectively). If the cut $(V_A, V_B)$ contains $s$ edges, then there is a two-party protocol that communicates $O(\sqrt{n} \cdot s)$ bits and solves $C_6$-freeness.

Our protocol uses different ideas from the two-party protocol of [11]; to bound the number of edges the players need to send each other, we rely on results from extremal combinatorics. We note that while there are other types of reductions to communication complexity that are used to obtain CONGEST lower bounds, notably a “moving cut” reduction used in many global problems, such as the Minimum Spanning Tree problem [37], it is highly unclear how to use these type of reductions to obtain lower bounds for subgraph-freeness problems.

Next, we show that lower bounds on $C_{2k}$ are related to circuit lower bounds, in the following sense:

**Theorem 5** (Informal) There exists an absolute constant $c < 1$ such that for any $k \geq 3$, proving a lower bound of $\Omega(n^{1-c})$
on \(C_{2k}\)-freeness for randomized CONGEST algorithms would imply new lower bounds on high-depth circuits with constant fan-in and fan-out.

We show this by extending a connection shown in [14] between lower bounds for the Congested Clique and circuit complexity to high-conductance components.

We also prove a similar obstacle for triangle-freeness, where we show that a lower bound of the form \(Ω(n^\alpha)\) for any constant \(\alpha > 0\) would imply new circuit lower bounds. This perhaps explains why this problem has so far resisted attempts to prove strong lower bounds for it.

**General subgraphs.** It was shown in [17] that some subgraphs are very hard to detect: for any \(k \geq 2\), there exists a graph on \(O(k)\) vertices that requires \(\tilde{O}(n^{2−\Theta(1/k)})\) rounds to detect. It was left open whether this bound is tight, or whether the loss of \(1/k\) in the exponent is an artifact of the proof: the lower bound of [17] is shown by a reduction from two-party communication complexity, where the graph is partitioned into two parts, with a cut of size \(\Theta(n^{1/k})\) between them. This causes the lower bound to “lose” a factor of \(n^{1/k}\).

We show that, surprisingly, the factor \(n^{1/k}\) is not just an artifact of the proof:

**Theorem 6** For any constant integer \(k\) and any subgraph \(H\) of size \(k\), there exists a randomized algorithm for \(H\)-freeness in CONGEST with round complexity of \(\tilde{O}(n^{2−\frac{2}{\sqrt{k+1}}})\), for both induced and non-induced variants.

Thus, subgraph-freeness is not “maximally hard” in CONGEST: it does not require truly quadratic time.

1.1 Related work

The problems of subgraph-freeness and listing have been extensively studied in both the centralized and the distributed settings; for conciseness, we mention here the most directly related results.

In [7,8] randomized algorithms based on expander decompositions for listing all triangles were shown, culminating in an \(\tilde{O}(n^{1/3})\) round algorithm in the CONGEST model. This improved the previous algorithm of [27]. The algorithms of [7,8] finds a conductance decomposition of the graph, and then uses a routing result of [23,24] to quickly find all triangles contained or adjacent to some cluster in the decomposition. Our \(K_4\)-listing algorithm uses the decomposition from [7], albeit in a somewhat different manner from the way it is used in [7]. We also use the \(K_4\)-listing algorithm for the Congested Clique of [13] as a subroutine; [13] shows that for any subgraph \(H\) of size \(s\), all copies of \(H\) can be found in \(O(n^{1−2/\sqrt{s}})\) rounds in the Congested Clique.

To our knowledge, prior to our work, no sublinear-time \(K_k\)-freeness algorithm was known for any \(s \geq 4\) in CONGEST. However, in [3], it is shown that if the network contains an \(\epsilon\)-near-clique of linear size, then an \(\epsilon^3\)-near clique of linear size can be found in constant time.

For 4-cycles, \(C_4\), a nearly-tight bound of \(\Theta(\sqrt{n})\) was shown in [14]. The lower bound was extended to an \(\tilde{O}(\sqrt{n})\) lower bound for any even-length cycle \(C_{2k}\) in [29]. In the Congested Clique, an \(O(n^{0.158})\) round algorithm for \(C_k\)-detection was shown by [6] based on algebraic methods. The first sublinear-time algorithm for \(C_{2k}\)-freeness for \(k \geq 3\) was given in [17], and we improve this algorithm here. It is known that odd-length cycles require nearly-linear time to detect [14].

It is shown in [17] that some subgraphs of size \(O(k)\) require \(\tilde{O}(n^{2−\frac{1}{k}})\) rounds to detect (for any constant \(k \geq 2\)). There are algorithms for clique-freeness and cycle-freeness in related models, e.g., [5,6,13,16,18,19], but they are not directly relevant to our work, except as mentioned above.

2 Preliminaries

The CONGEST model. The CONGEST model is a synchronous network model, where computation proceeds in rounds. The network is modeled as a graph, \(G = (V,E)\). Each graph node \(v \in V\) initially knows its own neighborhood, denoted \(N(v)\). It is assumed that nodes have unique identifiers, which we conflate with \(V\). In each round, each node of the network may send \(O(\log n)\) bits on each of its edges, and these messages are received by neighbors in the current round.

Some of our algorithms rely on results from the Congested Clique model. As in CONGEST, we have an input graph \(G = (V,E)\), where each vertex \(V\) is a separate computing node, which initially knows its neighborhood. However, unlike CONGEST, in the Congested Clique, all the nodes can talk directly to each other: in each round, each node can send \(O(\log n)\) bits to every other node in \(V\), even if the edge between them is not in \(E\). The Congested Clique admits very efficient algorithms for many distributed tasks, and our algorithms build on a clique detection algorithm for this model [13] by simulating it in regular CONGEST.

The main problem we are concerned with in this paper is the following:

**Definition 1** (Subgraph freeness and enumeration) Fix a connected constant-sized graph \(H\).

In the \(H\)-freeness problem, the goal is to determine whether the input graph \(G\) contains a copy of \(H\) as a subgraph or not, namely, if there is a subset \(V' \subseteq V\) and a subset \(E' \subseteq E\) such that the graph \((V', E')\) is isomorphic to \(H\). If \(G\) is \(H\)-free, then all nodes should accept, but if \(G\) contains \(H\) as a subgraph, then at least one node should reject.
In the *H*-enumeration (or listing) problem, each node outputs a (possibly empty) set of copies of *H* in *G*, such that together the nodes output all copies of *H* in *G*.

In the *induced* *H*-freeness problem, the goal is to determine whether the input graph *G* has an induced subgraph that is isomorphic to *H*, namely, whether there is a subset *V*′ ⊂ *V* for which (*V*′, *E*(*V*′)) is isomorphic to *H*, where *E*(*V*′) denotes the set of edges with both endpoints in *V*′.

For a graph *G*, we let *V*(*G*) denote the vertex set of *G*, and *E*(*G*) denote the edges of *G*, and denote their sizes as *n* = |*V*(*G*)|, *m* = |*E*(*G*)| respectively. For a vertex *v*, we denote its degree by *d*(v). We denote the diameter of the graph by *D*.

The *arboricity* of the graph is defined as the minimum number of edge-disjoint forests required to cover the graph edges. A graph with *m* edges has arboricity at most *O*(*√m*) [9].

For functions *g*, *f*, we say that *g*(n) = *O*(*f*(n)) if for some constant *c* it holds that *g*(n) = *O*(*f*(n)·log⁴⁹ n).

### 2.1 Mixing time and conductance

For a graph *G* and a vertex *v* ∈ *V*, an *l*-length lazy random walk starting at vertex *v* is a random walk on vertices of *G* starting at *v*, defined as follows. The walk is performed for *l* steps, and for each step the walk remains in the current vertex with probability 1/2, and otherwise traverses to a uniformly random neighbor. Denote by *P*_l(*v*, *u*) the probability that a *l*-length lazy random walk starting at *v* ends at *u*.

**Definition 2** The mixing time *τ*ₘᵢₓ of of a graph *G* is defined as the minimum *l* such that

\[ \forall v, u \left| P_l(v, u) - \frac{d(u)}{2m} \right| \leq \frac{d(u)}{2mn} \]

Informally, the *mixing time* of a graph is the time required for a lazy random walk on the graph to approach its stationary distribution.

The mixing time of a graph is closely related to its conductance, defined next. For a subset *S* ⊆ *G*, let Vol(*S*) denote the sum of degrees of the vertices in *S*, Vol(*S*) = \( \sum_{v \in S} d(v) \).

**Definition 3** The conductance *Φ*(*S*) of a non-empty subset *S* ⊂ *V* is *Φ*(*S*) = \( \frac{|E(S, V \setminus S)|}{\min \{\text{Vol}(S), \text{Vol}(V \setminus S)\}} \), where *E*(S, *V* \ S) denotes the set of edges with one endpoint in *S* and one endpoint in *V* \ S. For *S* = ∅ or *S* = *V*, the conductance is defined as *Φ*(*S*) = 1. The conductance of *G* is *Φ*(*G*) = \( \min_{S \subseteq V} \{\Phi(S)\} \).

The following theorem by Jerrum and Sinclair [28] states the relation between the mixing time of a graph and its conductance.

**Theorem 7** (Corollary 2.3 in [28]) Let *τ*ₘᵢₓ(*G*) denote the mixing time of a graph *G*, and let *Φ*(*G*) denote its conductance. It holds that:

\[ \Theta \left( \frac{1}{\Phi(G)} \right) \leq \tau_{mix}(G) \leq \Theta \left( \frac{\log n}{\Phi^2(G)} \right) . \]

**Remark 1** It is a well known property that *D* = *O*(*log n/\( \Phi(G) \)) (see e.g. [10]). In particular, if the conductance of a graph is at least \( \Omega(1/\log n) \) (or equivalently, the mixing time is *O*(\log n)) then the diameter is at most *O*(log n).

### 3 Enumerating all 4-cliques in sublinear time

In this section we show how to find all copies of *K*₄ in the network graph in *O*(n⁵/(6+o(1))) rounds.

Throughout the section, we will use two parameters, \( \epsilon = 1/2 \), \( \delta = 5/6 \). Since some parts of the algorithm will be reused in later sections with different values for \( \epsilon \), \( \delta \), we leave our results parameterized.

On a very high level, the algorithm works as follows: first, we decompose the edges *E* of the graph into two sets, *E*₁ and *E*₂, by recursively applying a graph decomposition algorithm from [7]. The set *E*₁ is *oriented* (i.e., each edge is assigned a direction), and every node *v* has at most \( \tilde{O}(n^\delta) \) outgoing edges in *E*₁; therefore, cliques contained entirely in *E*₁ are easy to find in \( \tilde{O}(n^\delta) \) rounds, by simply having all nodes announce to all their outgoing edges in *E*₁ to all their neighbors.

The set *E*₂ is the edge-disjoint union of \( \tilde{O}(n^{1-\delta}) \) “very well-connected” clusters of nodes. On each such cluster, we can fairly efficiently simulate the Congested Clique algorithm for finding 4-cliques of [13]. However, we need to find all 4-cliques that have at least one edge in *E*₂ (since we already dispensed with all cliques contained entirely in *E*₁). This includes cliques \{u₀, u₁, u₂, u₃\} such that \{u₀, u₁\} ∈ *E*₂, so that nodes *u*₀, *u*₁ are together in some cluster *C*, but nodes *u*₂, *u*₃ are outside cluster *C*; while nodes *u*₀, *u*₁ together know about almost all edges of the clique, the edge \{u₂, u₃\} is not necessarily known to any node of *C*. If we want to find such cliques by simulating a Congested Clique algorithm on *C*, we must first “bring in” edges from outside *C*, so that the cluster nodes know about them and can use them in the simulation.

There can be \( \Theta(n^2) \) edges outside of *C*, and we cannot afford to have all of them sent to nodes of *C*, because this could require too much time: for example, suppose that outside the cluster *C* we have a large set *U* of nodes that are all connected to each other (i.e., a large clique), and each node of *U* is connected to some node of *C* by one edge. In this case, we have roughly |*U*|² edges that could participate in 4-
cliques with the nodes of the cluster $C$, but there are only $|U|$ edges over which we can send these edges to $C$, so sending all $|U|^2$ edges would take $|U|$ time. However, in this case, precisely because each node in $U$ has few neighbors in $C$, there are not many possible 4-cliques it can form with the nodes of $C$, and we can use this fact to quickly find any such 4-cliques.

Thus, our algorithm considers two types of “external edges”: if $u_2, u_3$ do not have many neighbors in $C$, then we can use this fact to efficiently find all cliques containing $u_2, u_3$ and two nodes from $C$. On the other hand, if either $u_2$ or $u_3$ does have many neighbors in $C$, they can quickly send their entire neighborhood to the nodes of $C$, by splitting their neighborhood and sending each part of it on a different edge to $C$. Then, the cluster nodes, having learned about the external edges, will use them in their simulation of the Congested Clique, and find any 4-clique containing $u_2, u_3$.

We now describe our algorithm in more detail. We begin by showing how we apply the expander decomposition from [7] in a slightly different manner than [7] uses it, and then how we find 4-cliques on the resulting decomposed graph.

Using our “diameter reduction” technique, given by Theorem 15, we assume w.l.o.g. that the diameter of $G$ is $O(polylog(n))$. This reduction is very similar to the diameter reduction in distributed property testing problems [16].

### 3.1 Expander decomposition

A main ingredient in the algorithm is the expander decomposition developed in [7], which partitions the edges $E$ of the graph into three sets, $E_m, E_s, E_r$, such that $E_m$ induces a set of “well-connected clusters”, $E_s$ induces a directed graph with low out-degree, and $|E_r| \leq |E|/6$. The well-connected clusters in $E_m$ each satisfy the following:

**Definition 4** ($n^\delta$-cluster) Let $c_1, c_2, c_3 > 0$ be constants. A subset $G' = (V', E')$ of $G$ is called an $n^\delta$-cluster (or simply “cluster” for short) if it satisfies the following conditions:

(1) It has mixing time at most $c_1 \cdot \log^{c_2}(n)$,
(2) Each vertex $v \in V'$ has degree at least $c_3 n^\delta$ in $E'$.

**Definition 5** ($n^\delta$-decomposition) Given a graph $G = (V, E)$, an $n^\delta$-decomposition of $G$ is a partition of $E$ into three edge sets, $E = E_m \cup E_s \cup E_r$, such that

(1) Each connected component of the graph $(V, E_m)$ is an $n^\delta$-cluster,
(2) The set $E_s$ is a disjoint union $E_s = \bigcup_{v \in V} E_{s,v}$, where for each $v \in V$, $E_{s,v}$ is a set of edges incident to $v$ such that $|E_{s,v}| = O(n^\delta)$. We call $E_{s,v}$ the edges oriented away from $v$ in $E_s$, and,
(3) $|E_r| \leq |E|/6$.

In the sequel we abuse notation slightly, by thinking of a cluster as both a set of edges and the set of nodes that belong to the cluster.

It is shown in [7] that an $n^\delta$-decomposition can be computed efficiently in CONGEST. “Computing” the decomposition means that each node learns which of edges are in $E_m$, $E_s$ or $E_r$, and also learns the orientation for its edges in $E_r$. [7, Theorem 1] Given a graph $G = (V, E)$ with $|V| = n$, we can find, w.h.p., an $n^\delta$-decomposition in $\tilde{O}(n^{1-\delta})$ rounds in the CONGEST model.

In [7], the algorithm finds triangles by

(a) Computing an $n^\delta$-decomposition,
(b) Finding triangles that include an edge in either $E_s$ or $E_m$,
(c) Recursing on $E_r$. (Edges are sometimes moved from $E_m$ to $E_r$, but not too many.)

The key property is that any triangle that includes at least one edge from $E_s$, $E_m$ can be found efficiently by the algorithm of [7]; then these edges can be deleted, and we can recurse on the remaining edges ($E_r$). In contrast, when it comes to 4-cliques, we do not know how to efficiently find a 4-clique that includes an edge from $E_s$ or $E_m$, if the remaining edges are all in $E_r$. Therefore we cannot simply delete $E_s$, $E_m$ and recurse (we would miss cliques whose edges are split between $E_s$, $E_m$ and $E_r$). Instead, before we begin searching for 4-cliques, we eliminate all the edges in $E_r$, by recursively applying the decomposition of Theorem 8 until no edges remain. We obtain the following decomposition of the graph.

**Lemma 1** Fix a graph $G = (V, E)$ with $|V| = n$ and diameter $D$. We can find, w.h.p., in $O(n^{1-\delta} + D)$ rounds, a decomposition of $E$ to $E = E_m \cup E_s$ satisfying the following conditions:

1. $E_m$ is the union of at most $s = O(\log n)$ sets, $E_m = \bigcup_{i=1}^s E_i$, where each $E_i$ is the union of $O(n^{1-\delta})$ vertex-disjoint $n^\delta$-clusters, $C_i^1, \ldots, C_i^q$.
   The set $E_m^i$ is called the $i$-th level of the decomposition.
   We say that a node $u$ belongs to cluster $C_i^j$ if at least one of $u$’s edges is in $C_i^j$.
2. Each level-$i$ cluster $C_i^j$ has a unique identifier, which is a pair of the form $(i, x)$ where $x \in [n^{1-\delta}]$, and a unique leader node, which is some node in the cluster. Each node $u$ knows the identifiers of all the clusters $C_i^j$ to which $u$ belongs, the leaders for those clusters, and it knows which of its edges belong to which clusters.
3. $E_s$ is the disjoint union $E_s = \bigcup_{v \in V} E_{s,v}$, such that $|E_{s,v}| = O(n^\delta \log n)$ for each $v \in V$. Each node $v$ knows $E_{s,v}$.
Proof. Such a decomposition can be obtained by recursively applying Theorem 8, as follows: in the first step, we apply Theorem 8, obtaining a partition $E = E_0^1 \cup E_s^1 \cup E_i^1$ of the edges. We then recurse on $E_s^1$, using Theorem 8 again to obtain a partition $E_i^1 = E_m^1 \cup E_s^2 \cup E_i^2$. We continue in this manner, at each step $i$ applying Theorem 8 to the remaining edges $E_{r-1}^i$ from the previous step, until no edges remain, $E_r^i = \emptyset$.

Since $E_r^i \leq E_{r-1}^i / 6$, this process terminates after $s = O(\log n)$ steps. We now define:

$$E_m = \bigcup_i E_m^i, \quad E_s = \bigcup_i E_s^i.$$ 

Let $C_1^i, \ldots, C_k^i$ be the connected components of $E_m^i$, by Theorem 8, these are $n^8$ clusters. Note that $C_1^i, \ldots, C_k^i$ are vertex- and edge-disjoint, and by definition,

$$E_m = \bigcup_i E_m^i = \bigcup_{i,j} E(C_{ij}^i).$$

Also, since $|E_s^i| = O(n^8)$ for each level $i$, and there are $O(\log n)$ levels, we have $|E_s| = O(n^8 \log n)$, as required.

Unique cluster leaders and identifiers are assigned as follows. We iterate through the levels, and for each level $i$, every cluster formed in step $i$ learns the ID of the smallest node in the cluster; this is done by having all nodes forward the smallest ID they have heard so far, along all edges in $E_{m}^i$, for $O(\text{polylog}n)$ rounds (recall that by Remark 1, the diameter of each cluster is $O(\text{polylog}n)$). Let $u_1^i, \ldots, u_{k_i}^i$ be the smallest nodes in each of the clusters $C_1^i, \ldots, C_{k_i}^i$ of level $i$. These are the cluster leaders for step $i$.

Next, we disseminate all the IDs $u_1^i, \ldots, u_{k_i}^i$ of the leaders throughout the network, using pipelining [30]. This requires $O(k_i + D) = O(n^{1/3} + D)$ rounds, because there are at most $k_i \leq n^{1-3\delta}$ clusters (as each cluster has minimum degree $n^{8\delta}$). Finally, each cluster $C_{ij}^i$ is assigned the identifier $(i, r_{ij}^i)$, where $r_{ij}^i$ is the rank of the leader $u_{ij}^i$ among step $i$’s leaders, $u_1^i, \ldots, u_{k_i}^i$, sorted as natural numbers. Each node can compute the ID of the level-$i$ cluster it belongs to (if any), because it knows the IDs of all the level-$i$ leaders and of its own cluster’s level-$i$ leader.

The diameter reduction technique described in Sect. 6 allows us to assume w.l.o.g. that the diameter of the network is $D = O(\text{polylog}(n))$. Thus, in $O(n^{1-3\delta})$ rounds, we can select the smallest node in each cluster, and disseminate the IDs of these nodes throughout the network.

3.2 Finding 4-cliques

In the remainder of the section we describe our 4-clique listing algorithm, analyze its round complexity and prove its correctness.

We begin by computing the decomposition from Lemma 1. Next, we look for cliques entirely contained in $E_s$, the “low-degree” part of the graph. Then we turn to the harder task, finding cliques that include at least one edge of $E_m$ (i.e., a cluster edge). As we explained above, we divide these cliques into two types: those that have two nodes external to the cluster with few neighbors in the cluster, and those that do not.

Step 1: Finding cliques contained in $E_s$. To find 4-cliques contained entirely in $E_s$, we simply have each node $v$ send $E_s,v$ to all its neighbors. This requires $O(n^8 \log n)$ rounds, because $|E_s,v| = O(n^8 \log n)$ for each $v \in V$. Then, any node that sees a 4-clique outputs it.

Step 2: Finding cliques containing an edge from $E_m$.

Next, we search for copies of $K_4$ that have at least one edge in some cluster. We divide these cliques into two types. Let $\epsilon \in (0,1)$ be a parameter (as we said above, in this section we will use $\epsilon = 1/2$, but the next section re-uses some of the machinery we develop here with a different $\epsilon$).

For a cluster $C$ and a node $v$, let $MC(v) = N(v) \cap C$ denote the neighbors of vertex $v$ that belong to $C$. We say that node $v$ is $C$-light if

(a) $v \notin C$, and
(b) $|MC(v)| \leq n^\epsilon$, that is, $v$ has at most $n^\epsilon$ neighbors in $C$.

If $v$ is not $C$-light, then we say that $v$ is $C$-heavy.

Now let $H$ be a copy of $K_4$ that has at least one edge in $E_m$ (that is, in some cluster). We say that $H$ is light if $H$ contains at least two nodes that are $C$-light for some cluster $C$, and the two other nodes are contained in $C$. Otherwise, we say that $H$ is heavy.

Step 2(a): Finding light cliques. To find cliques containing at least two nodes that are light with respect to some cluster, we iterate through the clusters sequentially, in lexicographic order of their cluster IDs. For each cluster $C$, all nodes that belong to $C$ (i.e., have at least one edge in $C$) announce this fact to their neighbors. Next, each node $v$ that is $C$-light sends $MC(v)$ to all its neighbors; this requires $n^\epsilon$ rounds, as $v$ is $C$-light.

Upon receiving $MC(u)$ from each $C$-light neighbor $u \in N(v)$, node $v$ forms a list of “candidates” — triplets of nodes that may complete a $K_4$ with $v$:

$$L_v = \{ \{u, c_1, c_2\} \mid u \in N(v) \text{ is } C - \text{light}, \quad c_1, c_2 \in C, \quad c_1, c_2 \in MC(u) \cap MC(v) \}.$$
For each candidate \( \{u, c_1, c_2\} \), node \( v \) already knows that all edges of the 4-clique on \( \{v, u, c_1, c_2\} \) are present, except for edge \( \{c_1, c_2\} \), which may or may not be present. To check, node \( v \) goes through its candidates, and sends each cluster neighbor \( c_1 \in C \) a list of “edge queries”:

\[
Q_{v, c_1} = \{ c_1, c_2 \} | \exists u \in N(v) : \{ u, c_1, c_2 \} \in L_v \}
\]

the list of potential edges of \( c_1 \) that, if present, would complete a 4-clique. There are at most \( n^4 \) such edges: by definition of \( L_v \), if \( \{c_1, c_2\} \) is sent to \( c_1 \), then \( c_2 \in M_C(v) \), but \( |M_C(v)| \leq n^\epsilon \) (as \( v \) is \( C \)-light). Node \( c_1 \) responds with the subset \( (\{c_1\} \times N(c_1)) \cap Q_{v, c_1} \) of edges that are actually present, and node \( v \) outputs the 4-cliques it has found.

**Step 2(b): Finding heavy cliques.** Finally, we look for 4-cliques where at least one edge is in a cluster, and the other two nodes are not light w.r.t. that cluster; they might be in the cluster, or they might be outside it but have many neighbors in the cluster.

For a cluster \( C \), let

\[
F(C) = \{ \{u, v\} | u \in C \text{ or } u \text{ is } C \text{ - heavy} \}
\]

be the set of all edges incident to a node in \( C \) or to some \( C \)-heavy node. We iterate through the levels \( i = 1, \ldots, s \) of the decomposition; our goal is to find 4-cliques contained entirely in \( F(C^i) \), for all the level-\( i \) clusters, \( C^1, \ldots, C^i \).

To do this, we run in parallel a procedure on each level-\( i \) cluster \( C \), in which we have the cluster nodes of each cluster simulate the execution of the Congested Clique algorithm for \( K_4 \) enumeration from [13] on the \( n \)-vertex graph \((V, F(C))\). This part of the algorithm is carried out in four steps:

I. **Pull**: the nodes of the cluster \( C \) “pull” the edges of \( F(C) \) from nodes outside \( C \), such that every edge of \( F(C) \) is learned by at least one node in \( C \).

II. **Partition**: in this step, we compute a partition \( \{V_e\}_{e \in C} \) of \( V \), which is roughly balanced (i.e., \( |V_{c_1}| \approx |V_{c_2}| \) for each \( c_1, c_2 \in C \)). Each node \( c \in C \) will be responsible for simulating the nodes in \( V_e \).

III. **Shuffle**: the cluster nodes shuffle the edges of \( F(C) \) between themselves, so that each node \( c \in C \) learns the \( F(C) \)-neighborhood \( N_{F(C)}(v) \) for each node \( v \in V_e \) it needs to simulate. This is carried out by using the routing algorithm from [23].

IV. **Simulate**: the nodes of \( C \) simulate an \( n \)-vertex Congested Clique, and run the \( K_4 \)-enumeration algorithm of [13] on the graph \((V, F(C))\) to list all the 4-cliques in \( F(C) \).

**Remark 2** It is worth noting that [7] gives an extension of the Congested Clique algorithm of [13] to any graph with a low mixing time. This is unfortunately unsuited for our purposes, because in our case we do not run the algorithm only on edges incident to cluster nodes, but instead on a potentially much larger set, \( F(C) \). In addition, we simulate an Congested Clique of size \( n \) on an \( n^\epsilon \)-cluster, while in [7] the clusters do not need to simulate any external nodes.

As we said, these four steps are carried out in parallel for all the level-\( i \) clusters. Other than the first step, the remaining steps involve only intra-cluster communication (i.e., communication between nodes of the same cluster over the edges belonging to the cluster). Because the level-\( i \) clusters are vertex-disjoint components, we incur no extra congestion from simulating all level-\( i \) clusters in parallel; and since there are only \( O(\log n) \) levels, we can easily afford to go through the levels one after the other.

We elaborate on each step below.

**Pull.** As we explained, our goal in this step is to “pull” the edges \( F(C) \) into the cluster \( C \), such that each edge in \( F(C) \) will be known to at least one node in \( C \).

Each “internal” cluster edge \( \{u, v\} \in C \) is already known to its endpoints \( u, v \in C \) (recall that a node is said to belong to a cluster it has at least one edge in the cluster). Thus, we need only worry about the “outside” edges, \( F(C) \setminus C \), the edges incident to a heavy node w.r.t \( C \) which are not contained inside the cluster.

To pull these edges into \( C \), each \( C \)-heavy node \( v \) that is not in \( C \) partitions its edges arbitrarily into \( |M_C(v)| \) sets, each of size at most

\[
\frac{|N(v)|}{|M_C(v)|} \leq \left\lceil \frac{n}{n^\epsilon} \right\rceil \leq n^{1-\epsilon} + 1.
\]

and assigns to each such set to a cluster neighbor \( c \in M_C(v) \) in a one-to-one manner. We denote the set assigned to a cluster neighbor \( c \) by \( S_{v \rightarrow c} \). Then, in parallel for all \( c \in M_C(v) \), node \( v \) sends \( S_{v \rightarrow c} \) to node \( c \). This requires \( O(n^{1-\epsilon}) \) rounds.

For a node \( c \in C \), let

\[
F(c) = \{ \{c, u\} | u \in N(c) \} \cup \{ \{u, v\} | c \in M_C(v) \text{ and } u \in S_{v \rightarrow c} \}
\]

be the edges of \( F(C) \) that are learned by (or already known to) node \( c \). Following the Pull step, each edge in \( F(C) \) is known to at least one node, and at most two nodes, in \( C \):

**Observation 9** We have \( F(C) = \bigcup_{c \in C} F(c) \).

We define the initial load of node \( c \in C \) to be \( |F(c)| \), the number of \( F(C) \)-edges node \( c \) knows (including its own edges).

**Observation 10** For each \( c \in C \) we have \( |F(c)| \leq 2n^{2-\epsilon} \).

**Proof** Each cluster node \( c \in C \) receives at most \( |S_{v \rightarrow c}| \leq n^{1-\epsilon} \) edges from each heavy neighbor \( v \), and since \( c \) has
at most $n$ heavy neighbors, it receives at most $n^{2-\epsilon}$ edges in total. In addition to the edges it receives from heavy neighbors, node $c$ has up to $n - 1$ edges of its own, and so $|E(c)| \leq n^{2-\epsilon} + n - 1 < 2n^{2-\epsilon}$. \qed

**Partition.** The nodes of $C$ must now partition all the graph nodes $V$ among themselves, in a roughly-balanced way, quickly and without a lot of communication. Every cluster node needs to learn the entire partition, not just its own part, so that it knows which node of $V$ will be simulated by whom. Ideally, we would assign each node of $V$ to a uniformly random node in $C$, but representing such a random assignment could require up to $n \log n$ bits (depending on the size of $C$), so disseminating it to the nodes of $C$ would require a lot of communication. Instead, we use a family of $O(\log n)$-wise independent hash functions to ensure a relatively balanced partition, allowing us to represent the entire partition using only $O(\log^2 n)$ bits.

For convenience, we use only $n^3$ nodes in $C$, which we call the active nodes, to carry out the simulation; the cluster leader selects these nodes by choosing $n^3$ of its neighbors, $u_1, \ldots, u_{n^3}$ (recall that the minimum degree in each $n^3$-cluster is at least $n^3$, so the leader has enough neighbors.) The leader also selects an $\ell$-wise independent hash function $f : U \to \{n^3\}$, where $\ell = O(\log n)$, and $U$ is the domain from which IDs for the graph are drawn. Then the leader disseminates the assignment of active nodes, $\{(i, u_i)\}_{i=1,\ldots, n^3}$, and the $O(\log^2 n)$-bit representation of $f$ to all the cluster nodes. Using pipelining [30], this requires $\tilde{O}(n^3 + \text{diam}(C))$ rounds, where diam$(C) = \text{polylog}(n)$ is the diameter of the cluster $C$. (By Remark 1, as $C$ has polylogarithmic mixing time, its diameter is also polylogarithmic.)

Now let $V_{u_i} = \{v \mid f(v) = i\}$ be the set of nodes that active node $u_i$ will simulate. Using a concentration result from [38], for $O(\log n)$-wise independent random variables, we obtain:

**Lemma 2** Let $\ell = 4 \log n$, and assume that $f$ is $\ell$-wise independent. Then with probability at least $1 - 1/n$, we have $|V_{u_i}| \leq 2n^{1-\delta}$ for each active node $u_i$.

We defer the proof of this technical lemma until the end of the section.

**Shuffle and Simulate.** Next, we must route the edges of $F(C)$, from the cluster nodes that know them initially to the nodes that need them for the simulation. Then we run a simulation of the algorithm of [13], where each node in $C$ simulates at most $O(n^{1-\delta})$ nodes of $G$. For both purposes we use the routing scheme of [23], which allows us to deliver roughly $n^3$ messages from each cluster node to other cluster nodes in $n^{o(1)}$ rounds.

There are some technical details involved in the simulation, because we must ensure that nodes do not try to send or receive too many messages at once. Ultimately, we show the following result (which will also be used in the next section):

**Lemma 3** For a constant $0 < \epsilon \leq 1$, suppose an edge set $E'$ is partitioned between the nodes of an $n^3$-cluster $C$, so that each node $u \in C$ initially knows a subset $E'_u$ of size at most $O(n^{2-\epsilon})$. Then a simulation of $t$ rounds of the Congested Clique on $G' = (V', E')$ can be performed in $O(n^{2\cdot(2-\epsilon)} + (t+1) \cdot n^{2-2\delta} \cdot n^{o(1)})$ rounds, with success probability $1 - \frac{1}{n^3}$.

The proof is again quite technical, and it is deferred to the end of the section.

Using the simulation from Lemma 3, each cluster simulates the $K_4$ enumeration algorithm of [13]. In the full Congested Clique, this algorithm runs in $O(\sqrt{n})$ rounds, but since we are using clusters that simulate one round of the Congested Clique in roughly $n^{2-2\delta}$ rounds, our running time will be $O(n^{2-2\delta+1/2})$.

### 3.3 Correctness and running time of the algorithm

The algorithm’s success hinges on the success of several steps: the decomposition of Lemma 1, and for each level of the decomposition, the Partition step (Lemma 2) and the Simulation step (Lemma 3). Since every step succeeds with probability at least $\geq 1 - \frac{1}{n^3}$, by union bound all steps succeed w.h.p.. Let $S$ be the event that all these steps succeeded.

**Lemma 4** Conditioned on $S$, any copy of $K_4$ in $G$ is outputted by at least one node.

**Proof** First, consider 4-cliques contained entirely in $E_5$. These cliques are detected in Step (1): if there is a 4-clique on vertices $u_1, \ldots, u_4$ whose edges are all in $E_5$, then for each $i \neq j$, either $\{u_i, u_j\} \in E_{s,u_i}$ or $\{u_j, u_i\} \in E_{s,u_j}$, so either $u_i$ or $u_j$ will send $\{u_i, u_j\}$ to all its neighbors — including the other three clique nodes. It follows that each node of the 4-clique will learn all edges of the 4-clique and will output it.

Next, let $H$ be a 4-clique that is not contained in $E_5$; then $H$ has at least one edge $\{c_1, c_2\}$ in some cluster $C$. Let $v_1, v_2$ be the other two nodes of $H$. If $v_1$ and $v_2$ are both $C$-light (i.e., if $H$ is a light clique), then in Step 2(a), when we reach cluster $C$ in the lexicographic ordering, $v_1$ and $v_2$ will send each other the edges $\{v_1, c_1\}, \{v_1, c_2\}$ (sent from $v_1$ to $v_2$) and $\{v_2, c_1\}, \{v_2, c_2\}$ (sent from $v_2$ to $v_1$). Then, $u_1$ will form the candidate $\{v_2, c_1, c_2\}$, causing it to query $c_1$ and ask it whether edge $\{c_1, c_2\}$ is present. Since this edge is present, $c_1$ will send back a positive answer, and at this point node $v_1$ will output $H$, as it will now know about the presence of all of its edges.

Finally, let $H$ be a heavy 4-clique: it has some edge $\{c_1, c_2\}$ in a cluster $C$, and of the other two nodes of $H$, at least one is $C$-heavy. Call this node $u$, and let $v$ be the fourth node
of $H$. In Step 2(b), in the Pull step corresponding to cluster $C$’s level, node $u$ sends the edge $\{u, v\}$ to some $C$-neighbor $w$ (i.e., $\{u, v\} \in S_{u \rightarrow w}$), and at this point each edge of $H$ is known to at least one node of $C$ (since $c_1$, $c_2$ are themselves in $C$). Conditioned on $S$, the Partition and Simulation steps succeed in simulating the 4-clique detection algorithm of [13] (which is deterministic), and $H$ is found by some node of $C$. 

$\square$

**Lemma 5** The running time of the $K_4$ algorithm is $\tilde{O}(n^{5/6+o(1)})$ rounds.

**Proof** Recall that we set $\epsilon = 1/2$, $\delta = 5/6$. Each step of the algorithm has the following running time:

- Computing the decomposition of Lemma 1 requires $\tilde{O}(n^{1-\delta}) = \tilde{O}(n^{1/6})$ rounds.
- Step 1, where we find cliques contained in $E_s$, requires $\tilde{O}(n^\delta) = \tilde{O}(n^{5/6})$ rounds.
- Step 2(a), where we find light cliques, requires $O(n^\delta) = O(n^{1/2})$ rounds per cluster. We go through the clusters sequentially, and there are at most $O(n^{1-\delta}) = O(n^{1/6})$ clusters (because each cluster has minimum degree and therefore size $\Omega(n^\delta)$). Therefore this step requires $O(n^{1-\delta+\epsilon}) = O(n^{5/3})$ rounds.
- In Step 2(b), we go through the $O(\log n)$ levels of the decomposition, and operate on all the clusters in each level in parallel. For each level, we spend $O(n^{1-\epsilon})$ rounds having heavy nodes send their neighbors into every cluster for which they are heavy (by partitioning their edges among their neighbors in that cluster). The Partition step requires polylogarithmic time. Then we spend $O((n^{2-\delta-\epsilon} + n^{2-2\delta} + n^{2-2\delta+1/2}) \cdot n^{o(1)}) = O((n^{2/3} + n^{1/3} + n^{5/6}) \cdot n^{o(1)}) = O(n^{5/6+o(1)})$ rounds for the Shuffle and Simulations steps.

$\square$

### 3.4 Proof of Lemma 2

To analyze the partition used in the simulation, we rely on the following concentration bound:

**Theorem 11** ([38], Theorem 5(II)(b)) If $X$ is the sum of $k$-wise independent random variables, each of which is confined to the interval $[0, 1]$ and has $\mu = E[X]$, then for $\alpha \geq 1$ and assuming $k \leq \lceil \alpha \mu e^{-1/3} \rceil$, 

$$Pr(|X - \mu| \geq \alpha \mu) \leq e^{-|k/2|}.$$ 

**Proof of Lemma 2** Fix an active node $u$, and for each $v \in V$, let $X_{i,v}$ be an indicator for the event that $f(v) = i$. Then $|V_u| = \sum_{v \in V} X_{i,v}$. We have

$$E[|V_u|] = \sum_{v \in V} E[X_{i,v}] = n \cdot \frac{1}{n^\delta} = n^{1-\delta}.$$ 

By the tail bound given in Theorem 11,

$$Pr\left[|V_u| > 2n^{1-\delta}\right] < e^{-\left[\ell/2\right]} = e^{-2\log n} = \frac{1}{n^2}.$$ 

By union bound, the probability that any node in $C$ has $|V_u| > 2n^{1-\delta}$ is bounded by $1 - 1/n$. $\square$

### 3.5 Proof of Lemma 3

In the proof of the lemma, we use the routing algorithm of [23] for graphs with a low mixing time.

**Definition 6** An $(s, t)$-routing demand for $C$ is a collection of sets of messages, $\{M_u \rightarrow v\}_{u \in C}$, where $M_u \rightarrow v \subseteq \{0, 1\}^\ell$ is a set of messages that node $u$ wants to send to node $v$, and we have, for each $u \in C$,

$$\sum_{v \in C} |M_u \rightarrow v| \leq s, \text{ and } \sum_{v \in C} |M \rightarrow u_v| \leq t.$$ 

That is, each node is the sender of at most $s$ messages, and the target of at most $t$ messages. The length $\ell$ of each message is assumed to be $\ell = O(\log n)$, so that a single message can be sent in one round.

**Theorem 12** ([23], Theorem 1.2) For any $\alpha > 0$, $A \left( d_G(v) \cdot 2^{O(\sqrt{\log n \log \log n})}, d_G(v) \cdot 2^{O(\sqrt{\log n \log \log n})} \right)$ routing demand instance can be solved in $\tau_{mix} \cdot 2^{O(\sqrt{\log n \log \log n})}$ rounds, with probability $1 - \frac{1}{n^c}$, where $\tau_{mix}$ is the mixing time of $G$, and $d_G(v)$ is the degree of $v$ in $G$.

**Lemma 6** Let $\alpha, \beta \in (0, 2]$. For any $c > 0$, an $(n^{\alpha}, n^{\beta})$-routing demand can be routed on an $n^{\delta}$-cluster $C$ in $O(n^{\max\{\alpha, \beta\} - \delta + o(1)})$ rounds, with success probability $1 - 1/n^c$.

**Proof** In the cluster $C$, every node has degree at least $n^{\delta}$, and the mixing time is $\Omega(\text{polylog}(n))$; therefore, applying Theorem 12, we see that a set of routing requests where each node has $n^{\delta}$ messages to send and receive can be fulfilled, w.h.p., in $O(n^{o(1)})$ rounds.

Given an $(n^{\alpha}, n^{\beta})$-routing demand, let $M_v$ be the set of messages node $v$ needs to send. We assume w.l.o.g. that $|M_v| \geq n^\alpha/2$ (otherwise, node $v$ can pad $M_v$ by adding empty messages).
The naïve strategy to route all the messages would be to have each cluster node $v$ divide its messages $M_v$ into batches of $n^\delta$ messages, and try to route each message to its destination; however, the resulting set of routing requests would not necessarily obey the requirement of at most $n^\delta$ messages received at every node. Instead, for $N = 16cn^{\max(\alpha,\beta) - \delta} \log n$

iterations, in the $i$'th iteration each vertex $v$ selects a random subset $S_v^i \subseteq M_v$, where each message in $M_v$ is included in $S_v^i$ with probability

$$p = \frac{16c \log n}{N} = \frac{1}{n^{\max(\alpha,\beta) - \delta}}.$$  

Then, all nodes attempt to route their selected messages $\{S_v^i\}_{i \in C}$ to their destination, using Theorem 12.

We say that an iteration $i$ succeeds if

(a) All messages selected are delivered to their destinations,
(b) Each node $v$ sends at most $n^\delta$ messages, that is, $|S_v^i| \leq n^\delta$.
(c) Each node $v$ is the target of at most $n^\delta$ messages selected, that is,

$$\sum_{u \in C} |\{m \in S_v^i | u \text{ is } m \text{'s target}\}| \leq n^\delta.$$  

Now, let us analyze the probability that in $N$ iterations all iterations succeed.

First, consider the third condition: that no node is the sender of too many selected messages. Each node $v$ has $|M_v| \leq n^\alpha$, and every message in $M_v$ is selected with probability $p = 1/2n^{\max(\alpha,\beta) - \delta}$, so

$$E[|S_v^i|] \leq n^\alpha \cdot \frac{1}{2n^{\max(\alpha,\beta) - \delta}} \leq \frac{1}{2} n^\delta.$$  

By Chernoff, assuming sufficiently large $n$,

$$\Pr[|S_v^i| \geq \delta] \leq \frac{1}{1000n^{\alpha + \delta}}.$$  

Similarly, since every node is the target of at most $n^\delta$ messages in total, and since $n^\delta \cdot p \leq \frac{n^\delta}{2}$, the probability that in a given iteration a given node is the target of more than $n^\delta$ messages is bounded by $\frac{1}{1000n^{\alpha + \delta}}$. For sufficiently large $n$, by union bound over all $n$ nodes and all $N$ iterations, the probability that the second or third conditions fail to hold in any iteration is bounded by

$$\frac{2}{1000n^{\alpha + \delta}} \cdot n \cdot 16n^{\max(\alpha,\beta) - \delta} \log n < \frac{1}{3} \frac{n^3 \log n}{n^{\alpha + \delta}} < \frac{1}{4n^\delta}.$$  

For each node message $m \in \bigcup_{v \in V} M_v$, let $I_m$ be the number of times $m$ is sent.

$$E[I_m] = p \cdot N = \frac{16c^2 N \log n}{N} = 16c^2 \log n.$$  

By Chernoff,

$$\Pr[I_m < 1] < e^{(15/16) \log n/2} \leq \frac{1}{4n^{\alpha + \delta}}.$$  

As there are at most $n^3$ messages in total, all messages are sent at least once with probability at least $1 - \frac{1}{4n^{\alpha + \delta}}$. Therefore, by union bound, all iterations succeed, and all messages are sent with probability at least $1 - \frac{1}{4n^{\alpha + \delta}}$. \hfill \Box

Lemma 7 Let $C$ be an $n^\delta$-cluster. For a constant $0 < \epsilon \leq 1$, suppose an edge set $E'$ is partitioned between the nodes of $C$, so that each node $u \in C$ initially knows a subset $E'_u \subseteq E'$ of size at most $O(n^{2-\epsilon})$. Let $\{V_u\}_{u \in C}$ be a partition of $V$, such that $|V_u| \leq 2n^{\delta}$ for each $u \in C$, and let $N'(u)$ denote the neighborhood of $V_u$ with respect to $E'$. Then with probability at least $1 - \frac{1}{n}$ in $O((n^2 - 2\delta) \cdot n^\delta)$ rounds, every cluster node $u \in C$ can learn $N'(u)$.

Proof In the cluster $C$, every node has degree at least $n^\delta$, and the mixing time is $O(\text{polylog}(n))$; Therefore, by applying Lemma 6, each node $v$ sends each edge $(u, v) \in E'_v$ to $V_u$. As each node sends at most $O(n^{2-\epsilon} + n) = O(n^{2-\epsilon})$ messages, and each $u$ is the recipient of at most $O(n \cdot |V_u|) = O(n^{2-\delta})$ messages, number of rounds taken for the routing to finish is $O(n^{2-\max(\delta, \epsilon)} / n^\delta \cdot n^\delta) = O((n^2 - 2\delta + 2\delta) \cdot n^\delta)$. \hfill \Box

Finally, once each cluster node $u \in C$ has learned the edges in $F(C)$ incident to all nodes in $V_u$, we can simulate the Congested Clique and find any 4-cliques contained in $F(C)$.

After the procedure of Lemma 7, each vertex $u_t$ learns $N(V_{u_t})$, where each $|V_{u_t}| \leq 2n^{\delta-\delta}$. Following this, a round in the congested clique on $G'$ can be simulated in $O(n^{2-2\delta})$ rounds by using Lemma 6, with $\alpha = \beta = 2 - \delta$ (i.e $n \cdot O(n^{2-\delta})$ sent and received messages per node).

4 Enumerating all 5-cliques in sublinear time

In this section we show how to find all copies of $K_5$ in the network graph in $O(n^{73/75+o(1)})$ rounds in CONGEST with high probability.

The algorithm is very similar to the $K_4$ enumeration algorithm from Sect. 3: it begins by decomposing the network into dense well-connected clusters, and then searches for cliques using different strategies, depending on whether the vertices of the clique are entirely contained, partially contained, or entirely outside some dense cluster. The “hard case” with
Unfortunately, there is an error in the analysis of the conductance of $K_4$, is when two vertices $v_1, v_2$ are in some dense cluster $C_1$, and outside $C_1$ we have two vertices $v_3, v_4$ which have few edges to $C_1$ ("light vertices") and one vertex $v_5$ with many edges to $C_1$ (a "heavy vertex"). Despite the fact that the two vertices $v_1, v_2$ inside $C_1$ learn the edge $(v_3, v_4)$ at Step (1), they might not be able to propagate the light-light edges it learned inside the cluster in Step (2) due to having many light vertices adjacent to them. This differs from the $K_4$-freeness, in which if both the outer vertices are light, the $K_4$-copy is detected at Step (1). To handle this difficulty, we use a different decomposition, such that the number of “remainder” edges is a sub-constant fraction of the total number of edges in the graph. This implies a (slightly) sublinear bound on the arboricity of the edges outside any given cluster.

Throughout the section, we will use three parameters, $\epsilon = 28/75$, $\delta = 73/75$, $\gamma = 2/75$.

4.1 The decomposition

Our $K_5$-listing algorithm begins by decomposing the graph into well-connected components, but this time we use a slightly different decomposition: instead of recursively applying the decomposition from [7], we use a variant of the decomposition, introduced in [12], and we use it only once (not recursively). This yields a decomposition into vertex-disjoint components (unlike in Sect. 3), which has the following properties:

Definition 7 ([12], Definition 4.312) For parameters $0 < \delta < 1$ and $0 \leq \gamma \leq 1/2$, a tripartition $T_{\delta, \gamma}(G)$ of a simple, unweighted, undirected graph $G = (V, E)$ is a partition of the edge set $E$ into $E = \{E_m, E_s, E_r\}$ satisfying the following:

1. Each connected component induced by $E_m$ has mixing time $\tilde{O}(n^{12\gamma})$, and has minimum degree at least $c\nu^\delta$ for some constant $c > 0$.
2. The subgraph induced by $E_s$ has arboricity $O(n^\delta)$.
3. $|E_r| \leq c' m^{1-\gamma}$ for some constant $c' > 0$, and each edge of $E_r$ has endpoints in different connected components in the subgraph induced by the edge set $E_m$.

As in Sect. 3, we refer to the connected components induced by $E_m$ as clusters. Also, it is convenient to view the set $E_s$ as the union $E_s = \bigcup_{v \in V} E_{s,v}$, where $E_{s,v}$ are the edges of $E_s$ that are incident to node $v$. We think of the edges in $E_{s,v}$ as oriented away from $v$.

We say that an algorithm computes a tripartition $T_{\delta, \gamma}(G)$ if at the end of the algorithm, every node $v \in V$ knows which of its incident edges belongs to each of the sets $E_m, E_s, E_r$.

It is shown in [12] that a tripartition can be computed quickly:

Theorem 13 ([12], Theorem 4.4) For any $0 < \delta < 1$ and $0 \leq \gamma \leq 1/2$, there is an algorithm in CONGEST that computes a tripartition $T_{\delta, \gamma}(G)$ in $O(n^{1-\delta+20\gamma})$ rounds w.h.p.

4.2 Properties of low-arboricity graphs

We rely on two well-known properties of graph arboricity, whose proofs can be found in [1].

Lemma 8 A graph with $r$ edges has arboricity at most $2\sqrt{r}$.

Lemma 9 For a graph of arboricity $a$, it is possible to orient its edges such that each vertex has out-degree at most $a$. Moreover, there is a CONGEST algorithm with round complexity $O((\log n)$ for finding an orientation such that each vertex has at most $2a$ outgoing edges.

We note that the round complexity of the algorithm orienting the edges is independent of the value of the arboricity.

A converse lemma is also true, in a slightly weaker form:

Lemma 10 If the edges of a graph can be oriented such that each vertex has at most $a$ outgoing edges, then the arboricity of the graph is at most $2a$.

4.3 The algorithm

First, we apply the decomposition of Theorem 13 with parameters $(\delta, \gamma)$, to obtain a tripartition of the edges $(E_m, E_s, E_r)$ with $E_s$ having arboricity $O(n^\delta)$, $|E_r| \leq c'm^{1-\gamma}$, and where each connected component (cluster) in $E_m$ has min-degree at least $c'n^\gamma$ and mixing time $\tilde{O}(n^{12\gamma})$. Following the decomposition, we go through a very similar procedure to that of Sect. 3.

Step 1: learning clique edges from $E_s \cup E_r$

$E_r$ contains at most $c'm^{1-\gamma}$ edges, and therefore has arboricity $\leq 2\sqrt{c'm^{1-\gamma}} \leq 2\sqrt{c'n^{2-2\gamma}} = O(n^{1-\gamma})$. By Lemma 10, we see that $E_r \cup E_s$ has arboricity at most $O(n^{\delta} + n^{1-\gamma})$.

The network finds an orientation of the edges of $E_s \cup E_r$ such that the number of outgoing edges of each node is $O(n^{\delta} + n^{1-\gamma})$. This can be done in $O(\log n)$ rounds using Lemma 9.
To find 5-cliques contained entirely in \( E_t \cup E_r \), we simply have each node \( v \) send all of its outgoing edges in the arboricity orientation to all its neighbors. As the arboricity is \( O(n^\delta + n^{1-\gamma}) \), this can be done in \( O(n^\delta + n^{1-\gamma}) \) rounds. The nodes output the \( K_5 \) copies they detect in this step (copies fully contained in \( E_t \cup E_r \)).

Even if no clique is found, nodes remember the edges that they learned during this part of the algorithm as they move on to the next step.

**Step 2: finding cliques containing an edge from \( E_m \)**

Next, we search for copies of \( K_5 \) that have at least one edge in some cluster. As in the \( K_4 \) algorithm from Sect. 3, given a cluster \( C \), we say that a node \( v \) is \( C \)-light if \( v \notin C \) and \( v \) has at most \( n^\epsilon \) neighbors in \( C \) (that is, \(|N(v) \cap C| \leq n^\epsilon\)). If \( v \) is not \( C \)-light, then we say that \( v \) is \( C \)-heavy.

We do not use the same partition into “light” and “heavy” cliques that was used in Sect. 3. Instead, we use a combination of the strategies employed in Sect. 3, and then show by case analysis that any 5-clique will be detected.

**Step 2(a): finding cliques using light vertices**

This step is similar to the corresponding part in our \( K_4 \) algorithm. We iterate through the clusters sequentially, in lexicographic order of their cluster IDs. For each cluster \( C \), all nodes that belong to \( C \) (i.e., have at least one edge in \( C \)) announce this to their neighbors. Let \( M_C(v) = N(v) \cap C \) denote the neighbors of vertex \( v \) that belong to \( C \).

Next, each node \( v \) that is \( C \)-light sends \( M_C(v) \) to all its neighbors; as \( v \) is \( C \)-light, this requires \( O(n^\epsilon) \) rounds.

In parallel for each \( c_1 \) neighbor of \( v \) that is contained in a cluster for which \( v \) is light, \( v \) sends a list of “edge queries”

\[
Q_v,c_1 = \{c_1, c_2\} \mid \exists c_1, c_2 \text{ for which } v \text{ is light, s.t. } c_1 \in M_{C_1}(v) \land c_2 \in M_{C_2}(v),
\]

There are at most \( O(n^{1-\delta+\epsilon}) \) such edges: as the minimum degree is \( \Omega(n^\epsilon) \), and the clusters are vertex disjoint, there are \( O(n^{1-\delta}) \) clusters, and for any \( C \) for which \( v \) is light, \(|M_C(v)| \leq n^\epsilon \) (as \( v \) is \( C \)-light). Node \( c_1 \) responds with the subset \( (c_1) \times N(c_1) \) \( \cap Q_v,c_1 \) of edges that are actually present, and node \( v \) outputs the 5-cliques it has found using the edges it just learned and the edges it remembers from Step (1).

**Step 2(b): finding cliques using heavy vertices**

This last part of the algorithm is identical to the last part of our \( K_4 \) algorithm. We repeat it here for the reader’s convenience.

Let \( F(C) = \{u, v\} \mid u \in C \cup C \cup u \in C \text{ or } u \in C - \text{ heavy} \) be the set of all edges incident to a node in \( C \) or to some \( C \)-heavy node. Our goal is to find 5-cliques contained entirely in \( F(C) \), in parallel for all clusters. To do this, we have the cluster nodes simulate the execution of the Congested Clique algorithm for \( K_5 \) enumeration from [13] on the \( n \)-vertex graph \((V, F(C))\). This part of the algorithm is carried out in four steps:

I. **Pull**: the nodes of the cluster \( C \) “pull” the edges of \( F(C) \) from nodes outside \( C \), such that every edge of \( F(C) \) is learned by at least one node in \( C \).

II. **Partition**: in this step, we compute a partition \( \{V_c\}_{c \in C} \) of \( V \), which is roughly balanced (i.e., \(|V_c| \approx |V_{c'}|\) for each \( c_1, c_2 \in C \)). Each node \( c \in C \) will be responsible for simulating the nodes in \( V_c \).

III. **Shuffle**: the cluster nodes shuffle the edges of \( F(C) \) between themselves, so that each node \( c \in C \) learns the \( F(C) \)-neighborhood \( N_{F(C)}(v) \) for each node \( v \in V_c \) it needs to simulate. This is carried out by using the routing algorithm from [23].

IV. **Simulate**: the nodes of \( C \) simulate an \( n \)-vertex Congest Clique, and run the \( K_5 \)-enumeration algorithm of [13] on the graph \((V, F(C))\) to list all the 5-cliques in \( F(C) \).

All phases are performed as described in Sect. 3.

### 4.4 Algorithm correctness and analysis

**Round complexity.** By an analysis very similar to Sect. 3, the round complexity of the decomposition is \( \tilde{O}(n^{1-\delta+20\gamma}) \); the round complexity of Step (1) is \( O(n^\delta + n^{1-\gamma}) \) as the arboricity of \( E_t \cup E_r \) is \( O(n^\delta + n^{1-\gamma}) \); the round complexity of Step (2a) is \( O(n^{1-\delta+\epsilon}) \); the round complexity of the “Pull” phase is \( O(n^{1-\epsilon}) \); the “Partition” phase is done locally; The round complexity of the “Shuffle” phase is \( O(n^{2-\delta-\epsilon+12\gamma+o(1)}) \) and the round complexity of the “Simulate” phase is \( O(n^{2-\delta-\epsilon+12\gamma+o(1)}) \) where \( n^{12\gamma+o(1)} \) in both the “Shuffle” and “Simulate” phases comes from the mixing time of the new decomposition used.

**Lemma 11** For \( 0 < \epsilon, \delta < 1 \) and \( 0 < \gamma < 1/2 \), \( K_5 \) enumeration can be performed in

\[
\tilde{O}(n^{1-\delta} + n^{1-\delta+20\gamma} + n^{2-2\delta+12\gamma+3/5} + n^{1-\delta+\epsilon} + (n^{2-\delta-\epsilon} + n^{2-2\delta}) n^{12\gamma+o(1)} + n^\delta + n^{1-\gamma})
\]

rounds in CONGEST.

Choosing \( \gamma = 2/75, \delta = 73/75, \epsilon = 28/75 \), we get overall round complexity of \( O(n^{73/75+o(1)}) \).

**Correctness.**

**Lemma 12** All copies of \( K_5 \) are output by the algorithm with high probability.

**Proof** For a copy \( H \) of \( K_5 \), there are several possible cases:
I $H$ is entirely contained in $E_s \cup E_r$.

II $H$ has one edge in some cluster (i.e., in $E_m$), and the remaining edges are in $E_s \cup E_r$.

III $H$ has a triangle in some cluster, and the other edges are all in $E_s \cup E_r$.

IV $H$ has a triangle in some cluster, one edge in another cluster, and all other edges are in $E_s \cup E_r$.

V $H$ has one edge in one cluster, one edge in another cluster, and all the other edges are in $E_s \cup E_r$.

VI $H$ has a $K_4$ in some cluster, or is fully contained in one.

We note that any $K_3$-copy must fall within at least one of these cases: if all vertices of $H$ are in distinct clusters, it falls under case (1). If $H$ has two vertices in the same cluster, then depending on whether there are two other vertices that share another cluster it falls within case (5) or (2). If $H$ has 3 vertices belonging to the same cluster, then depending on whether the two remaining vertices share another cluster, it falls within case (3) or (4). Finally, if $H$ has 4 or 5 vertices belonging to the same cluster, it falls within case (6).

We show that in each case, $H$ is detected by our algorithm.

Observe that in the first step of the algorithm, all edges in $(E_s \cup E_r) \cap H$ (that is, all edges of the $5$-clique that lie in $E_s$ or in $E_r$) are learned by all vertices of $H$: consider an edge $\{u, v\} \subseteq (E_s \cup E_r) \cap H$, and suppose that in the arboricity orientation of $E_s \cup E_r$, the edge is oriented from $u$ to $v$. In the first step of the algorithm, $u$ sends its outgoing edges to all neighbors, and in particular, to all other nodes of the clique. Thus, all clique nodes learn about the edge $\{u, v\}$.

We immediately see that in Case I, where the clique $H$ is entirely contained in $E_s \cup E_r$, all clique vertices will find the clique.

In Case II, let $\{u, v\} \subseteq E_m \cap H$ be the edge of the clique that is in a cluster. Since $(E_s \cup E_r) \cap H$ is learned by all clique vertices, nodes $u, v$ know about all edges of the clique, and they output it.

In Cases III and IV, there are three vertices $u, v, w$ inside a cluster $C$, and two other vertices $x, y$ outside $C$. If both vertices $x, y$ outside $C$ are light with regards to $C$, then in Step (2a), they query the edges $\{u, v\}, \{v, w\}, \{u, w\}$, and find the clique. If at least one of $x, y$ is heavy, all clique edges will be known to $C$ after the “Pull” procedure, so the clique will be detected by the clique in the “Simulate” procedure of Step (2b).

In Case V, there are two clique edges in different clusters, $\{u_1, v_1\} \in C_1$ and $\{u_2, v_2\} \in C_2$. Recall that $(E_s \cup E_r) \cap H$ is learned by all clique vertices in Step (1), so the vertices $u_1, v_1$ already know all edges of the clique except $\{u_2, v_2\}$. If one of the vertices $u_1, v_1$ is light with respect to $C_2$, then in Step 2(a), the vertex queries in particular the edge $\{u_2, v_2\}$, and finds $H$.

If both $u_1, v_1$ are $C_2$-heavy, then in the “Pull” phase of Step (2b), the edge $\{u_1, v_1\}$ will be sent into the cluster $C_2$. All other clique edges are incident to either $u_2$ or $v_2$ (or both, in case of the edge $\{u_2, v_2\}$), so they will also be used in Step (2b) when we simulate the Congested Clique. Thus, in the “Simulate” phase of Step (2b), the cluster $C_2$ will find $H$.

In Case VI, all edges of $H$ edges are either contained in the cluster or are incident to a node in the cluster, so $H$ is found in the “Simulate” phase of Step (2b).
where again, \( \delta \in (0, 1) \) is the parameter that determines the threshold for what is considered “high degree”. In [17], the value of \( \delta \) is chosen fairly naively: since a \( C_k \)-free graph has at most \( O(n^{1+1/k}) \) edges, for any \( \delta \), we have \( |V_{d}^{\delta}| \leq O(n^{1+1/k-\delta}) \), and balancing the two terms in (1) yields the result.

Our \( C_k \) algorithm retains the framework described above, but uses a tighter bound on \( |V_{d}| \), which we prove in Lemma 13 below. This allows us to choose a smaller value for \( \delta \), lowering the threshold for what we consider a “high-degree node”.

**Bounding the number of high-degree vertices.** We say that a bipartite graph \( G = (A \cup B, E) \) is \((a, b)\)-regular if for all \( v \in A \) it holds that \( d(v) = a \) and for all \( v \in B \) it holds that \( d(v) = b \). Our tighter bound relies on known bounds for the Zarankiewicz number of even-length cycles:

**Definition 8 (Zarankiewicz numbers [40])** The Zarankiewicz number \( z(a, b, H) \) is defined as the maximum number of edges in an \((a, b)\)-regular bipartite graph that does not contain \( H \) as a subgraph.

We rely on the following upper bound:

**Theorem 14 (Zarankiewicz numbers for cycles [32,40])** For any integers \( a, b \geq 0 \) and \( k \geq 2 \),

\[
\exists (2k - 3) \left( \begin{array}{c} \frac{1}{2} + \frac{1}{2}, b + a + b \end{array} \right) \quad \text{for odd } k, \\
(2k - 3) \left( \begin{array}{c} a + b \end{array} \right) \quad \text{for even } k.
\]

Perhaps surprisingly, even though Zarankiewicz numbers only bound the size of bipartite \( H \)-free graphs, it turns out that we can use them to solve even cycle freeness in general graphs, because they allow us to better bound the number of high-degree vertices in a \( C_k \)-free graph.

**Lemma 13** Let \( c = c(k) \) be a large enough constant. For any \( d \geq cn^{1/k} \), let \( V_d \) be \( \{ v \in V \mid d(v) \geq d \} \). If \( G \) is \( C_k \)-free, then

\[
|V_d| \leq \begin{cases} \notag \mathcal{O}_k(\log(n^{1/k}, (n^{1+1/d})^{2/k})), & \text{for odd } k \geq 3, \\ \mathcal{O}_k(\log(n^{1/k}, (n^{1/2}/d)^{2/k})), & \text{for even } k \geq 4. \end{cases}
\]

Here, the notation \( \mathcal{O}_k(\cdot) \) hides constants that depend on \( k \). For an integer \( N \) and a graph \( H \), denote by \( \text{ex}(N, H) \) maximum number of edges an \( H \)-free graph with \( N \) vertices can contain.

**Proof of Lemma 13** Define the following sets of edges:

- \( E_d = E(V_d, V) \): edges incident to some node in \( V_d \).
- \( E_{int} \) : the internal edges \( E_{int} = E(V_d, V_d) \) of \( V_d \).
- \( E_{ext} = E_d \setminus E_{int} \): the external edges. Equivalently, \( E_{ext} = E(V_d, V \setminus V_d) \).

Observe that \( |E_d| \geq d \cdot |V_d|/2 \), since every node in \( V_d \) has degree at least \( d \). On the other hand, since \( G \) is \( C_k \)-free, so is \( G[V_d] \), and therefore \( |E_{int}| \leq \text{ex}(|V_d|, C_k) \leq c'(k) \cdot |V_d|^{1+1/k} \), as ex(\( N, C_{2k} \) ≤ c'(k)\( N^{1+1/k} \), for a constant \( c'(k) \) large enough in \( k \) (e.g. [4]). But this implies that for a constant \( c \geq 4c' \), we have

\[
|E_{ext}| = |E_d| - |E_{int}| \geq \frac{1}{2}d|V_d| - c'|V_d|^{1+1/k}
\]

\[
\geq \frac{1}{2}d|V_d| - \frac{1}{4}d|V_d| = \frac{1}{2}d|V_d| - \frac{d}{4}d|V_d|
\]

\[
\geq \frac{1}{4}d|V_d|.
\]

The bipartite graph induced on \( V_d \times (V \setminus V_d) \) is \( C_k \)-free (because \( G \) is \( C_k \)-free), and therefore

\[
|E_{ext}| \leq z(|V_d|, |V \setminus V_d|, C_k) \leq z(|V_d|, n, C_k).
\]

(We rely on the fact that Zarankiewicz numbers are non-decreasing, because for any \( a \) and \( b' > b \), an \( H \)-free, \((a, b')\)-regular bipartite graph with \( e \) edges is trivially extended to a \( H \)-free, \((a, b')\)-regular bipartite graph with \( e \) edges by simply adding nodes on the right-hand side that have no edges.)

For even \( k \geq 4 \), by Theorem 14,

\[
\frac{1}{4}d|V_d| \leq (2k - 3) \left( \left| V_d \right|^\frac{k+2}{2k} n^{\frac{1}{2}} + |V_d| + n \right)
\]

\[
\leq (2k - 3) \left( \left| V_d \right|^\frac{k+2}{2k} n^{\frac{1}{2}} + 2n \right).
\]

If \( |V_d| \leq (4n)^{k/(k+2)} \) then we are done, as the statement of the lemma is satisfied, so assume not. In that case,

\[
\left| V_d \right|^\frac{k+2}{2k} n^{\frac{1}{2}} \geq ((4n)^{k/(k+2)})^{\frac{k+2}{2k}} n^{\frac{1}{2}} = (4n)^{1/2} \cdot n^{1/2} = 2n,
\]

so for the right-hand side of (2) we obtain:

\[
(2k - 3) \left( \left| V_d \right|^\frac{k+2}{2k} n^{\frac{1}{2}} + 2n \right) \leq 2(2k - 3) \left( \left| V_d \right|^\frac{k+2}{2k} n^{\frac{1}{2}} \right).
\]

Together with (2),

\[
\frac{1}{4}d|V_d| \leq 2(2k - 3) \left( \left| V_d \right|^\frac{k+2}{2k} n^{\frac{1}{2}} \right),
\]

and by rearrangement

\[
|V_d|^{1-\frac{k+2}{2k}} = d|V_d|^\frac{k+2}{2k} \leq 8(2k - 3)n^{1/2}
\]

meaning that

\[
|V_d| \leq (8(2k - 3)(n^{1/2}/d))^{\frac{2k}{k+2}}.
\]
For odd $k \geq 3$, by Theorem 14,

\[
\frac{1}{4}d|V_d| \leq (2k-3) \left( (|V_d|n)^{\frac{1}{2}+\frac{1}{2k}} + |V_d| + n \right)
\leq (2k-3) \left( (|V_d|n)^{\frac{1}{2}+\frac{1}{2k}} + 2n \right).
\]

Again, if $|V_d| \leq \frac{n^{\frac{1}{k+1}}}{d}$ then we are done, so assume $|V_d| > \frac{n^{\frac{1}{k+1}}}{d}$. In this case,

\[
(|V_d|n)^{\frac{1}{2}+\frac{1}{2k}} \geq \left( n^{\frac{1}{2}+\frac{1}{k+1}} \right)^{\frac{1}{2}+\frac{1}{2k}} = \left( n^{\frac{2k+1}{2k+2}} \right)^{\frac{k+1}{2k}} = n.
\]

For the right-hand side of (4), we get

\[
(2k-3) \left( (|V_d|n)^{\frac{1}{2}+\frac{1}{2k}} + 2n \right) \leq 3(2k-3) \left( (|V_d|n)^{\frac{1}{2}+\frac{1}{2k}} \right),
\]

and by (4),

\[
d|V_d|^{\frac{k+1}{2k}} = d|V_d|^{\frac{k-1}{2k}} \leq 12(2k-3)n^{\frac{k+1}{2k}}
\]

Meaning

\[
|V_d| \leq 12(2k-3)n^{\frac{k+1}{2k}} / d^{\frac{2k}{k+1}}.
\]

Next, combined with the framework of [17], we show how to obtain an $\tilde{O}_k(n^{1-2/(k^2-k+2)})$ round algorithm for odd $k \geq 3$, and an $\tilde{O}_k(n^{1-2/(k^2-2k+4)})$ round algorithm for even $k \geq 4$ (Theorem 2).

Bounding the running time of our improved algorithm is somewhat hairy (due to the form of the bounds in Theorem 14). Before we prove the theorem, we state the following two technical lemmas, which are proven at the end of the section:

**Lemma 14** The following inequalities hold:

1. For any $k \geq 4$,

\[
\min_{\delta \geq 0} \max \left( (k-2)\delta + \frac{1}{k} \left( \frac{1}{2} - \delta \right), \left( \frac{2k}{k-2} \right) \right)
= 1 - \frac{2}{k^2 - 2k + 4},
\]

and furthermore,

\[
1 - \frac{2}{k^2 - 2k + 4} > \frac{k-1}{k}.
\]

2. For any $k \geq 3$,

\[
\min_{\delta \geq 0} \max \left( (k-2)\delta + \frac{1}{k} \left( \frac{1}{2} - \delta \right), \left( \frac{2k}{k-2} \right) \right)
= 1 - \frac{2}{k^2 - k + 2},
\]

and furthermore,

\[
1 - \frac{2}{k^2 - k + 2} > \frac{k-1}{k}.
\]

**Lemma 15** Let $f : [0, 1] \to \mathbb{R}$ and $g : [0, 1] \to \mathbb{R}$ be increasing and decreasing continuous functions respectively. Then for any $x \in [0, 1]$,

\[
\min_{\delta \in [x, 1]} \max(f(\delta), g(\delta)) = \min_{\delta \in [0, 1]} \max(f(\delta), f(x), g(\delta)).
\]

We obtain the following improved $C_{2k}$-freeness algorithm.

**Proof of Theorem 2** As we explained above, it is shown implicitly in [17] that the round complexity of $C_{2k}$-freeness is $\tilde{O}(\min_{\delta \geq 0} \max(n^{(k-2)\delta + \frac{1}{k}}, |V_{out}|))$. We now instantiate this algorithm with a different choice of $\delta$ than the one used in [17].

For even $k \geq 4$, plugging in the bound on $|V_{out}|$ from Lemma 13, we see that the running time is

\[
\tilde{O} \left( \min_{\delta, n^\delta \geq cn^{1/k}} \max \left( n^{(k-2)\delta + \frac{1}{k}}, n^{\frac{k+1}{k+2}}, (n^{1/2} / n^{\delta})^{\frac{2k}{k+2}} \right) \right),
\]

The first term increases with $\delta$, taking its minimum value when $n^{\delta} = cn^{1/k}$; the second term does not depend on $\delta$, and the third decreases as $\delta$ increases. Therefore, by Lemma 15,

\[
\min_{\delta, n^\delta \geq cn^{1/k}} \max \left( n^{(k-2)\delta + \frac{1}{k}}, n^{\frac{k+1}{k+2}}, (n^{1/2} / n^{\delta})^{\frac{2k}{k+2}} \right)
= \min_{\delta \geq 0} \max \left( n^{(k-2)\delta + \frac{1}{k}}, n^{\frac{k+1}{k+2}}, (n^{1/2} / n^{\delta})^{\frac{2k}{k+2}} \right).
\]

Furthermore, $(k-2)\frac{1}{k} + \frac{k-1}{k+2} = \frac{k-1}{k+2} > \frac{k}{k+2}$ for $k > 2$, therefore the round complexity is

\[
\tilde{O} \left( \min_{\delta \geq 0} \max \left( n^{(k-2)\delta + \frac{1}{k}}, n^{\frac{k+1}{k+2}}, (n^{1/2} / n^{\delta})^{\frac{2k}{k+2}} \right) \right).
\]

By Lemma 14, this evaluates to

\[
\tilde{O}(n^{1-2/(k^2-k+2)}).
\]

Similarly, for odd $k \geq 3$, the round complexity we obtain is

\[
\tilde{O} \left( \min_{\delta, n^\delta \geq cn^{1/k}} \max \left( n^{(k-2)\delta + \frac{1}{k}}, n^{\frac{k+1}{k+2}}, (n^{1/2} / n^{\delta})^{\frac{2k}{k+2}} \right) \right).
\]
As the first term is increasing in $\delta$ and the latter two are non-increasing in $\delta$, by Lemma 15,
\[
\min_{\delta \geq 0} \max \left( (k - 2)\delta + \frac{1}{k}, \frac{1}{2} - \delta \right) \cdot \left( \frac{2k}{k - 2} \right)
\]
is minimized when $(k - 2)\delta + \frac{1}{k} = \left( \frac{1}{2} - \delta \right) \cdot \left( \frac{2k}{k - 2} \right)$. Simplifying, we obtain
\[
\delta = \frac{(k^2 - k + 2)}{(k^2 - 2k + 4)}.
\]

Note that this value is non-negative for $k > 2$. Plugging this $\delta$ into the expression $(k - 2)\delta + \frac{1}{k}$, we get:
\[
(k - 2)\delta + \frac{1}{k} = 1 - \frac{2}{k^2 - k + 4}.
\]

And indeed, for $k > 2$ we have $1 - \frac{2}{k^2 - k + 4} > \frac{k - 1}{k}$.

As for the second claim in the lemma, we again see that the expression
\[
\min_{\delta \geq 0} \max \left( (k - 2)\delta + \frac{1}{k}, \frac{k + 1}{2k} - \delta \right) \cdot \left( \frac{2k}{k - 1} \right)
\]
is minimized when $(k - 2)\delta + \frac{1}{k} = \frac{k + 1}{2k} - \delta \cdot \left( \frac{2k}{k - 1} \right)$, assuming the value of this $\delta$ is non-negative. Simplifying yields
\[
\delta = \frac{(k^2 + 1)}{(k(k^2 - k + 2))},
\]
which is non-negative when $k > 2$. Plugging this $\delta$ into the expression $(k - 2)\delta + \frac{1}{k}$, we get:
\[
(k - 2)\delta + \frac{1}{k} = 1 - \frac{2}{k^2 - k + 2}.
\]

For $k > 2$, we have $1 - \frac{2}{k^2 - k + 2} > \frac{k - 1}{k}$, as desired. \qed

**Proof** (Lemma 15) First, observe that
\[
\min \max_{\delta \in [0,1]} (f(\delta), g(\delta)) \geq \min \max_{\delta \in [0,1]} (f(\delta), f(x), g(\delta))
\]
because $f(\delta) \geq f(x)$ whenever $\delta \geq x$, then
\[
\min \max_{\delta \in [0,1]} (f(\delta), f(x), g(\delta)) = \min \max_{\delta \in [0,1]} (f(\delta), g(\delta)) \geq \min \max_{\delta \in [0,1]} (f(\delta), g(\delta)).
\]

For the other direction, suppose for the sake of contradiction that
\[
\min \max_{\delta \in [0,1]} (f(\delta), g(\delta)) > \min \max_{\delta \in [0,1]} (f(\delta), f(x), g(\delta))
\]
and let $\gamma \in [0, x)$ be some value that minimizes the right-hand side, such that
\[
\min \max_{\delta \in [0,1]} (f(\delta), f(x), g(\gamma)) = \max(f(\gamma), f(x), g(\gamma)) < \min \max_{\delta \in [0,1]} (f(\delta), g(\delta)).
\]

Since $\gamma < x$ and $f$ is increasing, we have $f(\gamma) < f(x)$, so
\[
\max(f(\gamma), f(x), g(\gamma)) = \max(f(x), g(\gamma)).
\]
But since $g$ is decreasing and $\gamma < x$, we have $g(\gamma) > g(x)$, and therefore
\[
\max(f(x), g(\gamma)) \geq \max(f(x), g(x)) \geq \min \max_{\delta \in [0,1]} (f(\delta), g(\delta)),
\]
contradicting (10). \qed

### 6 Reducing the network diameter in subgraph-detection problems

In this section show a reduction from general networks to networks with small diameter for $H$-freeness and $H$-enumeration. This result uses a similar approach to the result of [16], which gives a similar result for property-testing of subgraph-freeness.

We show two different reductions: a randomized reduction that reduces the diameter to $k \log n$ for $|H| = k$, and a deterministic reduction reducing the diameter to $k \cdot 2O(\sqrt{\log n \log \log n})$. 

© Springer
Theorem 15  Consider a problem $\mathcal{P}$ which is either $H$-freeness or $H$-enumeration, where $|H| = k$. Let $\mathcal{A}$ be a protocol that solves $\mathcal{P}$ in time $T(n, D)$ with error probability $\rho = o\left(\frac{1}{\log n}\right)$. There is an algorithm $\mathcal{A}'$ that solves $\mathcal{P}$ with round complexity $\tilde{O}(T(n, O(k \log n)) + k \log^2 n)$ and error probability at most $c pn \log n + \frac{1}{\poly(n)}$, for some constant $c$.

Theorem 16  Consider a problem $\mathcal{P}$ which is either $H$-freeness or $H$-enumeration, where $|H| = k$. Let $\mathcal{A}$ be a deterministic protocol that solves $\mathcal{P}$ in time $T(n, D)$. There is a deterministic algorithm $\mathcal{A}'$ that solves $\mathcal{P}$ with round complexity $O(f(n) \cdot T(n, k \cdot f(n)))$, where $f(n) = 2^{O(\sqrt{\log n \log \log n})}$.

A $(d, c)$-network decomposition is a decomposition in which each node is given a color in $[c]$, such that for all $c_i \in [c]$ the diameter of each connected component in the subgraph induced by vertices with color $c_i$ is at most $d$. A $k$-hop $(d, c)$-decomposition is a $(d, c)$-decomposition where every two nodes with color $c_i$ either have hop-distance at least $\geq k$ in $G$, or they are in the same connected component in the graph induced by vertices with color $c_i$.

The randomized algorithm to compute the decomposition is a straightforward adaptation of the strong diameter (1-hop) network decomposition of Elkin and Neiman [15], which itself is a variation of the parallel graph decomposition algorithm of Miller, Peng, and Xu [31]. As we have to change some details in the proof of [15], we give the proof later in this section for completeness. Note that in this variant of the decomposition a node may have more than one color.

Theorem 17  Let $G = (V, E)$ be an $n$-node graph and let $k \geq 2$ be an integer. There is a randomized CONGEST-model algorithm that w.h.p. constructs a set of clusters of diameter $O(k \log n)$ such that each node is in at least one cluster, the clusters are colored with $O(\log n)$ colors, and clusters of the same color are at distance at least $k$ from each other in $G$.

The deterministic decomposition is taken to be the $k$-hop $(d, c)$-network decomposition of [22].

Theorem 18  ([22] Theorem 3) Let $G = (V, E)$ be an $n$-node graph and let $k \geq 1$ be an integer. There is a deterministic CONGEST-model algorithm that computes a $k$-hop $(k \cdot f(n), f(n))$-decomposition in $k \cdot f(n)$ rounds, where $f(n) = 2^{O(\sqrt{\log n \log \log n})}$.

Our reduction works as follows: Consider a problem $\mathcal{P}$ that is either $H$-freeness or $H$-enumeration, for an $H$, $|H| = k$. Let $\mathcal{A}$ a protocol that solves $\mathcal{P}$ in time $T(n, D)$. The network calculates a $(2k + 1)$-hop decomposition (either using Theorem 17 or Theorem 18).

Denote by $V_i$ the set of vertices with color $i$, and for a vertex set $A$, let $G_k(A)$ be the graph induced by $A$ and its $k$-neighborhood.

Next, sequentially for each color $c_i$, the network runs algorithm $\mathcal{A}$ on each cluster of $G_k(V_i)$. If $\mathcal{P}$ is an $H$-freeness problem, a node accepts if it accepts in $\mathcal{A}$ for all colors $c_i$, and rejects otherwise. If $\mathcal{P}$ is the $H$-enumeration problem, a node outputs the set of all $H$ copies detected by it in any $G_k(V_i)$.

Remark 3  We note that there is a known poly-logarithmic deterministic network decomposition [36], and assuming it can be extended to assure distance $\geq k$ between two clusters of the same color (which we believe should be possible), Theorem 16 can be improved to have only a poly-logarithmic overhead in round complexity by using this decomposition instead of the decomposition of [22].

6.1 Correctness and analysis

If the decomposition used was Theorem 17, the diameter in each $V_i$ is $O(k \log n) + k = O(k \log n)$, so the round complexity is at most $O(k \log n \cdot T(n, O(k \log n)) + k \log^2 n)$.

If the decomposition used was Theorem 18, the diameter in each $V_i$ is $k \cdot 2^{O(\sqrt{\log n \log \log n})} + k = k \cdot 2^{O(\sqrt{\log n \log \log n})}$, so the round complexity is at most $O(f(n) \cdot T(n, k \cdot f(n)))$.

Next, we show correctness:

Lemma 16  $H_0$ is a copy of $H$ in $G$ if and only if there exist $i \in [c]$ such that $H_0$ is in $G_k(V_i)$.

Proof. Let $v$ be an arbitrary vertex in $H_0$, and let $c(v)$ be its color. As $H_0$ has diameter $\leq k$, it is fully contained in $G_k(V_{c(v)})$. On the other hand, if there exist an $i$ such that $H_0$ is a copy of $H$ in $G_k(V_i)$, then clearly $H_0$ is a copy in $G$ as well.

By Lemma 16 indeed $G$ is $H$-free if and only if all $G_k(V_i)$ are $H$-free, and the set of copies of $H$ in $G$ is equal to the union of the set of copies of $H$ in $G_k(V_i)$. In the randomized decomposition, the error probability of $O(pn \log n + \frac{1}{\poly(n)})$ in Theorem 15 is obtained by taking union bound on the the event that any instance of $\mathcal{A}$ in a cluster in one of the $O(\log n)$ graphs fails, and the event that the decomposition itself fails.

6.2 Randomized decomposition

Proof of Theorem 17  The algorithm consists of $O(\log n)$ phases, where in each phase, the clusters of one color are constructed. For a single phase, we will show:

(I) W.h.p., all clusters have diameter $O(k \log n)$.

(II) For any two nodes $u, v \in V$ that end up in different clusters of the phase, we have $d_G(u, v) \geq k$.

(III) W.h.p., each phase can be implemented in $O(k \log n)$ rounds in the CONGEST model.

(IV) For each node $v \in V$, the probability to be in a cluster of the phase is at least $1/e$. 

 Springer
In order to make sure that clusters of the same color are at distance more than \( k \) on \( G \), each phase is run (with new independent randomness) on the original graph \( G \). That is, unlike in the algorithm of [15], which guarantees disjoint clusters, we do not remove nodes from the graph as soon as they are in some cluster. Together (I)–(IV) and the fact that different phases are independent directly imply the claim of the theorem.

The algorithm for a single phase is defined as follows. First, each node \( v \in V \), draws an independent random variable \( X_v \) according to the exponential distribution with rate \( 1/k \), i.e., \( X_v \sim \text{Exp}(1/k) \). For each node \( v \in V \), we define a radius \( R_v := \lceil X_v \rceil \).\(^3\) Note that w.h.p., each of the values \( R_v \) is upper bounded by \( R_{\max} = O(k \log n) \). The construction of the clusters is defined as follows. For any two nodes \( u, v \in V \), we define \( m_{u,v} := R_v - d_G(u, v) \). Each node \( u \in V \) then orders the values \( m_{u,v} \) for all \( v \in V \) (including \( v = u \)). Ties broken by giving preference to nodes \( v \) with larger ID. Let \( m_{u,v_1} \) and \( m_{u,v_2} \) (\( m_{u,v_1} \geq m_{u,v_2} \)) be the largest two such values. Node \( u \) then selects \( v_1 \) as its center and it joins the cluster of \( v_1 \) if and only if \( m_{u,v_1} - m_{u,v_2} > k - 1 \). Note that if \( u \) selects \( v_1 \) as its center, then clearly, all nodes on the shortest path from \( u \) to \( v_1 \) also select \( v_1 \) as their center and if \( u \) joins the cluster of \( v_1 \), all nodes on the shortest path from \( u \) to \( v_1 \) also join the cluster of \( v_1 \). Because, w.h.p., \( R_v = O(k \log n) \), we have therefore proved claim (I) (all clusters have diameter at most \( O(k \log n) \)). To prove (II), consider two nodes \( u \) and \( u' \) that join the clusters of \( v \) and \( v' \) for \( v \neq v' \). For the sake of contradiction, assume that \( d_G(u, u') \leq k - 1 \). Without loss of generality, assume that \( m_{u',v} \leq m_{u,v} \). We then have \( m_{u',v} \geq m_{u,v} - d_G(u, u') \geq m_{u,v} - (k - 1) \geq m_{u,v'} - (k - 1) \) and we thus get that \( m_{u,v'} - m_{u',v} \leq k - 1 \), a contradiction to the assumption that \( u' \) joins the cluster of \( v' \).

We next show (III), i.e., that the algorithm can be implemented in \( O(k \log n) \) rounds in the CONGEST model. First note that a node \( u \) can only join a cluster of a node \( v_1 \) if \( m_{u,v_1} - m_{u,v_2} > k - 1 \) and it thus suffices for \( u \) to collect \( m_{u,v} \) values that are positive. Each node \( v \) forwards its ID and its radius \( R_v \) to all nodes within distance \( R_v \). Assume that \( R_v \leq R_{\max} = O(k \log n) \) for all \( v \in V \). In order for each node to receive the largest two \( m_{u,v} \) values without congestion, each node \( v \) starts flooding its ID at time \( R_{\max} - R_v \). Node \( u \) then gets \( v \)’s ID exactly at time \( R_{\max} - R_v + d_G(u, v) = R_{\max} - m_{u,v} \). Because the largest two values \( m_{u,v_1} \) and \( m_{u,v_2} \) are also among the largest two values on their respective shortest paths from \( v_1 \) to \( u \) and from \( v_2 \) to \( u \), the largest two values \( m_{u,v_1} \) and \( m_{u,v_2} \) reach node \( u \) without congestion and thus at time at most \( R_{\max} = O(k \log n) \).

It remains to show that each node joins a cluster with probability at least \( 1/e \). To prove this, we use a technical lemma from [31], which (slightly restated) shows the following:

**Lemma 17** [31] Let \( d_1 \leq \cdots \leq d_n \) be arbitrary values and \( X_1, \ldots, X_n \) independent random variable from \( \text{Exp}(\beta) \). Then the probability that there is a gap of at least \( c \) between the largest and the second largest values of \( X_i - d_i \) is at most \( 1 - e^{-\beta c} \).

For any two nodes, let \( m'_{u,v} := X_v - d_G(u, v) \) (i.e., we replace \( R_v \) by the original exponential random variable \( X_v \)). Note that if the difference between the largest two \( m'_{u,v} \) values for \( u \) is more than \( k \), then the difference between the largest two \( m_{u,v} \) values is more than \( k - 1 \) and \( u \) joins a cluster. Claim (IV) now follows directly from Lemma 17 by choosing \( c = k \) and \( \beta = 1/k \).

**7 Lower bound on listing all copies of \( C_4 \)**

We show that listing all copies of \( C_4 \) in the graph requires \( \Theta(n) \) rounds.

The upper bound is easy: Each node sends its neighbors all of its edges, and outputs all 4-cycles known to it. As the diameter of \( C_4 \) is 2, and at the end of the process each node knows its 2-neighborhood, it knows all 4-cycles it participates in.

For our lower bound, our reduction graph is the same as in [14], but we reduce from a different problem: let \( \text{SETINTERSECTION}_m \) be the 2-party problem where Alice and Bob receive sets \( X, Y \subseteq [m] \), and must output the intersection \( X \cap Y \); each player outputs some elements, such that the union of the elements output by the players is exactly \( X \cap Y \). The \( \text{SETINTERSECTION}_m \) problem is a generalization of the \( \text{DISJOINTNESS}_m \) problem, in which the players need to determine whether \( X \cap Y = \emptyset \), and which is known to require \( \Omega(m) \) bits for randomized protocols that succeed with high constant probability [35].

**Lemma 18** If there is a protocol that lists all copies of \( C_4 \) in \( r \) rounds in graphs on \( 4n \) vertices, then there is a 2-party protocol for \( \text{SETINTERSECTION}_{m^2} \) using \( O(r \cdot n \log n) \) bits.

**Proof** Given a protocol for enumerating \( C_4 \), we can solve \( \text{SETINTERSECTION}_{m^2} \) as follows: identify the universe of the \( \text{SETINTERSECTION} \) problem with \([n] \times [n]\). Alice and Bob construct a graph \( G_{X,Y} \) on nodes \([n] \times [4]\). For each \( i \in [n] \), there is an edge between nodes \((i, 1), (i, 2)\) and between nodes \((i, 3), (i, 4)\). If \((i, j) \in E\), then we add an edge between nodes \((j, 1), (i, 4)\), and if \((i, j) \in X\) we add the edge \((i, 2), (j, 3)\).

By construction, the graph \( G_{X,Y} \) contains only 4-cycles of the form \((i, 1), (i, 2), (j, 3), (j, 4)\). Moreover, the edge

\(^3\) Note that the random variables \( R_v \) are actually geometrically distributed. We use the construction using exponential random variables to be consistent with the work in [15, 31].
(i, 2), (j, 3) is present iff (i, j) ∈ X, and the edge (j, 4), (i, 1) is present iff (i, j) ∈ Y. Therefore, each copy of C₄ in Gₓ,y corresponds to an intersection element (i, j) ∈ X ∩ Y.

We can simulate the execution of the distributed algorithm on Gₓ,y just as in [14]: For t = 1, . . . , r communication rounds, Alice maintains the state in round t of the vertices of the form (i, 1) and (i, 3) for any i ∈ [n], and Bob maintains the state of the vertices of the form (i, 2) and (i, 4) for any i ∈ [n]. For t = 1, . . . , r rounds, each player sends the other player the bits sent in the t’th round on the edges of the form (i, 1), (i, 4) or the form (i, 2), (i, 3) for any i ∈ [n] (which form a cut between the vertices maintained by Alice, and the vertices maintained by Bob), and locally updates the state of the vertices it simulates in their state in end of the t’th round.

As there are O(n) such edges, and the number of bits sent on each edge is O(log n), the communication cost of this protocol is O(n log n · r). When we are done, Alice outputs the elements (i, j) corresponding to the 4-cycles output by nodes on her side of the graph, and Bob does the same for his side of the graph.

Corollary 2Listing all copies of C₄ requires Ω(n/ log n) rounds in CONGEST.

8 Two-party protocol for detecting 6-cycles

We show that a certain type of reduction from two-party communication complexity cannot prove a lower bound stronger than Ω(√n) for C₆-freeness. This type of reduction is the one used in [14, 29] to prove a lower bound of Ω(√n) for 4-cycles (in [14]) and for C₂k for any k (in [29]), as well as many other distributed lower bounds.

The reduction has the following structure (we call them “partition-based”): suppose we want to prove that a problem P is hard to solve in a class G of graphs, over a fixed set V of nodes, by reduction from some 2-party communication complexity problem f : {0, 1}ᴺ × {0, 1}ᴺ → {0, 1}. We fix a partition V_A, V_B of the vertices V, and assume that given their inputs X, Y ∈ {0, 1}ᴺ (respectively), the two players jointly construct a graph Gₓ,y ∈ G, such that Alice knows the edges E_A incident to the nodes of V_A (even without knowing Bob’s input Y), and Bob knows the edges E_B incident to V_B (even without knowing Alice’s input X). Let S_A, S_B be the set of nodes in V_A (resp. V_B) that have at least one edge to a node in V_B (resp. V_A). We prove that Alice and Bob can simulate the execution of a distributed algorithm in the graph Gₓ,y without using a lot of communication per round, and furthermore, that from a solution to the distributed problem P, they can compute a solution to the function f that they want to solve. Then, if a communication lower bound on f is known, it implies a round lower bound for solving P.

We show, using Theorem 14, that for any partition V_A, V_B of a vertex set V, if it is assumed that Alice knows all edges incident to V_A and Bob knows the edges incident to V_B, then Alice and Bob can check whether the graph is C₆ free, using O((|S_A| + |S_B|) · √n log n) bits. This protocol rules out improving the existing lower bound of Ω(√n) for C₆-freeness of [29] by more than a logarithmic factor using partition-based reductions from 2-party communication complexity. (This result is inspired by [11], which showed a similar results for cliques. Our protocol, however, is completely different.)

Alice and Bob begin the protocol by checking if their own input contains a copy of C₆, and if so they reject and terminate. Otherwise, the players search for a 6-cycle that crosses the cut between them, and contains at least one edge from E_A \ E_B and at least one edge from E_B \ E_A.

For an edge set F and i ≥ 1, let Fᵢ ⊆ V² be the set of all pairs of nodes connected by a path of i edges from F.

Let A₁ = E_A ∩ (S_A × S_A) be the edges Alice sees between her cut nodes; Alice sends these edges to Bob. In addition, Alice wants Bob to learn the set

A₂ = {(u, v) | u, v ∈ S_A, ∃w /∈ S_A : {u, w} ∈ E_A ∩ {w, v} ∈ E_A}.

She does this in one of two ways:

- If the edge set A₃ = {(u, w) ∈ E_A | u ∈ S_A, w /∈ S_A} is of size at most |S_A| · √n, Alice sends A₃ to Bob, and he computes A₂ from it.
- Otherwise, Alice computes A₂ herself and sends it to Bob.

Define B₁, B₂, B₃ similarly; Bob acts symmetrically to Alice, so that she learns B₁ and B₂. Finally, both players check if they “see” a 6-cycle: Alice rejects if either E_A³ \ B₁ ̸= ∅ or E_A⁴ ∩ (B₂³ ⊔ B₂) ̸= ∅, and similarly for Bob.

The correctness of the protocol is proven by a case analysis, going over all possible ways a 6-cycle could be split between the two players.

Lemma 19 If G contains a 6-cycle

C = {(u₀, . . . , u₅), (ui, ui₊₁ mod 6)}ₑ∈[6],

then some player rejects.

Proof C has 6 nodes, each in either V_A or V_B; therefore, either |V_A ∩ V(C)| ≥ 3, or |V_B ∩ V(C)| ≥ 3. Assume w.l.o.g. that |V_A ∩ V(C)| ≥ 3. Therefore, up to symmetry, one of the following three conditions must occur: either u₀, u₂, u₄ ∈ V_A, or u₀, u₁, u₃ ∈ V_A, or u₀, u₁, u₂ ∈ V_A.
Lemma 20

\[ u_0, u_2, u_4 \in V_A; \text{since all edges of } C \text{ are incident to a vertex of Alice, } E(C) \subseteq E_A, \text{ and Alice immediately sees } C. \]

\[ u_0, u_1, u_3 \in V_A; \text{we split into two cases. If at least one of } u_2, u_4, u_5 \text{ is in } E_A \text{ then } E(C) \subseteq E_A, \text{ and Alice immediately sees } C. \text{ Otherwise, the only edge in } C \text{ Alice does not have in her input is } (u_4, u_5). \text{ But since } u_4, u_5 \in S_B, \text{ Bob sends } (u_4, u_5) \text{ in } B_1. \]

\[ u_0, u_1, u_2 \in V_A; \text{we divide into sub-cases.} \]

- If \( u_3, u_4, u_5 \in V_A \), then \( E(C) \subseteq E_A \), and Alice immediately sees \( C \).

- If \( u_3, u_4 \in V_A \) but \( u_5 \in V_B \), it is still true that \( E(C) \subseteq E_A \), because \( E_A \) includes all edges incident to \( u_0, \ldots, u_4 \), and this includes all edges of \( C \).

- If \( u_3 \in V_A \) but \( u_4, u_5 \in V_B \), then we have \( u_4, u_5 \in S_B \) (both nodes have edges to nodes in \( V_A \)) and \( (u_4, u_5) \in E_B \), and therefore \( (u_4, u_5) \in B_1 \). After Bob sends \( B_1 \) to Alice, she will find \( C \), as \( (u_4, u_5) \in E_A^2 \cap B_1 \).

- Finally, if \( u_3, u_4, u_5 \in V_B \), then \( u_3, u_5 \in S_B \), then \( (u_3, u_5) \in B_1 \) again, these nodes have edges to \( V_A \). If \( u_4 \in S_B \), then \( (u_3, u_5) \in B_2 \). Again, Alice will find \( C \), as \( (u_3, u_5) \in E_A^3 \cap B_2 \).

\[ \square \]

More interestingly, let us show how Theorem 14 is used to prove that the players send each other a total of \( O(|S_A| + |S_B|) \cdot \sqrt{n} \log n \) bits.

Lemma 20 If \( E_A \) and \( E_B \) are \( C_0 \)-free, then the players send each other sets of size at most \( O(|S_A| + |S_B|) \cdot \sqrt{n} \).

Proof Consider first \( A_1 \) (and similarly, \( B_1 \)). Recall that \( A_1 = E_A \cap (S_A \times S_A) \subseteq E_A \), and since \( E_A \) is \( C_0 \)-free, so is \( A_1 \). Since the Túran number of 6-cycles is \( ex(n, C_6) = O(n^{1+1/3}) \) [4], we have

\[ |A_1| \leq |S_A|^{4/3} = |S_A| \cdot |S_A|^{1/3} < (|S_A| + |S_B|) \cdot \sqrt{n}. \]

Next let us bound the size of the second set Alice sends. If \( |A_2| \leq |S_A| \cdot \sqrt{n} \), then Alice sends \( A_3 \); so, assume that \( |A_2| > |S_A| \cdot \sqrt{n} \). In this case Alice sends \( A_2 \), the set of pairs in \( S_A \) connected by a path of length 2 through some node not in \( S_A \). Note that \( |A_2| \leq |S_A|^2 \) (as \( A_2 \) is a set of pairs from \( S_A \)), so if we can show that \( |S_A| = O(\sqrt{n}) \), we are done.

Consider the bipartite graph \((S_A, V_A \setminus S_A, \ E_A \cap (V_A \times V_A));\) the set \( A_3 \) is the edge set of this graph. Using the bound on Zarankiewich Numbers in Theorem 14, because it is a \( C_0 \)-free graph, it has at most \( O((|S_A| \cdot |V_A \setminus S_A|)^{2/3}) = O((|S_A| \cdot n)^{2/3}) \) edges. However, since we assumed that \( |A_3| > |S_A| \cdot \sqrt{n} \), we have

\[ c(|S_A| \cdot n)^{2/3} > |S_A| \cdot \sqrt{n}, \]

where \( c \) is the constant from Theorem 14. But this implies that \( |S_A| < c^3 \sqrt{n} \), completing the proof.

\[ \square \]

9 Subquadratic algorithm for subgraph-freeness problems

In [17], it was shown that for any constant \( k \geq 1 \), there exist a subgraph \( H_k \) of size \( |V(H_k)| = O(k) \) such that any randomized algorithm solving \( H_k \)-freeness in \( \text{CONGEST} \) has round complexity of \( \tilde{O}(n^{2 - \frac{1}{k}}) \). As \( k \) grows, the bound becomes arbitrarily close to quadratic round complexity. This leads to the natural question of whether there exists a single subgraph of constant size \( H \) for which the round complexity of \( H \)-freeness is truly quadratic, \( \tilde{O}(n^2) \).

In this section, we answer negatively, by showing an almost matching upper bound:

Theorem 19 For any constant integer \( k \) and any subgraph \( H \) of size \( k \), there exists a randomized algorithm for \( H \)-freeness in \( \text{CONGEST} \) with round complexity of \( \tilde{O}(n^{2 - \frac{2}{\pi^2}}) \), for both induced and non-induced variants.

9.1 Preliminaries

Before presenting the algorithm, we describe subroutines from prior work, which we use in our algorithm.

The algorithm requires an efficient distributed scheduling routine, which allows us to run several distributed protocols in parallel on the same network, as we define next. The problem of efficient scheduling has been extensively studied in the past; for more background on this topic, see the introduction of [21] and the references therein.

The Distributed Algorithm Scheduling (DAS) Problem: Given protocols \( A_1, \ldots, A_k \) the network produces an execution so that for each algorithm, each vertex outputs the same value as if that algorithm was run alone.

For a protocol \( A \), let dilation \( (A) \) be its maximal round complexity, and let congestion \( (A, e) \) be the maximal number of messages passing on \( e \). For a DAS instance, its dilation is the maximal dilation over all protocols, and the congestion is \( \max_{e \in E} \left\{ \sum_{i=1}^{\ell} \text{congestion}(A_i, e) \right\} \).

We use the result of Ghaffari [21].

Theorem 20 (Theorem 1.3 [21]) For any instance of DAS, there is a randomized distributed algorithm using only private randomness that, with high probability, produces a schedule that runs all the algorithms in \( O(\text{dilation} \cdot \log n) \) rounds, after \( O(\text{dilation} \cdot \log^2(n)) \) rounds of pre-computation.
In addition, in our algorithm we use [12]’s variant of the decomposition of [7], which was previously stated above in Theorem 13.

9.2 High-level overview of the algorithm

The main idea of our algorithm is to construct several centralized components, which are connected components of the graph that have high internal connectivity, allowing some node of the component to learn all internal edges of the component. Some edges of the graph cross between the centralized components, but not too many, so we can afford for all players to learn all cross-cluster edges. We can then think of each centralized component as a “single player”, with the ability to perform any local computation on the graph of the centralized component, and use these “players” to solve the subgraph-freeness problem, by searching for copies of the subgraph that are partitioned between the players.

In the spirit of the notations used for expander decompositions, the inter-component edges of the centralized component decomposition are denoted by $E_r$, and the intra-component edges are denoted by $E_m$.

Let $A$ be the set of vertices that are incident to some inter-player (that is, inter-cluster) edge. The number of inter-cluster edges is very small, and hence $A$ is also small. In order to find copies of the subgraph $H$, we enumerate over all possible partial mappings $r$ of the vertices of $H$ onto vertices in $A$ (except the “empty” mapping, which maps no vertices), and check whether each such mapping can be completed into a valid mapping of $H$ onto $G$. There are only $n^{2-\Omega(1/k)}$ different partial mappings of up to $k$ vertices from $H$ onto $A$, and each mapping can be checked in constant time.

We say that a partial mapping $r$ is partially consistent if the following condition holds: for every edge $\{u_1, u_2\} \in H$, if the vertices $u_1, u_2$ of $H$ are mapped onto vertices $v_1, v_2$ of $G$ that lie in different centralized components, then the edge $\{v_1, v_2\}$ is in $E_r$. (To check for induced subgraphs, we also need to verify the converse direction.) Since all inter-player edges are known to the entire graph, each player can locally verify that a partial mapping is partially consistent, without any communication. Mappings that are not partially consistent are discarded.

Next, the players try to complete the partial mapping $r$ into a full mapping, using only internal vertices and edges the centralized clusters to fill in the “missing pieces”. (The mapping $r$ already “decided” which edges of $H$ are mapped onto inter-player edges, and we verified that those edges are present, so it remains to fill in the rest of the graph using only internal edges.)

After removing the edges of $H$ that the mapping $r$ mapped onto inter-player edges, we are left with vertex-disjoint, connected subgraphs $H_1, \ldots, H_k$ of $H$. Each such “missing piece” $H_i$ can be completed only by one centralized component: there is at least one vertex $u \in H_i$ for which $r(u)$ is defined, and belongs to some centralized component $C$ (otherwise $H_i$ would not be a separate connected component when we remove the edges of $H$ that are mapped onto $E_r$). Component $C$ is the only component that can “fill in” the vertices in $H_i$ for which $r$ is not defined: any attempt to use a vertex from a different centralized component would require the use of a new inter-player edge, but we are not allowed to use any such edges at this point. Thus, we can assign ownership of each piece $H_i$ to some player (centralized component), and have that player check whether or not it can extend $r$ in order to map $H_i$ onto its vertices and edges. Each player announces to the other players whether or not it was able to complete all the pieces for which it is responsible; if all pieces were completed, then we have found a copy of $H$, and we reject. Otherwise we move on to the next partial mapping.

9.3 Decomposition into centralized components

Formally, the properties we require of the decomposition are as follows:

Definition 9 (Centralized component decomposition) Given two parameters $\delta \in (0, 1)$ and an integer $S > 0$, a centralized component decomposition is a decomposition of $E(G)$ into two parts, $E_m$ and $E_r$, such that

- The graph induced by $E_m$ is composed of connected components $\{R_i\}$, such that in each $R_i$ there is a special root vertex that knows all edges in the subgraph $G(R_i)$. Each such connected component is called a centralized component.
- $|E_r| \leq n^{2-25}S$, and all vertices in $G$ know all edges in $E_r$.

In order to construct the centralized component decomposition, we rely on the following two lemmas, which, very roughly speaking, show that the relation of two cluster having many disjoint paths between them is an equivalence relation (i.e. transitive, symmetric and reflexive). Furthermore, if we have a set of clusters with high conductance and “sufficiently large” minimum cuts between them, then the nodes of these clusters can find many edge-disjoint paths between these clusters efficiently.

The first lemma is a well known fact of high conductance graphs; for the sake of completeness, we include a proof.

Lemma 21 Let $C$ be a graph with conductance $\Phi$ and minimum degree $n^\delta$. Then for any two vertices $x, y \in C$, there exist a set of $\Phi \cdot n^\delta$ edge-disjoint paths between $x$ and $y$ in $C$.

Proof Let $P$ be the edges of a minimum cut between $x, y$, and denote by $V_x, V_y$ the vertices in $x$’s and $y$’s side of
the cut respectively. As the minimum degree of both $x, y$ is at least $n^\delta$, \( \min(\Vol(V_x), \Vol(V_y)) \geq n^\delta \). By definition of conductance (Definition 3), \( \Phi \leq |P|/n^\delta \), or equivalently, \( |P| \geq \Phi \cdot n^\delta \).

**Lemma 22** For \( \epsilon \in (0, 1), \delta > 6\epsilon, \) and \( S \in [n], \) let \( U \) be a graph, \( C \) a set of \( n^\delta \)-clusters in \( U \) with conductance at least \( 1/n^{6\epsilon} \), and let \( p = |C| \). Further assume that there is an ordering of the clusters of \( C, C_1, \ldots, C_p \) with the property that for each \( j = 2, \ldots, p \), there exists an index \( i \in \{1, \ldots, j-1\} \) such that the minimum cut separating \( C_j \) from \( C_i \) has size at least \( S \). Then for each cluster vertex \( x \in \bigcup_{i=1}^p C_i \) and for each \( y \in C_1 \),

1. There exist \( t = \min(S, n^{\delta-6\epsilon}/2) \) edge-disjoint paths
   \[ \{P^t_{x,y}\}_{\ell \in [t]} \]
   connecting \( x \) and \( y \) in \( U \).
2. Suppose that each vertex of \( U \) knows all edges which are not internal to a cluster in \( C \), and that every cluster vertex \( v \in C_j \) knows all edges of \( C_j \) (for all \( j = 1, \ldots, p \)). Then each vertex \( u \) in \( U \) can locally decide which of its edges (if any) participate in each path \( P^t_{x,y} \), for any \( \ell \in [t] \).

**Proof** To prove (1), we prove equivalently that there does not exist a cut of size smaller than \( t \) separating a vertex in one of the clusters of \( C \) from a vertex in \( C_1 \). Let us assume by contradiction that this is not true. Let \( C_1 \) be the first cluster in the ordering such that there exists a set \( R \) of edges, such that \( |R| < t \) and removing \( R \) separates a vertex \( v_j \in C_j \) from a vertex \( v \in C_1 \). If \( j = 1 \), then by Lemma 21 there exist a set of edge-disjoint paths between vertices \( x, y \) of cardinality \( n^\delta/n^{6\epsilon} = n^{\delta-6\epsilon} \). For \( j > 1 \), let \( i < j \) be an index such that \( C_i \) and \( C_j \) have \( S \) edge-disjoint paths. Therefore, they must be connected by at least one path, \( P_{j,i} \), that is disjoint from \( R \). Let \( u_j \in C_j \) be the origin of \( P_{j,i} \) in \( C_j \), and let \( u_i \in C_i \) be the end point of that path. Also note that as \( C_j \) has high conductance, and high min-degree, it does not have an induced cut smaller than \( n^{\delta-6\epsilon} \). Therefore, there is a path \( P_{v_j, u_i} \) from \( v_j \) to \( u_i \) that is disjoint from \( R \). Hence, \( P_{v_j, u_i} \circ P_{j,i} \) is a path from \( v_j \) to \( v_i \) that is disjoint from \( R \). As we assumed \( C_j \) is the first cluster that has a vertex not connected to a vertex in \( C_1 \) in \( U^c = (V, E \setminus R) \), \( v_i \) must be connected to \( u_i \). Therefore \( v_j \) is also connected to \( v_i \) through \( P_{v_j, u_i} \circ P_{j,i} \circ P_{v_i, u_1} \), which is a contradiction, thus proving (1).

To prove (2), we use a theorem by Nash-Williams [33], stating the following: for any undirected graph \( C \) with minimum cut \( c \) there exists a set of \( c/2 \) edges-disjoint spanning trees of \( C \). Hence, each cluster has \( t \) edge-disjoint spanning trees. Let \( U^c \) be a graph in which every cluster \( C_i \) is represented by a node \( v_i^c \) and every edge with an endpoint \( v \in C_i \) is replaced by an edge with the endpoint \( v_i^c \). Since all vertices in \( U \) know all non inner-cluster edges, it follows that all vertices know the graph \( U^c \). For every \( x, y \), all vertices in \( U \) compute the paths \( \{P^\ell_{x,y}\}_{\ell \in [t]} \) in \( U^c \), identically and locally. If path \( P^\ell_{x,y} \) enters the cluster \( C_j \) via an edge that in \( U \) leads to vertex \( u \in C_j \), and leaves it via an edge leaving \( v \in C_j \), the vertices of \( C_j \) complete the path in \( U \) using the path on the \( \ell \)th spanning tree of \( C_j \).

**Lemma 23** Computing the decomposition can be done in \( O(n^2/S + n^{2-\delta/7+o(1)} + n^{1+\delta}) \) rounds.

**Proof** Assume the network is given parameters \( \delta \in (0, 1) \) and \( S > 0 \) (which informally represent the min-degree inside clusters, and the inter-connectivity between clusters inside the same equivalence class respectively), and let \( \epsilon = \delta/14 \). First, using Theorem 13 the network computes the decomposition of [12], setting \( E_m, E_s, E_r \leftarrow T_\delta,\epsilon(G) \) in \( \tilde{O}(n^{1-\delta+20\epsilon}) \) rounds. Following this, each vertex broadcasts to the entire network its id, its cluster-ID, and all of the edges incident to it that are in \( E_s \cup E_r \). As there are at most \( \tilde{O}(n^{1-\delta} + m^{1-\epsilon}) \) and at most \( n \) cluster-vertices, broadcasting this information can be done in \( \tilde{O}(n^{1+\delta} + n^{2-2\epsilon}) \) rounds. In addition, all vertices learn all edges in their cluster. This can be done using the routing scheme described in Theorem 12 in \( \tilde{O}(n^{2-\delta+12\epsilon}) \) rounds.

Let \( G(E_s \cup E_r) \) be the graph induced by edges of \( E_s \) and \( E_r \). Define a min-cut between two clusters \( C_i, C_j \) as the minimum-sized set of edges in \( G \) that their removal disconnects \( C_i, C_j \) in \( G \). Two clusters are said to have a large cut if their min-cut is larger than \( S \), and otherwise they are said to have a small cut. Let \( M(C_i, C_j) \) be a minimal-sized cut between \( C_i, C_j \) if it is small, and otherwise \( M(C_i, C_j) = \emptyset \). Let \( E_r = \bigcup_{i \neq j} M(C_i, C_j) \), and let \( G' = (V, E \setminus E_r) \) be the graph that is obtained from \( G \) by removing the set of small-cuts edges. Note that \( |E_r| \leq \left( \frac{n^1}{2} \right) \cdot S = \tilde{O}(n^{2-2\epsilon}) \cdot S \), as there are at most \( n^{1-\delta} \) clusters, and for each pair we add to \( E_r \) at most \( S \) edges.

Let the cluster graph \( G' \) be the graph where each cluster is a node, and each two nodes have an edge iff their clusters have a large min-cut. Note that since the edges of \( E_s \) and \( E_r \) are known to all vertices in \( G \), all vertices know \( G' \).

The vertices of \( G' \) compute an ordering on the clusters by computing locally and identically a BFS of the graph \( G' \), and order the clusters by the order their respective nodes were discovered by in the BFS. The above order defines a partial ordering \( P_{i,j} \) on the clusters of each component \( CC_i \). For every cluster \( C_j \in CC_i \) except for the first cluster according to \( P_{i,j} \), the cluster that discovered it in the BFS \( C_j \), precedes it in \( P_{i,j} \) and the minimal separating cut between them in \( G \) is larger than \( S \).

By Lemma 22(1) on \( U = G \) and \( C \) as the clusters of a connected component \( CC_i \) in \( G' \), we get that for each connected component \( CC_i \) in \( G' \), all vertices in the first cluster in \( P_{i,j} \) are connected by \( t \) disjoint paths to each of the other clusters vertices in the component.
For each connected component \( CC_i \) in \( G' \), the vertices of the component chooses the cluster vertex with the highest ID to be its “root” vertex, \( u_i^* \), and each vertex compute \( t = \min(n^{\delta-6e}/2, S) \) disjoint paths from every cluster vertex in \( CC_i \) to \( u_i^* \) using the algorithm described in Lemma 22 part (2).

Next, each cluster-vertex sends all its edges that are in \( E_m \) to \( u_i^* \) by dividing the edges (almost) equally between the \( t \) disjoint paths using Theorem 20. We define the set of algorithms the network has to schedule to be \( \{A_e\}_{e \in CC_i} \) where \( A_e \) is the task of sending \( e \) to \( u_i^* \). The congestion on any given edge resulting from a single cluster-vertex \( v \) is at most \( O(d(v)/t) \) and the dilation is at most \( O(n) \). The total congestion on an edge is therefore at most \( \sum_{v \in V} d(v)/t = O(n^2/t) \). By Theorem 20, sending all edges can be done in \( \tilde{O}(n^2/t + n) \) rounds.

The decomposition takes \( \tilde{O}(n^{1-\delta+20e}) \) rounds. Broadcasting the edges of \( E_v \) and \( E_r \) takes \( \tilde{O}(n^{1+\delta} + n^2) \) rounds. Having each vertex learn all edges in its cluster takes \( \tilde{O}(n^{2-\delta+12e+o(1)}) \). Having the root vertex learn all edges takes \( \tilde{O}(n^2/t + n) \) rounds. Therefore, the total round complexity is \( O(n^{1-\delta+20e} + n^{1+\delta} + n^2 + n^{2-\delta+12e+o(1)} + n^2/t) \). Since \( e = \delta/14 \) give us a complexity of \( O(n^{1+\delta} + n^{2-\delta+7e+o(1)} + n^2/t) = \tilde{O}(n^{1+\delta} + n^2 + n^{2-\delta+7e+o(1)} + n^2) / \min(n^{\delta-6e}/2, S) = \tilde{O}(n^{1+\delta} + n^{2-\delta+7e+o(1)} + n^2) / \min(n^{\delta-7}/2, S) = \tilde{O}(n^{1+\delta} + n^{2-\delta+7e+o(1)} + n^2/S) \). \( \Box \)

### 9.4 Detailed description of the H-freeness algorithm

Let \( 0 < \delta \leq 1 \) and \( S \in [n] \) be parameters of the algorithm that will be set subsequently.

First, the network computes the decomposition of Lemma 23 with \( \delta \) and \( S \). Let \( \{CC_i\} \) denote the centralized components of \( G_m = (V, E_m) \). Detecting an instance of \( H \) that is completely contained in one of the centralized components \( CC_i \) can now be done in zero rounds, by having the root vertex check for it. Hence, we only need to consider instances of \( H \) that contain vertices from different centralized components.

Let \( A \) be the set of vertices incident to an edge in \( E_r \). The network sequentially considers each set \( A \subset A \) of vertices of size at most \( k \), and each mapping \( r_A : A \rightarrow V(H) \) from vertices of \( A \) to vertices of \( H \). For each such mapping \( A, r_A \), the network checks if there exists a mapping \( r : G \rightarrow H \) that is consistent with \( r_A \).

For each \( A, r_A \), if there were edges in \( E_r \) that were mapped into \( H \), we remove these edges from \( H \), and denote the obtained graph \( H' \). We associate each connected component in \( H' \) with a centralized component in \( G_m \) as follows. Let \( H' = H'_1 \cup \ldots \cup H'_j \) be the set of connected components of \( H' \). Observe that every connected component in \( H' \) contains at least one vertex from \( A \), and that all vertices in \( A \) that are mapped into the same connected component \( H'_i \) belong to the same centralized component in \( G_m \), which we denote \( CC_{r_A}(H'_i) \). Note that it might be the case that \( CC_{r_A}(H'_i) = CC_{r_A}(H'_j) \) for \( i \neq j \). Symmetrically \( CC_{r_A}^{-1}(CC_i) \) is a mapping from the centralized components of \( G_m \) into sets of connected components of \( H' \).

Now, the root vertex of each centralized component \( CC_i \) of \( G_m \) detects if there exists an extension of \( r_A \) from the vertices of \( CC_i \setminus A \) to \( CC_{r_A}^{-1}(CC_i) \). If there is such a mapping then it sends it to smallest-ID root vertex. Observe that the union of all partial extensions with \( r_A \) is a mapping of an induced instance of \( H \) into \( H' \). This is the case since edges of \( H \) are either mapped from two vertices in \( A \) or two vertices from a single centralized components \( CC_i \) of \( G_m \). In the first case, the mapping \( r_A \) ensures that this edge is mapped to an edge in \( H' \), and in the latter case this is ensured by the partial extension computed by \( CC_i \). As for “non-edges”, assume towards a contradiction that there exists an edge \((u, v) \in G \) that is mapped to a non-edge in \( H' \). If both \( u \) and \( v \) belong to the same \( CC_i \) then this contradicts the extension computed by the root vertex of \( CC_i \). Otherwise, \( u \) and \( v \) belong to different centralized components, implying that \( u, v \in A' \) and therefore in \( A \), and this contradicts the partial mapping \( r_A \). Hence, the smallest-ID root vertex can detect if the union of all the partial mappings is a mapping of an induced instance.

#### 9.5 Analysis and round complexity

**Correctness:** We show that the algorithm accepts if and only if there is an instance, or an induced instance (depending on what we wish to check), of \( H \) in \( G \). We give the proof for induced instances, as the proof for non-induced is very similar.

If there exists an induced instance of \( H \) in \( G \), and it is contained in a connected component of \( G_m \), then it is found by the root vertex of that connected component. Otherwise, let \( A, r_A \) be the set of vertices in \( A \) that are part of the instance, and the mapping of them to \( H \) respectively. When the algorithm considers \( A, r_A \) for the vertices incident to \( E_r \) and these vertices’ their mapping to the instance respectively, it accepts, as each connected component \( H'_i \) of \( H' \) is discovered by the centralized component \( CC_{r_A}(H'_i) \).

If the algorithm accepts, then there is a choice of \( A \) and mapping \( r_A \), and some extensions of mappings in the connected components of \( G' \) that together make \( H \). It remains to show that these mappings do not contradict one another. As different mappings map vertices from different connected components, no vertex can appear in two mappings, and vertices between mappings cannot be neighbors. The parts of \( H \) that are mapped in different extensions of \( A \) also do not share vertices, and their edges are disjoint. Therefore, the mappings do not contradict.
Round Complexity: The round complexity of computing the decomposition of Lemma 23 is $\tilde{O}(n^2/S + n^{2-\delta/2} + o(1)) + n^{1+\delta})$.

Each choice of a set $|A| \leq k$, and every mapping from $A$ to $V(H)$ requires at most $|A|$ root vertices in the algorithm to send at most $O(1)$ messages to the smallest-ID root vertex. Therefore, the process of verifying each choice of $A$, $r_A$ has dilation of at most $O(n)$ and congestion of at most $O(1)$. Therefore, they can all be run in $(n^2 - 2\delta S)^k = \tilde{O}(n^{k(2-\delta)}S^k)$ rounds using the scheduling algorithms of Theorem 20.

Therefore the total complexity is $\tilde{O}(n^{k(2-\delta)}S^k + n^2/S + n^{2-\delta/2} + o(1) + n^{1+\delta})$. Taking $\delta = 1 - \frac{2}{\log n}$, and $S = n^{\frac{1}{\log n}}$ we get that the round complexity is $\tilde{O}(n^{2-\frac{2}{\log n}} + n^{2-1/7+2/(2k+7)+o(1)})$. In particular, if $k \geq 5$ this gives $\tilde{O}(n^{2-\frac{2}{\log n}} + n^{1.88}) = \tilde{O}(n^{2-\frac{2}{\log n}})$ rounds.

We note that for $k \leq 4$, there is a trivial $O(n)$ algorithm for $H$-freeness, in which every node sends its entire neighborhood to each of its neighbors. It can be shown by a simple case analysis that if the network is not $H$-free, there exists a node that can detect a copy of $H$.

10 Hardness of lower bounds for $C_{2r}$-freeness and triangle freeness

We show that any sufficiently strong lower bound on $C_{2r}$-freeness or on triangle-freeness in CONGEST would imply new circuit lower bounds — specifically, an explicit polynomial depth lower bound for circuits with constant fan-in and fan-out, input size $N$, and $N^{1+\alpha}$ wires (where $\alpha > 0$ is a constant). This means that such bounds are unlikely to be proven without a major breakthrough in circuit complexity. In comparison, the best known lower bound for the size of a circuit on an explicit problem in a $O(\log N)$ depth circuit with arbitrary binary Boolean operator gates is $(3 + 1/86)N - o(N)$ (see e.g. [25]), and it is considered a major open problem to obtain an explicit super-linear wire lower bound for these circuits.

The barrier is proven by showing a reduction from solving the problem in general graphs to solving it in high-conductance graphs, and then showing that high-conductance clusters can simulate the class $C$ of circuits described above. This implies that in order to rule out an efficient CONGEST algorithm, one must also rule out the existence of a low-depth circuit in the class $C$.

Throughout this section, we assume that we are working with graphs that have diameter $O(\log n)$; as we showed in Sect. 6, this assumption is without loss of generality. Having small diameter allows us to quickly perform some global operations, such as computing the number of edges in the graph.

10.1 Hardness of lower bounds for $C_{2r}$-freeness

In this subsection we show that an $\tilde{O}(n^{1-\frac{1}{\log n}})$ lower bound on $C_{2r}$-freeness in the CONGEST model would imply a circuit lower bound. The constant 1/361^2 in the exponent results mostly from the parameters of the expander decomposition, which we use in our reduction.

As we said above, the reduction has two steps. We begin by showing that solving $C_{2r}$-freeness on any graph $G$ can be reduced with high constant probability to solving $C_{2r}$-freeness on a graph $G'$, such that $G'$ has small mixing time.

One ingredient in the reduction is the following lemma, which shows that for every edge $e \in G$, it can be decided with constant probability $p$ whether $e$ participates in an $r$-cycle in $O(r^2)$ rounds. The proof is a simplified version of the proof of Theorem 3 in [16].

Lemma 24 (Simplified version of Thm. 3 in [16]) Given a graph $G = (V, E)$, an edge $e \in E$, a value $r \in \mathbb{N}$ and a value $p < 1$, there exists a randomized $O(r^2 \log 1/p)$-round
algorithm that with probability at least $1 - p$ detects if $e$ participates in an $r$-cycle. At the end of the algorithm, the endpoints of edge $e$ know whether or not they participate in an $r$-cycle.

**Proof.** We follow the proof of Theorem 3 in [16].

We first assign each node $w \in V$ a uniformly random value $\text{color}(w) \in [r]$, and remove all directed edges $(u, v)$ which are not colored $(i, (i + 1) \mod r)$ for some $i \in r$. Note that any directed cycle that remains must have length at least $r$.

Next, we start a $(r - 1)$-round BFS from the smaller-ID endpoint of edge $e$. The BFS may cross only edges that were not discarded. After $r - 1$ rounds, if the larger-ID endpoint of $e$ receives the BFS token, then we have found an $r$-cycle that includes $e$.

Each invocation of the procedure above has one-sided error probability of $1 - 1/r^{r-1}$: if edge $e$ does not participate in an $r$-cycle, then it will never receive its BFS token after $r$ rounds, because we have eliminated all cycles of length smaller than $r$; and if $e$ does participate in an $r$-cycle, then with probability $1/r^{r-1}$, the cycle will be colored in ascending cyclic order (mod $r$), starting from the smaller endpoint of edge $e$ and ending at the larger endpoint. When this event occurs, the cycle is detected. To boost the success probability to $1 - p$, we repeat $O(r^{r-1} \cdot \ln(1/p))$ times, using new independently-chosen colors each time. 

We are now ready to prove the first part of the reduction, where we reduce from general graphs to graphs with high conductance (i.e., low mixing time).

**Lemma 25** Let $r > 360$, and $\alpha = \frac{362r}{\log 100}$. Solving $C_{2r}$-freeness on any graph $G$ can be reduced with high constant probability in $O(n^{1 - \frac{1}{2}})$ rounds to solving $C_{2r}$-freeness on a graph $G'$ with conductance $\tilde{\Omega}(n^{-\frac{180}{180}} - \frac{1}{180})$.

**Proof.** By Theorem 1 of Bondy and Simonovits [2], any graph $G = (V, E)$ with $|E| \geq 100r \cdot n^{1+1/r}$ has a cycle $C_{2r}$ for every $j \in [r, rn^{1/r}]$. Therefore, we can assume that $G$ has $|E| < 100r \cdot n^{1+1/r}$, as otherwise we can simply reject. (Recall that we assume logarithmic diameter, so we can afford to count the edges of the graph.)

We use the $(\epsilon, \phi)$-expander decomposition of Chang and Saranurak [8], defined next. An $(\epsilon, \phi)$-expander decomposition of a graph $G = (V, E)$ is defined as a partition of the vertex set into $\ell$ clusters, $V = \bigcup_{i=1}^{\ell} V_i$, satisfying the following conditions:

1. For each cluster $V_i$, the conductance of $G_i$ satisfies $\Phi(G[V_i]) \geq \phi$.
2. Let $E_{ic}$ denote the set of inter-cluster edges $E_{ic} = \partial(V_1) \cup \ldots \cup \partial(V_\ell)$, where $\partial(V_i)$ is the set of edges between $V_i$ and $V \setminus V_i$. Then $|E_{ic}| < \epsilon|E|$.

**Theorem 21** ([8], Theorem 1) For $\epsilon \in (0, 1)$, and a positive integer $k$ an $(\epsilon, \phi)$-expander decomposition with $\phi = (\epsilon/\log n)^{2\cdot180}$ can be constructed in $O((\epsilon n)^{1/k})$.

We set $k = 2$ and $\epsilon = n^{-\frac{1}{2} - \frac{1}{2}}$, and compute an $(\epsilon, \phi)$-decomposition for $G$ so that for every $i$, $\Phi(G[V_i]) = \Omega(\epsilon/\log n)^{2\cdot180} = \Omega(n^{-\frac{180}{180}} + \frac{1}{180})$. The round complexity is $O((\epsilon |E|)^{1/k}) \cdot (\log n)^{2\cdot180} = O(n^{-\frac{1}{2} - \frac{1}{2}} + \frac{1}{180})$.

For every inter-cluster edge $e \in E_{ic}$, we invoke the algorithm described in Lemma 24 with error parameter $p = 1/(10\epsilon|E|)$ in order to decide if $e$ participates in a 2r-cycle. This is done sequentially, and since there are at most $\epsilon|E| = O(n^{1 - \frac{1}{2}})$ inter-cluster edges, this process terminates in $O((2r)^{2r} \cdot \epsilon |E| \cdot \log 1/p) = \tilde{O}(n^{1 - \frac{1}{2}})$ rounds. By taking a union bound over the edges in $E_{ic}$, it holds that if any edge $e \in E_{ic}$ participates in a 2r-cycle in $G$, then it would have been detected with high probability.

If no edge in $E_{ic}$ detects a cycle, then it remains to decide whether there exists a 2r-cycle within any of the clusters $V_i$. This is true since all the inter-cluster edges were removed, so that any existing 2r-cycle will be completely contained in one of the clusters. Therefore, solving $C_{2r}$-freeness on $G$ is reduced to solving $C_{2r}$-freeness on $\ell$ components in parallel, each having conductance $\Phi = \tilde{\Omega}(\epsilon/\log n)^{2\cdot180} = \tilde{\Omega}(n^{-\frac{180}{180}} + \frac{1}{180})$. The number of rounds required for the reduction is $\tilde{O}(n^{1 - \frac{1}{2}} + n^{\frac{1}{2} + \frac{1}{180}})$ which for $r > 360$ and $\alpha = \frac{362r}{\log 100}$ is bounded by $\tilde{O}(n^{1 - \frac{1}{2}})$.

Next, we show that “simple” circuits can be simulated in the CONGEST model in a number of rounds that depends on the mixing time of $G$, as well as the number of wires and the depth of the circuit. We first formally define the notion of a circuit.

**Definition 10** A circuit $C$ is a directed acyclic graph, where the nodes of the graph represent gates from some class of Boolean functions, and the edges represent wires. The depth of $C$ is the path of the longest path from any input node to any output node. The fan-in of the circuit is the maximal in-degree of any node in the graph, and the fan-out is the maximal out-degree.

In [14], Drucker, Kuhn and Ooshman prove that if a function $f$ can be computed by a circuit $C$ of depth $R$ with $b$-separable gates and $N = n^2 \cdot s$ wires, then there exists an $O(R)$-rounds protocol $P$ for $f$ in the CLIQUE-CAST model on graphs with $n$ vertices and bandwidth $O(b + s)$, where $b$-separability is some measure of a gate’s simplicity.

Here we prove a weaker variant of the above that is suitable for the CONGEST model.

**Lemma 26** Let $U$ be a graph $U = (V, E)$ with $|V| = n$ vertices and $|E| = n^{1+\delta}$ edges, with mixing time $\tau_{mix}$. Suppose
that \( f : \{0, 1\}^{cn^{1+\delta} \log n} \rightarrow \{0, 1\} \) for some \( c > 1 \) is computed by a circuit \( C \) of depth \( R \), comprising of gates with constant fan-in and fan-out, and at most \( O(c \cdot s \cdot n^{1+\delta} \log n) \) wires for \( s \leq n \). Then for any input partition that assigns to each vertex in \( U \) no more than \( c \deg(v) \log n \) input wires, there is an \( O(R \cdot c \cdot s \cdot \tau_{\text{mix}} \cdot 2^O((\log n \log \log n))) \)-round protocol in the CONGEST model on \( U \) that computes \( f \) under the input partition.

**Proof** The proof follows the steps of the proof of Theorem 2 in [14]:

Assigning gates to vertices. Since each gate has at least one wire, and the total number of wires is \( c \cdot s \cdot n^{1+\delta} \log n \), there can be at most \( c \cdot s \cdot n^{1+\delta} \log n \leq cs \cdot |E| \log n \) gates. Therefore, we can partition the gates between the vertices, such that each vertex \( v \) is assigned \( O(c \cdot s \cdot \deg(v) \log n) \) gates, where \( \deg(v) \) is the degree of \( v \).

Evaluating the circuit. As in [14], we partition the gates of \( C \) into \( R + 1 \) layers \( L_0, \ldots, L_R \) according to their depth in the circuit, with \( L_0 \) being the input wires and \( L_R \) the output wires. We evaluate the circuit in \( R \) stages, each corresponding to one layer of the circuit. We shall prove that we can evaluate all gates in a single layer in \( O(c \cdot s \cdot \tau_{\text{mix}} \cdot 2^O((\log n \log \log n)) \) rounds, thus concluding the proof.

We prove by induction on the number of layers, \( R + 1 \). The input layer \( L_0 \) requires no evaluation. Therefore, assume that we have already evaluated the circuit up to layer \( L_{i-1} \), and we wish to evaluate \( L_i \). In order to compute all gates in \( L_i \), each player needs to learn the values of the input wires of its assigned gates. Since each vertex \( v \) is assigned at most \( O(c \cdot s \cdot \deg(v) \log n) \) gates and each gate has constant fan-in, \( v \) needs to learn \( O(c \cdot s \cdot \deg(v)) \) bits. As \( U \) has mixing time \( \tau_{\text{mix}} \), by Theorem 12, this takes \( O(c \cdot s \cdot \tau_{\text{mix}} \cdot 2^O((\log n \log \log n)) \) rounds. \( \square \)

**Corollary 3** Let \( C \) be a constant fan-in and fan-out circuit \( C \) with depth \( R \) and \( N = s \cdot n^{1+\delta} \log n \) wires for \( s \leq n \) that computes \( C_{2r} \)-freedom for any given graph over \( n \) vertices. Then there exists a CONGEST protocol that solves \( C_{2r} \)-freedom for \( G \) in \( \tilde{O}(n^{1-1/\alpha} + n \cdot R \cdot n^{360/\alpha} + n^{360/\alpha} \cdot 2^O((\log n \log \log n))) \) rounds.

**Proof** Let \( G \) be the subgraph at the hand. By Lemma 25, we can reduce \( C_{2r} \)-freedom on \( G \), to solving \( C_{2r} \)-freedom on \( \ell \leq n \) edge-disjoint clusters \( V_1 \cup \ldots \cup V_\ell \) of \( G \) in \( \tilde{O}(n^{1-1/\alpha}) \) rounds. Furthermore, by Lemma 25 each of the clusters has conductance \( \Phi(G(V_i)) = \tilde{\Omega}(n^{1-1/\alpha} + n^{160/\alpha}) \), so that by Theorem 7, each cluster has mixing time \( \tau_{\text{mix}}(V_i) = \tilde{O}(n^{160/\alpha} + n^{360/\alpha}) \). Therefore, by Lemma 26, for each cluster \( V_i \) there exists an \( O(s \cdot R \cdot \tau_{\text{mix}}(V_i)) = \tilde{O}(s \cdot R \cdot n^{360/\alpha} + n^{360/\alpha} \cdot 2^O((\log n \log \log n))) \) protocol for solving \( C_{2r} \)-freedom on \( G(V_i) \). As the clusters are edge-disjoint, these protocols can be computed in parallel. Therefore, \( C_{2r} \)-freedom on \( G \) can be computed in \( \tilde{O}(n^{1-1/\alpha} + n \cdot R \cdot n^{360/\alpha} + n^{360/\alpha} \cdot 2^O((\log n \log \log n))) \) rounds. \( \square \)

The above implies that there exists a constant \( c \geq 1/361^2 \) such that any lower bound of the form \( \tilde{O}(n^{1-\epsilon}) \) on \( C_{2r} \)-freedom imply lower bounds far beyond the reach of current circuit complexity techniques.

**Corollary 4** For any constant \( c \geq 1/361^2 \) such that if \( C_{2r} \)-freedom for graphs over \( n \) vertices cannot be computed in CONGEST in \( \Omega(n^{1-\epsilon}/2^O((\log n \log \log n))) \) rounds, then \( C_{2r} \)-freedom cannot be computed by constant fan-in and fan-out circuits with \( s \cdot n^{1+\delta} \) wires for \( s = n^{(1-\epsilon)/2} \) and depth \( R = \tilde{O}(n^{1-\epsilon}/2 + 2^O((\log n \log \log n))) \).

**Proof** By Corollary 3, \( C_{2r} \)-freedom on \( G \) can be computed in \( \tilde{O}(n^{1-1/\alpha} + n \cdot R \cdot n^{360/\alpha} + n^{360/\alpha} \cdot 2^O((\log n \log \log n))) \) rounds.

Since \( \alpha \) is decreasing with \( r \) and \( r > 360 \), we have that \( \alpha \leq 362^2 \). Therefore, the above bound is bounded by \( \tilde{O}(n^{1-1/\alpha} + n \cdot R \cdot n^{360/\alpha} + n^{360/\alpha} \cdot 2^O((\log n \log \log n)) + n^{1-1/\alpha} \). Hence, any lower bounds of the form \( \Omega(n^{1-\epsilon}/2^O((\log n \log \log n))) \) for \( c \leq 1/361^2 \) on the number of rounds required to compute with high constant probability \( C_{2r} \)-freedom on a graph \( G \) over \( n \) vertices in the CONGEST model, imply a lower bound of \( \tilde{O}(2^O((\log n \log \log n))) \) on \( R \) for constant fan-in and fan-out gate circuits with \( sn^{1+\delta} \log n \) wires. \( \square \)

### 10.2 Barrier of \( \Omega(n^{\alpha}) \) for lower bounds on triangle freeness

We now turn our attention to triangles, and show that for any constant \( \alpha > 0 \), a round complexity lower bound of \( \Omega(n^{\alpha}) \) on triangle freeness implies strong circuit complexity lower bounds. This reduction is strongly based on the algorithm of [7,8] for triangle enumeration, which in essence reduces the problem to solving a related problem on high conductance graphs. In combination with Lemma 26 above, we obtain the barrier.

The first step is to reduce from general graphs to graphs with high conductance: let \( A_2 \) be an algorithm that solves triangle freeness in a communication network with conductance \( \Phi \) and \( n \) nodes, where every node is given \( O(\deg(v)) \) edges as additional input (which are not necessarily edges \( v \) is incident to). Let \( T_2(n, \Phi) \) be the round complexity of \( A_2 \). We show that given such an algorithm we can solve triangle freeness in \( O(A_2(n, \text{polylog} n \log n)) \) rounds in CONGEST.

First, the network computes the decomposition of Theorem 21 with \( \epsilon = 1/6 \), taking \( k \) to be a large enough constant so that the round complexity of the decomposition is less than \( O(n^{\alpha}) \). A node is called good if it has more edges in \( E_m \) than in \( E_r \), and otherwise bad. We call an edge \( e \in E_m \) bad if at least one of its endpoints is bad.

**Lemma 27** ([7]) The number of bad edges is at most \( 2en \).

After computing the decomposition, each cluster \( C \) calculates the number of vertices and the number of edges in
the cluster. Since $\Phi = \tilde{O}(1)$, the diameter of each cluster is $\tilde{O}(1)$, and therefore this can be done in $\tilde{O}(1)$ rounds (for example, by constructing a spanning tree on the cluster, and collecting the number of edges and nodes up the tree). Then, each cluster runs $A_2$ in parallel, where the input of each good node is all its edges (including edges leaving the cluster), and for each bad node, its edges in the cluster. We note that indeed by the definition of a good node, every node has $O(\deg_C(v))$ edges as input for $A_2$, where $\deg_C(v)$ is $v$’s degree in its cluster. If $A_2$ outputs that there exists a triangle, the cluster rejects and terminates. Otherwise, each cluster $C$ removes all good edges which are contained in $C$. The network then recurses on the remaining edges until $O(1)$ edges remain. Since $|E_{\text{good}}| = m - |E_{\text{bad}}| - em \geq m - 3em = m/2$, in each iteration the network removes half of its edges, and the number of iterations are at most $O(\log n)$.

Clearly, if a node rejects then the graph contains a triangle with high probability (by the correctness of $A_2$). On the other hand, recall that the input graph of $A_2$ consists of the edges incident to good nodes in $C$; therefore if $A_2$ returns that there is no triangle, the network may safely remove all edges between two good nodes, as the triangle is contained in the union of inputs of both good endpoints. This concludes the reduction from general graphs to graphs with high conductance.

Next, we show how to simulate any circuit from a “simple” family of circuits on graphs with high conductance.

Consider the following family of functions $f_N : \{0, 1\}^{2N \log^2 N} \rightarrow \{0, 1\}$: given an encoding of a graph $\mathcal{G}$ with at most $N$ edges, $f$ returns 1 iff the graph contains a triangle.

**Corollary 5** *If triangle freeness cannot be solved in less than $c_1n^\alpha$ rounds for any $c_1 > 0$, then there exist constants $c_2, c_3 > 0$ such that there is no family of circuits that solve for all $N$ the function $f_N$ with $c_2N^{\alpha/4}/2^{c_3\log n \log \log n}$ depth and at most $c_2N^{1+\alpha/4}/2^{c_3\log n \log \log n}$ wires.*

**Proof** Let $\mathcal{F}$ be an infinite family of graphs for which triangle freeness cannot be solved in less than $c_1n^\alpha$ rounds. Let $c_4 > 0$ be a constant such that the conductance of the clusters obtained by Theorem 21 is greater than $1/\log^{c_4} n$. Assume by contradiction that for sufficiently large $c_2$, $c_3$ there is an infinite family of circuits solving for any $N$ the function $f_N$ with $c_2N^{\alpha/4}/2^{c_3\log n \log \log n}$ depth and $c_2N^{1+\alpha/4}/2^{c_3\log n \log \log n}$ wires. Then by Lemma 26 taking $s = n^{\alpha/2}/2^{c_3\log n \log \log n}$ and $R = n^{\alpha/2}/2^{c_3\log n \log \log n}$ (both of which are larger than $c_2N^{\alpha/4}/2^{c_3\log n \log \log n}$) there exists an algorithm with round complexity $c_1n^\alpha/\log n$

---

4 For a graph with $N$ edges $\{(u_i, v_i)\}_{i=1}^{N}$, the graph is encoded by the string $u_{1,i}id, v_{1,i}id, \ldots, u_{N,i}id, v_{N,i}id$, where each id is padded to $\lceil \log N \rceil$ bits, and the rest of the string is padded in such a manner that indicates that there are no further edges.
