Sublinear-Time Distributed Algorithms for Detecting Small Cliques and Even Cycles

Talya Eden
Electrical Engineering Department, Tel-Aviv University, Israel
talyaa01@gmail.com

Nimrod Fiat
Computer Science Department, Tel-Aviv University, Israel
nimrodf1@tau.ac.il

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

Fabian Kuhn
Computer Science Department, University of Freiburg, Germany
kuhn@cs.uni-freiburg.de

Rotem Oshman
Computer Science Department, Tel-Aviv University, Israel
roshman@mail.tau.ac.il

Abstract

In this paper we give sublinear-time distributed algorithms in the CONGEST model for subgraph detection for two classes of graphs: 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 listed in sublinear time, $O(n^{5/6+o(1)})$ rounds and $O(n^{21/22+o(1)})$ rounds, respectively. Prior to our work, it was not known whether it was possible to even check if the network *contains* a 4-clique or a 5-clique in sublinear time.

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 $\Omega(n^{1/2+\delta})$ lower bound on C_{2k}-freeness implies new lower bounds in circuit complexity.

For general subgraphs, it was shown in [14] 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 of [14].

2012 ACM Subject Classification Networks → Network algorithms; Theory of computation → Distributed algorithms; Theory of computation → Lower bounds and information complexity

Keywords and phrases Distributed Computing, Subgraph Freeness, CONGEST

Digital Object Identifier 10.4230/LIPIcs.DISC.2019.15

Related Version A full version of the paper is available at `https://www.cs.tau.ac.il/~roshman/papers/DISC19_EFFKO19.pdf`.

Funding *Talya Eden*: The Israel Science Foundation grant No.1146/18. Talya Eden is greatful to the Azrieli Foundation for an award of an Azrieli Fellowship.
Nimrod Fiat: The Israel Science Foundation grant No.1146/18 and the Kadar family award.
Orr Fischer: The Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11) and

the Laura Schwarz-Kipp Institute of Computer Networks.
Rotem Oshman: The Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

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 [27, 28, 25]).

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 [11, 10, 23, 19, 21, 14]. 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 odd-length cycles, are known to require linear time [11], and some subgraphs even require nearly-quadratic time [14]. In contrast, for triangles [5] and for even-length cycles [14], sublinear-time algorithms are known.

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* $\tilde{O}(n^{5/6+o(1)})$ *rounds in* CONGEST*, and all 5-cliques can be enumerated in* $\tilde{O}(n^{21/22+o(1)})$ *rounds.*

Our algorithm builds on a recent approach of [5], 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 [5].

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 [11]). Our improved algorithm exploits a new connection to extremal combinatorics: we show that the *Zarankiewitz 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 [14] and improve its running time:

▶ **Theorem 2.** *C_{2k}-freeness can be solved in at most $\tilde{O}_k(n^{1-2/(k^2-k+2)})$ rounds for odd $k \geq 3$, and at most $\tilde{O}_k(n^{1-2/(k^2-2k+4)})$ rounds for even $k \geq 4$.*

For example, for 6-cycles we improve the running time from $\tilde{O}(n^{5/6})$ (in [14]) to $\tilde{O}(n^{3/4})$ 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 *check* if 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.** *Enumerating all C_4 in the graph requires $\tilde{\Theta}(n)$ rounds.*

Obstacles on proving lower bounds for even-length cycles. For any $k \geq 2$, the lower bound for C_{2k}-freeness has been "stuck" at $\Omega(\sqrt{n})$ for a long time [11, 21]. 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, 21, 8], cannot be used to give a lower bound better than $\tilde{\Omega}(\sqrt{n})$ on C_6-freeness. Following [8], 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 adjacent 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 $\tilde{O}(\sqrt{n} \cdot s)$ bits and solves C_6-freeness.*

Our protocol uses different ideas from the two-party protocol of [8]; to bound the number of edges the players need to send each other, we rely on results from extremal combinatorics.

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}-hardness 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 [11] between lower bounds for the Congested Clique and circuit complexity to high-conductance components.

General subgraphs. It was shown in [14] that some subgraphs are very hard to detect: for any $k \geq 4$, there exists a graph on k vertices that requires $\tilde{\Omega}(n^{2-\Theta(1/k)})$ 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 [14] 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 k and subgraph H of size k, the H-freeness problem can be solved in $O(n^{2-2/(3k+1)+o(1)})$ rounds in* CONGEST*.*

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 lack of space, we mention here the most directly related results. In [5, 6] 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 [20]. The algorithms of [5, 6] finds a *conductance decomposition* of the graph, and then uses a routing result of [17, 18] to quickly find all triangles contained or adjacent to some cluster in the decomposition. Our K_4-listing algorithm uses the decomposition from [5], albeit in a somewhat different manner from the way it is used in [5]. We also use the K_s-listing algorithm for the Congested Clique of [10] as a subroutine; [10] shows that all copies of K_s can be found in $O(n^{1-2/s})$ in the Congested Clique. To our knowledge, prior to our work, no sublinear-time K_s-freeness algorithm was known for any $s \geq 4$. However, in [1], it is shown that if the network contains an ϵ-*near-clique* of linear size, then an ϵ^3-near clique of linear size can be found in constant time. For 4-cycles, C_4, a nearly-tight bound of $\tilde{\Theta}(\sqrt{n})$ was shown in [11]. The lower bound was extended to an $\tilde{\Omega}(\sqrt{n})$ lower bound for any even-length cycle C_{2k} in [21]. In the Congested Clique, an $O(n^{0.158})$ round algorithm for C_k-detection was shown by [4] based on algebraic methods. The first sublinear-time algorithm for C_{2k}-freeness for $k \geq 3$ was given in [14], and we improve this algorithm here. It is known that odd-length cycles require nearly-linear time to detect [11]. It is shown in [14] that some subgraphs of size $O(k)$ require $\tilde{\Omega}(n^{2-\frac{1}{k}})$ rounds to detect (for any constant $k \geq 4$). There are algorithms for clique-freeness and cycle-freeness in related models, e.g., [4, 10, 3, 16, 13, 15], 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 [10] by *simulating* it in regular CONGEST.

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

▶ **Definition 7** (Subgraph freeness and enumeration)**.** *Fix a 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. 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.*

For a graph G, we let $V(G)$ denote the vertex set of G, and $E(G)$ denote the edges of G.

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 $O(\sqrt{m})$ [7].

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 $O(n^{5/6+o(1)})$ rounds. Throughout the section, we will use two parameters: $\epsilon = 1/2, \delta = 5/6$. Since some parts of the algorithm will be re-used in later sections with different values for ϵ, δ, 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_s and E_m, by recursively applying a graph decomposition from [5]. The set E_s has the property that every node v has at most n^δ edges in E_s; therefore, cliques contained entirely in E_s are easy to find, by simply having all nodes announce to all their neighbors all their edges in E_s.

The set E_m is the edge-disjoint union of $O(n^{1-\delta})$ "very well-connected" clusters of nodes. On each such cluster, we can fairly efficiently simulate the Congested Clique algorithm for finding 4-cliques of [10]. However, we need to find *all* 4-cliques that have at least one edge in E_m. This includes cliques $\{u_0, u_1, u_2, u_3\}$ such that $\{u_0, u_1\} \in E_m$, 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 $\{u_2, u_3\}$ 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 would require too much time. We resolve this difficulty by considering two types of "external nodes". 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 conductance decomposition from [5] in a slightly different manner than [5] uses it, and then how we find 4-cliques on the resulting decomposed graph.

3.1 Conductance Decomposition

A main ingredient in the algorithm is the conductance decomposition developed in [5], 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 low-degree graph, and $|E_r| \leq |E|/6$. The well-connected clusters in E_m each satisfy the following:

▶ **Definition 8** (n^δ-cluster). *A subset $G' = (V', E')$ of G is called an* n^δ-cluster *(or simply "cluster" for short) if it satisfies the following conditions:*

1. *It has $O(\text{polylog}(n))$ mixing time,*

2. *End each vertex $v \in V'$ has degree $\Omega(n^\delta)$ in E'.*

The *mixing time* of a graph is the time required for a random walk on the graph to approach its stationary distribution; the precise definition is not needed for our purposes here, because as in [5], we use the mixing-time guarantee to apply a routing result from [17] as a black box.

In [5], the algorithm finds triangles by (a) decomposing the edge-set, (b) finding triangles that include an edge in E_s or E_m, and then (c) recursing on E_r. (Edges are sometimes moved from E_m to E_r, but not too many.) In contrast, for our purposes here, we must first eliminate all the edges of E_r: we have no guarantee on these edges other than the fact that there are not too many of them, and to find a 4-clique touching E_m or E_s that may include edges from E_r, this is not sufficient. By recursively applying the decomposition from [5], we obtain the following decomposition of the graph.

▶ **Lemma 9.** *Fix a graph $G = (V, E)$ with $|V| = n$ and diameter D. We can find, w.h.p., in $\tilde{O}(n^{1-\delta} + D)$ rounds, a decomposition of E to $E = E_m \cup E_s$ satisfying the following conditions:*

(*a*) *E_m is the union of at most $s = O(\log n)$ sets, $E_m = \bigcup_{i=1}^{s} E_m^i$, where each E_m^i is the vertex-disjoint union of $O(n^{1-\delta})$ n^δ-clusters, $C_i^1, \ldots, C_i^{k_i}$.*
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.*

(*b*) *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.

(*c*) *$E_s = \bigcup_{v \in V} E_{s,v}$, where $E_{s,v}$ is a subset of edges incident to v and $|E_{s,v}| \leq n^\delta \log n$. Each vertex v knows $E_{s,v}$.*

To assign the clusters unique identifiers and leaders, we rely on our "diameter reduction" technique, which allows us to assume w.l.o.g. that the diameter of the network is $D = O(\text{polylog}(n))$ (See full version for details [12]). Thus, in $\tilde{O}(n^{1-\delta})$ rounds, we can select the smallest node in each cluster, and disseminate the IDs of these nodes throughout the network.

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

3.2 Finding 4-Cliques

In the remainder of the section we describe our 4-clique listing algorithm; the runtime analysis and correctness proof appear in the full version of the paper [12].

We begin by computing the decomposition from Lemma 9. 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 all of $E_{s,v}$ to all its neighbors. As $|E_{s,v}| \leq n^\delta$, this can be done in n^δ rounds. Any node that sees a copy of K_4 outputs it. It is easy to verify that any 4-clique whose edges are all in E_s will be detected by all four of its vertices.

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 ϵ). Given a cluster C, we say that a node v is *C-light* if $v \notin C$ and v has at most n^ϵ 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*.

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. 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 C, all nodes that belong to C (i.e., have at least one edge in C) announce this fact 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; this requires $O(n^\epsilon)$ rounds, as v is C-light. Upon receiving $M_C(u)$ from each C-light neighbor $u \in N(u)$, node v forms a list of "candidates" — triplets of nodes that may complete a K_4 with v:

$$\mathcal{L}_v = \{\{u, c_1, c_2\} \mid u \in N(v) \text{ is } C\text{-light}, c_1, c_2 \in C, \text{ and } c_1, c_2 \in M_C(u) \cap M_C(v)\}.$$

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",

$$\mathcal{Q}_{v,c_1} = \{\{c_1, c_2\} \mid \exists u \in N(v) : \{u, c_1, c_2\} \in \mathcal{L}_v\},$$

the list of potential edges of c_1 that, if present, would complete a 4-clique. There are at most n^ϵ such edges: by definition of $\mathcal{L}_v$, 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 $(\{c_1\} \times N(c_1)) \cap \mathcal{Q}_{v,c_1}$ 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.

Let $F(C) = \{\{u, v\} \mid u \in C \text{ or } u \text{ is } C\text{-heavy}\}$ be the set of all edges adjacent to 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 $F(C_i^j)$, in parallel for all the step-i clusters, $C_i^1, \ldots, C_i^{k_i}$. To do this, we have the cluster nodes simulate the execution of the Congested Clique algorithm for K_4 enumeration from [10] 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., $|V_{c_1}| \approx |V_{c_2}|$ for each $c_1, c_2 \in C$). Each node $c \in C$ will be responsible for simulating the nodes in V_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 [17].

IV. **Simulate**: the nodes of C simulate an n-vertex Congested Clique, and run the K_4-enumeration algorithm of [10] on the graph $(V, F(C))$ to list all the 4-cliques in $F(C)$.

▶ Remark 10. It is worth noting that [5] gives an extension of the Congested Clique algorithm of [10] 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 adjacent to cluster nodes, but instead on a potentially much larger set, $F(C)$. In addition, we simulate an Congested Clique of size n on a n^δ-cluster, while in [5] 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.

We elaborate on each step below.

Pull. For each level-i cluster C in parallel, the nodes do the following. For a node v, let $S(v) = \{\{v, u\} \mid u \in N(v) \wedge v < u\}$ be the edges adjacent to v in which v has the smaller ID.

Each C-heavy node v that is not in C partitions its edges $S(v)$ into $|M_C(v)|$ sets, each of size at most $|S(v)|/|M_C(v)| \leq n/n^\epsilon = n^{1-\epsilon}$, and sends each set to a different neighbor in C, in $O(n^{1-\epsilon})$ rounds.

Following this step, each edge in $F(C)$ is known to exactly one node in C; we define the *initial load* of node $c \in C$ to be the number of $F(C)$-edges it knows (including its own edges).

▷ Observation 11. Since each C-heavy node partitions its edges into at least n^ϵ sets, and $c \in C$ has at most n neighbors that are C-heavy, the initial load of c is at most $n \cdot n^{1-\epsilon} = n^{2-\epsilon}$.

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 this 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^δ nodes, which we call the *active nodes*, to carry out the simulation; the cluster leader selects these nodes by choosing n^δ of its neighbors, $u_1, \ldots, u_{n^\delta}$. The leader also selects an ℓ-wise independent hash function $f : U \to [n^\delta]$, 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, $\{(i, u_i)\}_{i=1,\ldots,n^\delta}$, and the $O(\log^2 n)$-bit representation of f to all the cluster nodes. Using pipelining, this requires $\tilde{O}(n^\delta)$ rounds. (We note that since the cluster has polylogarithmic mixing time, it also has polylogarithmic diameter.)

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 [24], for $O(\log n)$-wise independent random variables, we obtain:

▶ **Lemma 12.** *Let $\ell = 4 \log n$, and assume that f is ℓ-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.*

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 [10], where each node in C simulates at most $O(n^{1-\delta})$ nodes of G. For both purposes we use the routing scheme of [17], which allows us to deliver roughly n^{δ} 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. These details are deferred to the full version of the paper. Ultimately, we show the following result:

▶ **Lemma 13.** *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 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 $O((n^{2-\delta-\epsilon} + t \cdot n^{2-2\delta}) \cdot n^{o(1)})$ rounds, with success probability $1 - \frac{1}{n^2}$.*

Using this simulation, each cluster simulates the K_4 enumeration algorithm of [10]. 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 $O(n^{2-2\delta+1/2})$. This completes our high-level overview of the K_4-enumeration algorithm.

3.3 Listing 5-Cliques

We outline at a high level the changes needed to list all 5-cliques.

The main difficulty in moving from 4-cliques to 5-cliques is that a 5-clique can be partitioned between clusters, so that **(a)** each edge of the 5-clique belongs to a different cluster (recall that the clusters constructed in Lemma 9 are not vertex-disjoint), and furthermore, **(b)** no cluster is able to "pull in" all the edges of the 5-clique, because from the perspective of each cluster, of the three nodes outside the cluster, two are light and one is heavy.

We modify our framework so that it can handle 5-cliques by making two main changes:

(1) We do not call the decomposition from [5] recursively. Instead, we execute only one step of the decomposition, which yields a partition of the edges E into a set of well-connected, vertex-disjoint clusters; a sparse set, E_s; and the "remainder", E_r, which is of size at most $|E_r| \leq \alpha \cdot |E|$.

(2) We change the parameters of the decomposition so that E_r, the "remaining edges", constitute a *sub-constant* fraction of the total number of edges (i.e., $\alpha = o(1)$), see [9]. (We pay for this by an increased mixing time.) This implies that the arboricity of E_r is *sublinear*, as any graph with m edges has arboricity at most $\sqrt{m}$.

We are then able to overcome the difficulties and find all copies of K_5 using the fact that the edges outside the clusters have sublinear arboricity, and that the clusters are vertex-disjoint; the resulting running time is $O(n^{21/22+o(1)})$ rounds.

4 Improved Algorithm for C_{2k}-Freeness

We now turn out attention to even-length cycles, and give an improved algorithm for C_{2k}-freeness.

Our main technical contribution is to show that, because of a new connection to Zarankiewicz numbers, if we have a graph that is C_{2k}-free, then this graph cannot have too many high-degree nodes. The bound we obtain is tighter than a bound used in [14], and it yields an improved algorithm for C_{2k}-freeness.

▶ **Definition 14** (Zarankiewicz numbers [26]). *The Zarankiewicz number $z(a, b, H)$ is defined as the maximum number of edges in an (a, b)-bipartite graph that does not contain H as a subgraph.*

We rely on the following upper bound:

▶ **Theorem 15** (Zarankiewicz numbers for cycles [26, 22]). *For any integers $a, b \geq 0$ and $k \geq 2$,*

$$z(a, b, C_{2k}) \leq \begin{cases} (2k-3)\left((ab)^{\frac{1}{2}+\frac{1}{2k}} + a + b\right), & \text{for odd } k, \\ (2k-3)\left(a^{\frac{k+2}{2k}} b^{\frac{1}{2}} + a + b\right), & \text{for even } k. \end{cases}$$

Using Theorem 15, we can better bound the number of high-degree vertices in C_{2k}-free graphs.

▶ **Lemma 16.** *Let $c = c(k)$ be a large enough constant, and fix a graph $G = (V, E)$. For any $d \geq cn^{1/k}$, let $V_d = \{v \in V \mid d(v) \geq d\}$. If G is C_{2k}-free, then*

$$|V_d| \leq \begin{cases} O_k(\max(n^{\frac{k-1}{k+1}}, (n^{\frac{k+1}{2k}}/d)^{\frac{2k}{k-1}})), & \text{for odd } k \geq 3, \\ O_k(\max(n^{\frac{k}{k+2}}, (n^{1/2}/d)^{\frac{2k}{k-2}})), & \text{for even } k \geq 4. \end{cases}$$

Here, the notation $O_k(\cdot)$ hides constants that depend on k.

For a graph H and an integer n, let $\mathrm{ex}(H, n)$ be the Túran number of H - the maximum number of edges in an n-vertex H-free graph.

Because the calculations become tedious for general k, we include here a (somewhat informal) proof for 6-cycles ($k = 3$), which conveys the general ideas, and defer the general case to the full version of the paper [12].

Proof of Lemma 16 for $k = 3$. We need to show that $|V_d| \leq O(\max(n^{1/2}, (n^{2/3}/d)^3))$.

Define the following sets of edges:

- $E_d = E \cap (V_d \times V)$: edges adjacent to some node in V_d.
- E_{int}: the internal edges $E_{int} = E \cap (V_d \times V_d)$ of V_d.
- $E_{ext} = E_d \setminus E_{int}$: the external edges $E_{ext} = E \cap (V_d \times (V \setminus V_d))$ adjacent to some node in 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_6-free, so is the subgraph G_d induced on G by V_d; therefore, $|E_{int}| \leq \mathrm{ex}(|V_d|, C_6) = O(|V_d|^{1+1/k})$, where $\mathrm{ex}(N, C_6) = O(N^{1+1/3})$ [2]. Because we required that $d \geq cn^{1/k}$, we have $|E_{int}| = O(|V_d|^{1+1/k}) = O(n^{1/k}|V_d|) \overset{\text{choice of } c}{\leq} (d \cdot |V_d|)/4$. Therefore,

$$|E_{ext}| = |E_d| - |E_{int}| \geq d \cdot |V_d|/2 - d \cdot |V_d|/4 = \Omega(d \cdot |V_d|). \tag{1}$$

In other words, if V_d is large, then the cut between V_d and $V \setminus V_d$ is also large. However, to this cut we can apply Theorem 15: the bipartite graph induced by G on $V_d \times (V \setminus V_d)$ is C_6-free, because all of G is C_6-free; by Theorem 15,

$$|E_{ext}| \leq z(|V_d|, |V \setminus V_d|, C_6) \leq z(|V_d|, n, C_6) = O((|V_d| \cdot n)^{2/3} + n). \tag{2}$$

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

Our bound follows from (2), depending on which of the two terms, $(|V_d| \cdot n)^{2/3}$ or n, is larger. If $(|V_d| \cdot n)^{2/3} = O(n)$, then $|V_d| \leq O(n^{1/2})$. Otherwise, from (2) we have $|E_{ext}| = O((|V_d| \cdot n)^{2/3})$, and together with (1), we obtain $|V_d| = O((n^{2/3}/d)^3)$, completing the proof. ◀

The C_{2k}-algorithm from [14] uses the bound on the number of high-degree nodes as follows. First, we search for a C_{2k} that contains a high-degree node: we go over these nodes one after the other, and starting a BFS from each one to check if it participates in a copy of C_{2k}. Subsequently, the high-degree nodes are removed from the graph, together with all their edges. Next, we rely on an observation already used in [11], which is that a C_{2k}-free graph has *arboricity* at most $O(n^{1/k})$; in particular, its vertices can be quickly partitioned into $O(\log n)$ layers $V^1, \ldots, V^\ell$, $\ell = \Theta(\log n)$, such that the number of edges from any node in layer V^i to all higher layers $\bigcup_{j>i} V^j$ is $O(n^{1/k})$. Together with the fact that all high-degree nodes have been removed from the graph, this partition allows us to quickly find any remaining copies of C_{2k} in the graph.

For an integer d, let $V_d = \{v \mid d(v) \geq d\}$ be the set of vertices with degree at least d. Putting both parts together, the running time of the C_{2k} algorithm from [14] is given by:

$$\tilde{O}_k(|V_{n^\delta}| + n^{(k-2)\delta + \frac{1}{k}}), \tag{3}$$

where $\delta \in (0, 1)$ is a parameter that determines the threshold for what is considered "high degree". In [14], the value of δ is chosen fairly naïvely: since a C_{2k}-free graph has at most $O(n^{1+1/k})$ edges, for any δ we have $|V_{n^\delta}| \leq O(n^{1+1/k-\delta})$, and balancing the two terms in (3) yields the result.

Our C_{2k} algorithm retains the framework described above, but uses the bound from Lemma 16 to bound $|V_d|$. This allows us to choose a smaller value for δ, lowering the threshold for what we consider a "high-degree node". We obtain the following improved C_{2k}-freeness algorithm. Again, we focus here on the case of 6-cycles, and the full version, which involves even more calculations, appears in the full version of the paper [12].

▶ **Theorem 17.** *C_6-freeness can be solved in $\tilde{O}(n^{3/4})$ rounds in the* CONGEST *model.*

Proof. Recall from (3) that after choosing a degree threshold n^δ, we can eliminate any potential 6-cycle involving a node of V_{n^δ} in $O(|V_{n^\delta}|)$ rounds, and then check the remaining graph in $O(n^{\delta+1/3})$ rounds. Setting $\delta = 5/12$, we have by Lemma 16 that $|V_{n^{5/12}}| = O(n^{3/4})$ and also $n^{\delta+1/3} = n^{3/4}$, yielding the desired running time. ◀

5 Subquadratic Algorithm for Subgraph-Freeness for Any Subgraph

In [14], it was shown that for any $k \geq 1$, there is a subgraph H_k of size $|H_k| = O(k)$ such that randomized H_k-freeness requires $\tilde{\Omega}_k(n^{2-\frac{1}{k}})$ rounds. (The notation $\tilde{\Omega}_k$ hides factors that depend only on k, which we think of as constant.) The bound approaches quadratic as k grows, but in subgraph-freeness, we typically think of k as a constant rather than a growing function. This leads to the question: is there a single subgraph H of some fixed size, such that H-freeness requires $\tilde{\Omega}(n^2)$ rounds?

In this section we give a negative answer, by showing an upper bound that nearly matches the lower bound from [14]:

▶ **Theorem 18.** *For any constant $k \geq 4$ and any subgraph H of size k, there is an algorithm for H-freeness and exact H-counting in $\tilde{O}(n^{2-\frac{2}{3k+1}+o(1)})$ rounds.*

The same upper bound also holds for *induced* subgraph-freeness and counting. Furthermore, the algorithm can be modified to enumerate all copies of H, in $\Theta(n^{2-\Theta(1/k)} + \sqrt{\#H})$ rounds, where $\sqrt{\#H}$ is the number of copies present. We can also show that there exist graphs H which require $\Omega(\sqrt{\#H})$ rounds to enumerate all copies.

Our algorithm begins by decomposing the edges of the graph into two sets, E_m, E_r, with the following properties. Here, $\delta = 1 - \Theta(1/k)$ and $S = n^{\Theta(1/k)}$ are parameters whose exact values will be fixed later.

(1) The graph induced by E_m is composed of maximal connected components $CC = \{R_i\}_{i\in[C]}$ (where C is the number of connected components), which we refer to as *centralized components*. In each centralized component R_i, there is a special *root vertex* $r_i \in V(R_i)$, whose identity is known to all vertices in R_i. Moreover, the root vertex knows all the edges of the subgraph $G\{R_i\}$ (the subgraph induced on G by the vertices $V(R_i)$).

(2) $|E_r| \leq n^{2-2\delta}S$, and each vertex in G knows all the edges in E_r.

The decomposition is computed in $\tilde{O}(n^2/S + n^{2-\delta/10} + n^{1+\delta})$ rounds. We give a brief overview, and then explain how to use the centralized components.

5.1 Computing the Centralized Components

The first step in computing the decomposition is to run the decomposition from [9] with different parameters than the ones used in [9]. It returns a partition of the graph into three edge sets, $E = E_{m'} \cup E_{s'} \cup E_{r'}$, where

(a) $E_{s'}$ has bounded arboricity,

(b) $E_{r'}$ is subquadratic in size,

(c) $E_{m'}$ consists of vertex-disjoint high-conductance clusters with high minimum degree.

Next, we divide the clusters of $E_{m'}$ into *equivalence classes*, where two clusters are in the same cluster iff they have many edge disjoint paths between them. We prove that indeed this relation is transitive: if a cluster A has many edge disjoint paths to clusters B and C, then clusters B, C have many disjoint paths between them. The resulting equivalence classes have high connectivity, and therefore, some vertex in each class can learn all the edges in that class in sub-quadratic time. Moreover, the cut separating any two distinct equivalence classes is bounded in size (otherwise they would be the same class).

The algorithm takes E_r be the set of all edges that participate in some cut between two distinct equivalent classes; there are at most $n^{\Theta(1/k)}$ such edges, because the cuts are not too large. Finally, let $E_m = E \setminus E_r$.

5.2 Finding Copies of H

After computing the decomposition, we can immediately detect any copy of the subgraph H whose vertices all belong to the same centralized component R_i, because each such copy is known to the root vertex $r_i \in R_i$. We can also find any copies of H whose edges are contained entirely in E_r, because all nodes know E_r. It remains to find copies of H that include some edges from E_m and some from E_r; that is, copies of H that include at least one vertex from some centralized component, but also some edge from E_r. To find such "split" copies of H, we "guess" which vertices of H should be mapped to a vertex inside some centralized component, and which vertices of H should be mapped to vertices adjacent to edges of E_r, and then verify that we can "stitch together" a complete copy of H by having the root vertices locally check that their centralized component contains the missing pieces and that E_r connects everything properly. This must be done carefully, to ensure that we do not detect false copies of H; we now describe the process in more detail.

Mappings and partial mappings of H. A copy of H can be represented by a mapping $\rho : V(H) \to V(G)$ of the vertices of H to the vertices of G, such that for any $x, y \in V(H)$, if $(x, y) \in E(H)$ then $(\rho(x), \rho(y)) \in G$. We call such a mapping a *good total mapping of* H.

Given a good total mapping ρ of H, we are interested in understanding how the copy of H witnessed by ρ is split between centralized components, so that we can "stitch it together". Let $V_r \subseteq V(G)$ be the set of vertices adjacent to the edges of E_r. We distinguish between two types of vertices:

- *Border vertices:* vertices $x \in V(H)$ such that $\rho(x) \in V_r$. These are vertices serve as potential "stitching points", because they are adjacent to the edges E_r that interconnect the centralized components.
- *Internal vertices:* vertices $x \in V(H)$ such that $\rho(x) \in (\bigcup_i V(R_i)) \setminus V_r$.

(This is a partition, i.e., any vertex of H is either a border vertex or an internal vertex.)

A *border mapping* is a mapping $\sigma : V(H) \to V_r \cup \{*\}$, which specifies for some vertices $x \in V(H)$ a target $\sigma(x) \in V_r$ onto which they are mapped in G, and leaves some vertices unmapped, $\sigma(x) = *$. Intuitively, a border mapping specifies the "interface" between a copy of H whose existence we are trying to verify, and each of the centralized components. We say that a border mapping σ is *good* if there exists a good total mapping ρ of H, such that

- ρ extends σ, that is, for any $x \in V(H)$, if $\sigma(x) \neq *$, then $\rho(x) = \sigma(x)$; and,
- ρ does not add any border vertices, that is, for any $x \in V(H)$ such that $\sigma(x) = *$, we have $\rho(x) \notin V_r$.

Clearly, if G contains a copy of H, then there is a good border mapping, obtained by taking a good total mapping of H and replacing any $\rho(x) \notin V_r$ with $*$.

Our algorithm enumerates over all border mappings, and checks whether each one is good. If we find a good border mapping, we reject, as we have found a copy of H. If we have gone over all border mappings and none are good, we accept. It remains to show that we can efficiently check whether a given border mapping is good, and also that there are not too many border mappings to check.

Checking a border mapping. Fix a border mapping $\sigma : V(H) \to V_r \cup \{*\}$ which is known to all nodes of G, and let us describe how the nodes check whether σ is good, i.e., whether it can be extended into a good total mapping by adding internal vertices.

We call an edge $\{x, y\} \in E(H)$ *external* if $\sigma(x) \neq *, \sigma(y) \neq *$ and $\{\sigma(x), \sigma(y)\} \in E_r$, that is, this edge is mapped outside the centralized components. An edge that is not external is called *internal*, and it includes at least one internal node (possibly two).

Each root vertex locally checks which parts of H it is "responsible for filling in", as follows: we delete from H all the external edges (but not their endpoints), obtaining a collection $H_1, \ldots, H_k$ of connected components—at least two, since we assumed that H is not contained inside any centralized component. Observe that for each $x \in V(H)$ such that $\sigma(x) = *$, there is some i such that $x \in V(H_i)$.

A component H_i is *owned by centralized component* R if there is some internal edge $\{x, y\} \in E(H_i)$ such that $\sigma(x) \in V_r \cap V(R)$, that is, an internal edge that "touches" R. Note that each H_i is owned by exactly one centralized component, and furthermore, if H_i is owned by R, then the internal nodes of H_i *must be filled in* using nodes of R: let us say that a total mapping $\rho : V(H) \to V(G)$ *respects ownership* if for any internal node $x \in H_i$ (for some $i = 1, \ldots, k$), if H_i is owned by centralized component R, then $\rho(x) \in V(R)$.

▶ **Lemma 19.** *For any border mapping σ, any good total mapping ρ that extends σ and does not add any border nodes must respect ownership.*

After deciding which centralized components own which parts $H_1, \ldots, H_k$, the centralized components try to locally complete σ by assigning internal vertices they own to vertices inside the centralized component. More concretely, for each i, the root vertex r whose centralized component R owns H_i looks for a *local mapping for* H_i, $\rho_i : H_i \to V(R)$, such that

- ρ_i agrees with σ on all border vertices in H_i, that is, for any $x \in H_i$ such that $\sigma(x) \in V_r$, we have $\rho_i(x) = \sigma(x)$; and,
- Each edge of H_i is mapped onto some edge of E_m inside R. That is, if $\{x, y\} \in E(H_i)$, then $\{\rho_i(x), \rho_i(y)\} \in E_m$.

We say that a root vertex $r \in V(R)$ is *happy* if, for each H_i owned by R, there is a local mapping ρ_i satisfying the requirements above.

So far, everything we described is done locally, with no communication—all nodes know the edges E_r, and consequently the know V_r; and each root vertex knows all edges of E_m inside its centralized component, so it can check if there is a local mapping ρ_i satisfying the requirements.

Finally, each root vertex announces whether it is happy or not, and the network computes an AND over these answers to check if all root vertices are happy. If all are happy, then σ is a good mapping, and the network rejects. Otherwise, we move on to the next border mapping.

If all root vertices are happy, then we can "piece together" the partial mappings $\sigma, \rho_1, \ldots, \rho_k$ into a total mapping ρ, which we prove is good:

▶ **Lemma 20.** *If there exist local mappings $\rho_1, \ldots, \rho_k$ for $H_1, \ldots, H_k$ (respectively), then G contains a copy of H.*

▶ **Corollary 21.** *Given the decomposition into centralized components, we can check whether G contains a copy of H in $O\left(\left(n^{2-2\delta}S\right)^k + D\right)$ rounds.*

Proof. Since $|E_r| \leq n^{2-2\delta}S$, we also have $|V_r| \leq n^{2-2\delta}S$, so the number of border mappings is at most $\left(n^{2-2\delta}S\right)^k$. To check each border mapping, we only need to compute an AND over the happiness of each root vertex; using pipelining, we can compute all these ANDs in parallel, in a total of $O\left(\left(n^{2-2\delta}S\right)^k + D\right)$ rounds. ◀

6 Hardness of Proving Lower Bounds For C_{2k}

We give a brief overview of two obstacles on proving strong lower bounds for even-length cycles. All details appear in the full version of the paper [12].

To date, all lower bounds on subgraph-freeness have been shown by reduction from two-party communication complexity, with the players partitioning the network graph between them [11, 14, 21, 8]. The players *simulate* the execution of a distributed algorithm, and the cost of the simulation per round corresponds to the size of the cut between the players' parts. We usually reduce from n-bit set disjointness, which means that we must have (size of the cut) $\cdot$ (number of rounds) $\geq n$ (up to a $\log n$ factor). In [8], it was shown that this approach cannot yield a lower bound greater than $\widetilde{\Omega}(\sqrt{n})$ for 4-cliques, because no matter how we try to partition the graph between the two players, they can solve the K_4-freeness problem using $\widetilde{O}(\sqrt{n} \cdot s)$ bits, where s is the size of the cut between them.

We show an analogous result for 6-cycles, using different ideas: for any fixed partition of the graph between two players, they can solve C_6-freeness using roughly $\sqrt{n} \cdot s$ bits. Our two-party protocol uses the connection to Zarankiewitz numbers that we already exploited in Section 4, to argue that the players can compactly encode paths of length two, three or four

that are entirely contained on their side of the graph. This enables the players to "stitch together" a 6-cycle that is split between them, if there is one.

Next, we show that although it seems plausible that the round complexity of C_{2k}-freeness approaches $\Omega(n)$ as $k \to \infty$, proving such a result would imply very powerful circuit lower bounds. In fact, there is an absolute constant $c \in (1/2, 1)$, such that proving any lower bound greater than $\Omega(n^c)$ on *any* particular even-length cycle is already difficult.

We are inspired by a result from [11], where it is shown that the Congested Clique is able to simulate certain types of circuits, which have very large fan-in, efficiently. We are not able to use this result as-is, nor do we simulate the same type of circuit. Instead, we show that **(a)** the problem of C_{2k}-freeness in general networks can be reduced to solving C_{2k}-freeness in networks with high conductance; and **(b)** using the routing scheme of [17, 18], we show that networks with sufficiently high conductance can simulate constant fan-in circuits of depth n^δ, where δ is a constant. Therefore, a lower bound of the form $\Omega(n^c)$, for some absolute constant $c \in (1/2, 1)$, would imply lower bounds on the size of circuits with constant fan-in and depth n^δ. At present, it is still open to find an explicit function that cannot be computed by a circuit of *linear* size and *logarithmic* depth.

References

1 Zvika Brakerski and Boaz Patt-Shamir. Distributed discovery of large near-cliques. *Distributed Computing*, 24(2):79–89, 2011.

2 Boris Bukh and Zilin Jiang. A bound on the number of edges in graphs without an even cycle. *Combinatorics, Probability and Computing*, 26(1):1–15, 2017.

3 Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. In *Distributed Computing: 30th International Symposium, DISC 2016. Proceedings*, pages 43–56, 2016.

4 Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015*, pages 143–152, 2015.

5 Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 821–840, 2019.

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

7 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on computing*, 14(1):210–223, 1985.

8 Artur Czumaj and Christian Konrad. Detecting cliques in CONGEST networks. In *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, pages 16:1–16:15, 2018.

9 Mohit Daga, Monika Henzinger, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed edge connectivity in sublinear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 343–354, 2019.

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

11 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 367–376, 2014.

12 Talya Eden, Nimrod Fiat, Orr Fischer, Fabian Kuhn, and Rotem Oshman. Sublinear-time distributed algorithms for detecting small cliques and even cycles. `https://www.cs.tau.ac.il/~roshman/papers/DISC19_EFFKO.pdf`.

13 Guy Even, Orr Fischer, Pierre Fraigniaud, Tzlil Gonen, Reut Levi, Moti Medina, Pedro Montealegre, Dennis Olivetti, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. Three Notes on Distributed Property Testing. In *31st International Symposium on Distributed Computing (DISC 2017)*, volume 91 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:30, 2017.

14 Orr Fischer, Tzlil Gonen, Fabian Kuhn, and Rotem Oshman. Possibilities and impossibilities for distributed subgraph detection. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 153–162, 2018.

15 Pierre Fraigniaud and Dennis Olivetti. Distributed detection of cycles. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*, pages 153–162, 2017.

16 Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed testing of excluded subgraphs. In *Distributed Computing: 30th International Symposium, DISC 2016. Proceedings*, pages 342–356, 2016.

17 Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed MST and routing in almost mixing time. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 131–140, 2017.

18 Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In *32nd International Symposium on Distributed Computing (DISC)*, pages 31:1–31:16, 2018.

19 Tzlil Gonen and Rotem Oshman. Lower bounds for subgraph detection in the congest model. To appear in OPODIS 2017, 2017.

20 Taisuke Izumi and François Le Gall. Triangle finding and listing in congest networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '17, pages 381–389, 2017.

21 Janne H. Korhonen and Joel Rybicki. Deterministic subgraph detection in broadcast CONGEST. In *21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, December 18-20, 2017*, pages 4:1–4:16, 2017.

22 Assaf Naor and Jacques Verstraëte. A note on bipartite graphs without 2k-cycles. *Combinatorics, Probability and Computing*, 14(5-6):845–849, 2005.

23 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 405–414, 2018.

24 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 331–340, 1993.

25 Michael Szell and Stefan Thurner. Measuring social dynamics in a massive multiplayer online game. *Social Networks*, 32(4):313 – 329, 2010.

26 Jacques Verstraëte. Extremal problems for cycles in graphs. In *Recent Trends in Combinatorics*, pages 83–116. Springer, 2016.

27 Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994.

28 Duncan J. Watts. Collective dynamics of 'small-world' networks. *Nature*, 393:440, 1998.