Distributed Detection of Cliques in Dynamic Networks

Matthias Bonne
Department of Computer Science, Technion, Haifa, Israel
matt.b@cs.technion.ac.il

Keren Censor-Hillel
Department of Computer Science, Technion, Haifa, Israel
ckeren@cs.technion.ac.il

Abstract

This paper provides an in-depth study of the fundamental problems of finding small subgraphs in distributed dynamic networks.

While some problems are trivially easy to handle, such as detecting a triangle that emerges after an edge insertion, we show that, perhaps somewhat surprisingly, other problems exhibit a wide range of complexities in terms of the trade-offs between their round and bandwidth complexities.

In the case of triangles, which are only affected by the topology of the immediate neighborhood, some end results are:

- The bandwidth complexity of 1-round dynamic triangle detection or listing is $\Theta(1)$.
- The bandwidth complexity of 1-round dynamic triangle membership listing is $\Theta(1)$ for node/edge deletions, $\Theta(n^{1/2})$ for edge insertions, and $\Theta(n)$ for node insertions.
- The bandwidth complexity of 1-round dynamic triangle membership detection is $\Theta(1)$ for node/edge deletions, $O(\log n)$ for edge insertions, and $\Theta(n)$ for node insertions.

Most of our upper and lower bounds are *tight*. Additionally, we provide almost always tight upper and lower bounds for larger cliques.

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

Keywords and phrases distributed computing, subgraph detection, dynamic graphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.132

Category Track C: Foundations of Networks and Multi-Agent Systems: Models, Algorithms and Information Management

Related Version A full version of the paper is available at [5], `http://arxiv.org/abs/1904.11440`.

Acknowledgements This project has received funding from the European Union's Horizon 2020 Research And Innovation Program under grant agreement no. 755839.

1 Introduction

A fundamental problem in many computational settings is to find small subgraphs. In distributed networks it is particularly vital for various reasons, among which is the ability to perform some tasks much faster if, say, triangles do not occur in the underlying network graph (see, eg., [24, 30]).

Finding cliques is a *local* task that trivially requires only a single communication round if the message size is unbounded. However, its complexity may dramatically increase when the bandwidth is restricted to the standard $O(\log n)$ bits, for an n-node network. For example, the complexity of detecting 4-cliques is at least $\Omega(n^{1/2})$ [11]. For triangles, the complexity is yet a fascinating riddle, where only recently the first non-trivial complexities of $\tilde{O}(n^{2/3})$ and $\tilde{O}(n^{3/4})$ have been given for detection and listing, respectively [25], and the current

46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 132; pp. 132:1–132:15


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

state-of-the-art is the $\tilde{O}(n^{1/2})$-round algorithm of [10]. For listing, there is an $\tilde{\Omega}(n^{1/3})$ lower bound [25, 28]. The only non-trivial lower bounds for detection say that a single round is insufficient, as given in [1] and extended for randomized algorithms in [18]. In [1] it was also shown that 1-bit algorithms require $\Omega(\log^* n)$ rounds, improved in [18] to $\Omega(\log n)$.

In this paper, we address the question of detecting small cliques in *dynamic* networks of limited bandwidth. We consider a model that captures real-world behavior in networks that undergo changes, such as nodes joining or leaving the network, or communication links between nodes that appear or disappear. Various problems have been studied in many variants of such a setting (see Section 1.3).

The task of finding cliques is unique, in that it is trivial if the bandwidth is not restricted, and it can be easily guaranteed that at the end of each round all nodes have the correct output. This implies that one does not have to wait for *stabilization* and does not have to assume that the network is *quiet* for any positive number of rounds. If, however, the bandwidth is restricted, the solution may not be as simple, although some problems can still be solved even with very small bandwidth.

As a toy example, consider the case of triangle listing when a new edge is inserted to the graph. The endpoints of the inserted edge simply broadcast a single bit that indicates this change to all of their neighbors, and hence if a new triangle is created then its third endpoint detects this by receiving two such indications.

Nevertheless, we show that this simplified case is far from reflecting the general complexities of clique problems in such a dynamic setting. For example, the above algorithm does not solve the problem of *membership-listing* of triangles, in which *each* node should list all triangles that contain it. Indeed, we prove that this stronger variant *cannot* be solved with constant bandwidth, and, in fact, every solution must use at least $\Omega(\sqrt{n})$ bits.

Our contributions provide an in-depth study of various detection and listing problems of small cliques in this dynamic setting, as we formally define and summarize next.

1.1 Our contributions

For a subgraph H we categorize four types of tasks: Detecting an appearance of H in the network, for which it is sufficient that a single node does so, listing all appearances of H, such that every appearance is listed at least by a single node, and their two *membership* variants, membership-detection and membership-listing, for which each node has to detect whether it is a member of a copy of H, or list all copies of H to which it belongs, respectively.

The model is explicitly defined in Section 1.4. In a nutshell, there can be one topology change in every round, followed by a standard communication round among neighboring nodes, of B bits per message, where B is the bandwidth. An algorithm takes r rounds if the outputs of the nodes are consistent with a correct solution after r communication rounds that follow the last change. In particular, a 1-round algorithm is an algorithm in which the outputs are correct already at the end of the round in which the topology change occurred. Hence, 1-round algorithms are very strong, in the sense that they work even in settings in which no round is free of changes. We note that in what follows, all of our algorithms are deterministic and all of our lower bounds hold also for randomized algorithms.

Triangles. Our upper and lower bounds for triangles ($H = K_3$) are displayed in Table 1.

Most of the complexities in this table are shown to be tight, by designing algorithms and proving matching lower bounds. The one exception is for membership detection with edge insertions, for which we show an algorithm that uses $O(\log n)$ bits of bandwidth, but we do not know whether this is tight. However, we also show a 1-round algorithm for this problem

Table 1 The bandwidth complexities of 1-round algorithms for dynamic triangle problems.

	Node deletions	Edge deletions	Edge insertions	Node insertions
Detection/Listing	0	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Membership detection	0	$\Theta(1)$	$O(\log n)$	$\Theta(n)$
Membership listing	0	$\Theta(1)$	$\Theta(\sqrt{n})$	$\Theta(n)$

that works with a bandwidth of $O((\Delta \log n)^{1/2})$, where Δ is the maximum degree in the graph, implying that if our logarithmic algorithm is optimal, then a proof for its optimality must exhibit a worst case in which the maximum degree is $\Omega(\log n)$.

A single round is sufficient for solving all clique problems, given enough (linear) bandwidth. Nevertheless, for the sake of comparison, we show that with just one additional communication round, all of the bandwidth complexities in Table 1 drop to $\Theta(1)$, apart from membership listing of triangles, whose bandwidth complexity for r-round algorithms is $\Theta(n/r)$.

Larger Cliques. We also study the bandwidth complexities for finding cliques on $s > 3$ nodes. Here, too, a single round is sufficient for all problems, and the goal is to find the bandwidth complexity of each problem. Some of the algorithms and lower bounds that we show for triangles carry over to larger cliques, but others do not. Yet, with additional techniques, we prove that for cliques of constant size, almost all of the 1-round bandwidth complexities are the same as their triangle counterparts. Due to lack of space, the study of s-cliques is deferred to the full version [5].

Combining types of changes. In most cases, one can obtain an algorithm that handles several types of changes, by a simple combination of the corresponding algorithms. However, intriguingly, sometimes this is not the case. A prime example is when trying to combine edge insertions and node insertions for triangle detection: a node obtaining 1-bit indications of a change from two neighbors cannot tell whether this is due to an edge inserted between them, or due to an insertion of a node that is a neighbor of both. In some of these cases we provide techniques to overcome these difficulties, and use them to adjust our algorithms to cope with more than one type of change.

1.2 Challenges and techniques

Algorithms. The main challenge for designing algorithms is how to convey enough information about the topology changes that occur, despite non-trivial (in particular, sublinear) bandwidth. Consider, e.g., edge insertions. As described, while listing triangles is trivial with a single indication bit, this fails for membership detection or membership listing.

For membership detection we can still provide a very simple algorithm for which a logarithmic bandwidth suffices, by sending the identity of the new edge, and by *helping* neighbors in the triangle to know that they are such. For membership listing even this is insufficient. To overcome this challenge, we introduce a technique for sending and collecting *digests* of neighborhood information. When all digests from a given neighbor have been collected, one can determine the entire neighborhood of this neighbor. The caveat in using such an approach in a straightforward manner is that a node needs to list its triangles with a newly connected neighbor already at the same round in which they connected, and cannot wait to receive all of its neighbor's digests. By our *specific choice* of a digests, we ensure that a newly-connected neighbor has enough information to give a correct output after a single communication round, despite receiving only a single digest from its latest neighbor.

Lower bounds. For the lower bounds, our goal is to argue that in order to guarantee that all nodes give a correct output, each node must receive enough information about the rest of the graph. To do this, we identify sequences of topology changes that exhibit a worst-case behavior, in the sense that a node cannot give a correct output if it receives too little information.

One approach for doing this is to look at a particular node x, and define as many sequences as possible such that the correct output of x is different for each sequence. At the same time, we ensure that the number of messages that x receives from the other nodes is as small as possible. These two requirements are conflicting – if x can have many different outputs, it must have many different neighbors, which implies that it receives many messages with information. Still, we are able to find such a family of sequences for each problem, and we wrap-up our constructions by using counting arguments to prove the desired lower bounds. In some cases, e.g., membership-listing under edge insertions, even this is insufficient, and we construct the sequences such that one *critical* edge affects the output of x, but it is *added last*, so that it conveys as little information as possible.

The above can be seen as another step in the spirit of the *fooling views* framework, which was introduced in [1] for obtaining the first lower bounds for triangle detection under limited bandwidth. After being beautifully extended by [18], our paper essentially gives another indication of the power of the fooling views framework in proving lower bounds for bandwidth-restricted settings.

On the way to constructing our worst-case graph sequences, we prove combinatorial lemmas that show the existence of graphs with certain desired properties. To make our lower bounds apply also for randomized algorithms, we rely on Yao's lemma and on additional machinery that we develop.

1.3 Additional related work

Dynamic distributed algorithms. Dynamic networks have been the subject of much research. A central paradigm is that of *self-stabilization* [12], in which the system undergoes various changes, and after some quiet time needs to converge to a stable state. The model addressed in this paper can be considered as such, but our focus is on algorithms that do not require any quiet rounds (though we also address the gain in bandwidth complexity if the system does allow them, for the sake of comparison). Yet, we assume a single topology change in each round. A single change in each round and enough time to recover is assumed in recent algorithms for maintaining a maximal independent set [2, 3, 9, 14, 23] and matchings [31], and for analyzing amortized complexity [29]. Highly-dynamic networks, in which multiple topology changes can occur in each round are analyzed in [4, 7], and it is an intriguing open question how efficient can subgraph detection be in such a setting. Various other dynamic settings in which the topology changes can be almost arbitrary have been proposed, but the questions that arise in such networks address the flow of information and agreement problems, as they are highly non-structured. A significant paper that relates to our work is [26], which differs from our setting in the bandwidth assumption. Clique detection in the latter model is trivial, and indeed their paper addresses other problems.

Distributed subgraph detection. In the CONGEST model and in related models, where the nodes synchronously communicate with $O(\log n)$ bits, much research is devoted to subgraph detection, e.g., [1, 10, 13, 20, 22, 25, 27, 28].

A related problem is property-testing of subgraphs, where the nodes need to determine, for a given subgraph H, whether the graph is H-free or *far* from being H-free [6, 8, 16, 17, 19, 21].

1.4 Preliminaries

In the dynamic setting, the network is a sequence of graphs. The initial graph, whose topology is known to the nodes, represents the state of the network at some starting point in time, and every other graph in the sequence is either identical to its preceding graph or obtained from it by a single topology change. The network is synchronized, and in each round, each node can send to each one of its neighbors a message of B bits, where B is the bandwidth of the network. Each node has a unique ID and knows the IDs of all its neighbors. We denote by n the number of possible IDs. Hence, n is an upper bound on the number of nodes that can participate in the network at all times.

Note that the nodes do not receive any indication when a topology change occurs. A node can deduce that a change has occurred by comparing the list of its neighbors in the current round and in the previous round. This implies that a node u cannot distinguish between the insertion of an edge uv and the insertion of a node v which is connected to u, as the list of neighbors of u is the same in both cases.

An algorithm can be designed to handle edge insertions or deletions or node insertions or deletions, or any combination of these. We say that an algorithm is an r-round algorithm if the outputs of the nodes after r rounds of communication starting from the last topology change are correct. The (deterministic or randomized) r-round bandwidth complexity of a problem is the minimum bandwidth for which an r-round (deterministic or randomized) algorithm exists. We denote by $\mathsf{MemList}(H)$, $\mathsf{MemDetect}(H)$, $\mathsf{List}(H)$, and $\mathsf{Detect}(H)$, respectively, the problems of membership listing, membership detection, listing and detection of H. The following is a simple observation.

▶ **Observation 1.** *Let r be an integer and let H be a graph. Denote by B_{MemList}, $B_{\mathsf{MemDetect}}$, B_{List} and B_{Detect} the (deterministic or randomized) r-round bandwidth complexities of $\mathsf{MemList}(H)$, $\mathsf{MemDetect}(H)$, $\mathsf{List}(H)$, and $\mathsf{Detect}(H)$, respectively, under some type of topology changes. Then: $B_{\mathsf{Detect}} \leq B_{\mathsf{List}} \leq B_{\mathsf{MemList}}$ and $B_{\mathsf{Detect}} \leq B_{\mathsf{MemDetect}} \leq B_{\mathsf{MemList}}$.*

Due to this observation, our exposition starts with the more challenging task of membership-listing, then addresses membership-detection, and concludes with listing and detection. Within each case, we first provide algorithms for each type of topology change, and then prove lower bounds. Missing proofs are deferred to the full version [5].

2 Triangle problems

2.1 Membership listing

Table 2 summarizes our results for membership listing of triangles.

Table 2 Bandwidth complexities of $\mathsf{MemList}(K_3)$.

	Node deletions	Edge deletions	Edge insertions	Node insertions
$r = 1$	0	$\Theta(1)$	$\Theta(\sqrt{n})$	$\Theta(n)$
$r \geq 2$	0	$\Theta(1)$	$\Theta(1)$	$\Theta(n/r)$

2.1.1 Upper bounds

Our algorithms for node and edge deletions appear in the full version [5]. Note that combining these algorithms in a direct manner *does not* handle both types of changes. We provide a more subtle combination of the two algorithms for doing so within $O(1)$ rounds. Handling edge insertions and node insertions is much more complicated. We start by showing an algorithm that handles edge insertions.

▶ **Theorem 2.1.** *The deterministic* 1*-round bandwidth complexity of* MemList(K_3) *under edge insertions is* $O(\sqrt{n})$.

Proof. Let $N_u(r)$ be the set of neighbours of u on round r. Note that $N_u(r)$ can be encoded as an n-bit string, which indicates, for every node x, whether or not x is a neighbour of u on round r.

The algorithm is as follows. When a new edge uv is inserted on round r, both u and v send to all their neighbors the identity of their new neighbor, denoted by `NEWID`.

In addition, u sends a bitmask of $\lceil\sqrt{n}\rceil$ bits to v, indicating, for every one of the previous $\lceil\sqrt{n}\rceil$ rounds, whether or not one of u's neighbors has sent a `NEWID` to u. We denote this information by $\texttt{LAST}_u$. Node v sends to u the same information.

Finally, u encodes $N_u(r)$ as an n-bit string, denoted $\texttt{ALL}_u(r)$, and starts sending it to v in chunks of $\lceil\sqrt{n}\rceil$ bits per round. This process begins on round r and ends on round $r + \lfloor\sqrt{n}\rfloor$, when the entire string has been sent. During these rounds, u keeps sending `NEWID` to v, and to all its other neighbors, as described above. Node v does the same. It should be noted that this continuous communication between u and v is not intended for allowing them to detect triangles that appear by the insertion of the edge uv, as these are detected immediately due to previous information. Rather, communicating `ALL` allows u and v to detect triangles that appear by other topology changes that may occur in subsequent rounds, as we show below.

Overall, `NEWID` requires $O(\log n)$ bits, while `LAST` and `ALL` require $O(\sqrt{n})$ bits. Thus, the required bandwidth is $O(\sqrt{n})$.

We show that at the end of each round, every node u has enough information to determine, for every two of its neighbors v and w, whether or not the edge vw exists.

First, since only edge insertions are considered, if the edge vw exists in the initial graph, it exists throughout. Also, if vw is inserted when at least one of the edges uv or uw already exists, then u receives this information through `NEWID`. The only other case is when vw does not exist in the initial graph, and is inserted when u is not yet connected to either v or w. That is, the initial graph does not contain any of these three edges, and vw is inserted before the other two. W.l.o.g. assume that uv is inserted before uw. Now, let t_v be the round in which uv is inserted, and let t_w be the round in which uw is inserted.

If $t_w - t_v \leq \lceil\sqrt{n}\rceil$, then when uv is inserted, v sends a `NEWID` to w. Therefore, when uw is inserted, u can determine from $\texttt{LAST}_w$ that in round t_v a neighbor of w has sent a `NEWID` to w. Since the only edge inserted in that round is uv, u determines that v is a neighbor of w. If $t_w - t_v > \lceil\sqrt{n}\rceil$, then, in round t_w, v has already sent the entire string $\texttt{ALL}_v(t_v)$ to u, which indicates that w is a neighbor of v. Therefore, in all cases, u determines whether or not the edge vw exists, as claimed. ◀

The algorithm given for Theorem 2.1 can be extended to handle edge deletions and node deletions as well. Also, if we are promised a quiet round, the problem can be solved with $O(1)$ bits of bandwidth (see full version [5]). Node insertions are harder to handle than the other types of changes. If we are promised $r - 1$ quiet rounds, a simple algorithm exists that uses only $O\left(\frac{n}{r}\right)$ bits of bandwidth. As we show in Section 2.1.2, this is tight.

▶ **Theorem 2.2.** *For every r, the deterministic r-round bandwidth complexity of* MemList(K_3) *under any type of change is $O(n/r)$.*

Proof. The algorithm is as follows: on every round, every node u prepares an n-bit message that specifies its current list of neighbors. Then, it breaks this message into B-bit blocks, where $B = \lceil \frac{n}{r} \rceil$, and sends it to all its neighbors, one block on every round.

Additionally, on every round, u sends to all its neighbors one bit indicating whether or not its list of neighbors has changed on the current round. Whenever the list of neighbors changes, u builds its new list of neighbors, breaks it into blocks, and starts sending it again.

After at most $r-1$ quiet rounds, all nodes are guaranteed to finish sending their list to all their neighbors, thus every node can determine whether any pair of its neighbors are connected to each other or not, as required. ◀

2.1.2 Lower bounds

The 1-round bandwidth complexities for node and edge deletions are clearly tight. Next, we show that handling edge insertions in 1 round requires at least $\Omega(\sqrt{n})$ bits of bandwidth. By Theorem 2.1, this bound is tight. We first prove the following lemma, which shows that a sufficiently dense bipartite graph includes a large enough complete bipartite subgraph. This has the same spirit as the results of Erdös [15], used in [1, 18] for bounding the number of single-bit rounds for detecting triangles, but here the sides can be of different sizes.

▶ **Lemma 2.3.** *For every $\varepsilon \in (0,1)$ there exist $\alpha, \beta, \gamma \in (0,1)$, such that for every bipartite graph $G = (L \cup R, E)$ having at least $(1-\varepsilon)|L||R|$ edges, there are subsets $A \subseteq L$ and $B \subseteq R$, whose sizes are $|A| \geq \alpha \cdot |L|$ and $|B| \geq \beta \cdot \gamma^{|L|} \cdot |R|$, such that $uv \in E$ for every $u \in A$ and $v \in B$ (i.e., the induced subgraph on $A \cup B$ is a complete bipartite graph).*

Proof. Let $\alpha = \frac{1-\varepsilon}{6}$, and let $\mathcal{A}$ be the set of all subsets of L whose size is exactly $\lceil \alpha \cdot |L| \rceil$. For every $A \in \mathcal{A}$, we denote by N_A the set of all vertices in R which are connected to every vertex in A. Consider the sum $S = \sum_{A \in \mathcal{A}} |N_A|$. Let $M = \max\{|N_A| : A \in \mathcal{A}\}$. Then:

$$S \leq \sum_{A \in \mathcal{A}} M = \binom{|L|}{\lceil \alpha \cdot |L| \rceil} \cdot M \tag{1}$$

On the other hand, the sum S can be computed by counting, for every $v \in R$, the number of sets $A \in \mathcal{A}$ such that $v \in N_A$:

$$S = \sum_{A \in \mathcal{A}} |N_A| = \sum_{A \in \mathcal{A}} \sum_{v \in N_A} 1 = \sum_{v \in R} \sum_{\substack{A \in \mathcal{A}: \\ v \in N_A}} 1 = \sum_{v \in R} |\{A \in \mathcal{A} : v \in N_A\}|$$

For $p \in (0,1)$, let k_p be the number of vertices in R whose degree is at least $p \cdot |L|$. For every $v \in R$ whose degree is $d(v)$ we have $|\{A \in \mathcal{A} : v \in N_A\}| = \binom{d(v)}{\lceil \alpha \cdot |L| \rceil}$. Therefore we can bound the above sum

$$S = \sum_{v \in R} \binom{d(v)}{\lceil \alpha \cdot |L| \rceil} \geq \sum_{\substack{v \in R: \\ d(v) \geq p \cdot |L|}} \binom{d(v)}{\lceil \alpha \cdot |L| \rceil} \geq \sum_{\substack{v \in R: \\ d(v) \geq p \cdot |L|}} \binom{\lceil p \cdot |L| \rceil}{\lceil \alpha \cdot |L| \rceil} = k_p \cdot \binom{\lceil p \cdot |L| \rceil}{\lceil \alpha \cdot |L| \rceil}.$$

Combining this with (1) gives $\binom{|L|}{\lceil \alpha \cdot |L| \rceil} \cdot M \geq k_p \cdot \binom{\lceil p \cdot |L| \rceil}{\lceil \alpha \cdot |L| \rceil}$, which implies:

$$M \geq k_p \cdot \binom{\lceil p \cdot |L| \rceil}{\lceil \alpha \cdot |L| \rceil} \Big/ \binom{|L|}{\lceil \alpha \cdot |L| \rceil} \tag{2}$$

We can bound k_p as follows:

$$|E| = \sum_{v\in R} d(v) = \sum_{\substack{v\in R:\\ d(v)\geq p\cdot|L|}} d(v) + \sum_{\substack{v\in R:\\ d(v)<p\cdot|L|}} d(v) \leq \sum_{\substack{v\in R:\\ d(v)\geq p\cdot|L|}} |L| + \sum_{\substack{v\in R:\\ d(v)<p\cdot|L|}} p\cdot|L|$$
$$= k_p\cdot|L| + (|R|-k_p)\cdot p\cdot|L| = (1-p)\cdot k_p\cdot|L| + p\cdot|L|\cdot|R|$$

On the other hand we have $|E| \geq (1-\varepsilon)\cdot|L|\cdot|R|$, and therefore $(1-p)\cdot k_p\cdot|L| + p\cdot|L|\cdot|R| \geq (1-\varepsilon)\cdot|L|\cdot|R|$, which implies $k_p \geq \frac{1-\varepsilon-p}{1-p}\cdot|R|$. Setting $p=\frac{1-\varepsilon}{2}$ gives $k_p \geq \frac{1-\varepsilon}{1+\varepsilon}\cdot|R|$, and substituting this into (2) gives:

$$M \geq \frac{1-\varepsilon}{1+\varepsilon}\cdot\frac{\binom{\lceil p\cdot|L|\rceil}{\lceil\alpha\cdot|L|\rceil}}{\binom{|L|}{\lceil\alpha\cdot|L|\rceil}}\cdot|R|$$

Finally, we bound the binomial fraction on the right-hand size as follows. For simplicity, define $a=|L|, b=\lceil p\cdot|L|\rceil, c=\lceil\alpha\cdot|L|\rceil$. Note that $a\geq b\geq c$. Now:

$$\frac{\binom{b}{c}}{\binom{a}{c}} = \frac{\prod_{i=1}^{c}(i+b-c)}{\prod_{i=1}^{c}(i+a-c)} = \prod_{i=1}^{c}\frac{i+b-c}{i+a-c} = \prod_{i=1}^{c}(1-\frac{a-b}{i+a-c})$$
$$\geq \prod_{i=1}^{c}(1-\frac{a-b}{a-c}) = \prod_{i=1}^{c}\frac{b-c}{a-c} = \left(\frac{b-c}{a-c}\right)^c$$

Therefore:

$$\frac{\binom{\lceil p\cdot|L|\rceil}{\lceil\alpha\cdot|L|\rceil}}{\binom{|L|}{\lceil\alpha\cdot|L|\rceil}} \geq \left(\frac{\lceil p\cdot|L|\rceil - \lceil\alpha\cdot|L|\rceil}{|L|-\lceil\alpha\cdot|L|\rceil}\right)^{\lceil\alpha\cdot|L|\rceil} \tag{3}$$

In order to choose appropriate values for β and γ, we now distinguish between two cases for the graph G. If $|L|\leq\frac{1}{\alpha}$, then $\alpha|L|\leq 1$. Since there must be a vertex in $|L|$ having at least $(1-\varepsilon)|R|$ neighbors in R, the claim holds for every $\beta\leq 1$ and $\gamma\leq 1-\varepsilon$.

If $|L|>\frac{1}{\alpha}$, we can further develop the right hand side of (3) as follows:

$$\left(\frac{\lceil p|L|\rceil - \lceil\alpha|L|\rceil}{|L|-\lceil\alpha|L|\rceil}\right)^{\lceil\alpha|L|\rceil} \geq \left(\frac{p|L|-(\alpha|L|+1)}{|L|-\alpha|L|}\right)^{\alpha|L|+1} = \left(\frac{p-\alpha}{1-\alpha} - \frac{1}{(1-\alpha)|L|}\right)^{\alpha|L|+1}$$
$$> \left(\frac{p-\alpha}{1-\alpha} - \frac{1}{(1-\alpha)\frac{1}{\alpha}}\right)^{\alpha|L|+1} = \left(\frac{p-2\alpha}{1-\alpha}\right)^{\alpha|L|+1}$$
$$= \left(\frac{1-\varepsilon}{5+\varepsilon}\right)^{\alpha|L|+1}$$

To sum it all up, we now have $M > \left(\frac{1-\varepsilon}{5+\varepsilon}\right)^{\alpha\cdot|L|+1}$. Recalling the definition of M, this inequality implies that there exists $A\subseteq L$ whose size is exactly $\lceil\alpha\cdot|L|\rceil$, such that $|N_A| > \left(\frac{1-\varepsilon}{5+\varepsilon}\right)^{\alpha\cdot|L|+1}$, i.e., there are more than $\left(\frac{1-\varepsilon}{5+\varepsilon}\right)^{\alpha\cdot|L|+1}$ vertices in R which are connected to every vertex in A. Therefore, the claim holds for every β and γ such that: $\beta\leq\frac{1-\varepsilon}{5+\varepsilon}, \gamma\leq \left(\frac{1-\varepsilon}{5+\varepsilon}\right)^{\alpha} = \left(\frac{1-\varepsilon}{5+\varepsilon}\right)^{\frac{1-\varepsilon}{6}}$. Thus, given $\varepsilon\in(0,1)$, the following values of α,β,γ satisfy the claim for every G:

$$\alpha = \frac{1-\varepsilon}{6}, \beta = \frac{1-\varepsilon}{5+\varepsilon}, \gamma = \min\left\{1-\varepsilon, \left(\frac{1-\varepsilon}{5+\varepsilon}\right)^{\frac{1-\varepsilon}{6}}\right\}.$$

◀

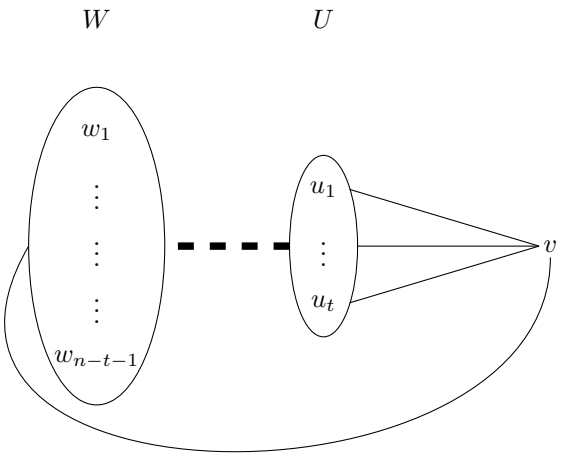

Figure 1 The lower bound sequence for MemList(K_3) with edge insertions. The connections between W and U are chosen. Then, v is connected to all of U. The single edge from v to W is added last.

▶ **Theorem 2.4.** *The randomized* 1*-round bandwidth complexity of* MemList(K_3) *under edge insertions is* $\Omega(\sqrt{n})$.

Proof. Let A be a (randomized) 1-round algorithm that solves MemList(K_3) under edge insertions using bandwidth B with error probability ε. Let t be a parameter to be defined later, and consider a tripartite graph with n nodes as in Figure 1. Let $\mathcal{C}$ be the set of all possible bipartite graphs with vertex sets W and U. Note that $|\mathcal{C}| = 2^{t(n-t-1)}$. For every $C \in \mathcal{C}$ and every $w \in W$, we define a sequence of changes $S_{C,w}$ as follows. We start with no edges. Then, we insert edges between U and W to get the bipartite graph C. During the next t rounds, we connect v to every $u \in U$, one by one. Finally, we insert the edge vw. In the end of $S_{C,w}$, after 1 additional round of communication, node v must output a list of all triangles containing v. By construction, this list is uniquely determined by the set of neighbors of w in U. Thus, node v needs to know the set of all neighbors of w after 1 additional round of communication.

We assume that the output of v is correct with probability at least $1 - \varepsilon$. Therefore, by Yao's lemma, there exists a deterministic algorithm A' that solves the same problem correctly for at least $1 - \varepsilon$ of all inputs. We define a bipartite graph with node sets $\mathcal{C}$ and W, such that the edge $w - C$ exists if and only if the output of v is correct for the sequence $S_{C,w}$.

We have $|W| = n - t - 1$ and $|\mathcal{C}| = 2^{t(n-t-1)}$. Also, by assumption, this graph contains at least $(1 - \varepsilon)$ of all possible edges. Therefore, by Lemma 2.3, there exists $W_1 \subseteq W$ of size at least $\alpha \cdot (n - t - 1)$ and $\mathcal{C}_1 \subseteq \mathcal{C}$ of size at least $\beta \cdot \gamma^{n-t-1} \cdot 2^{t(n-t-1)}$, for some $\alpha, \beta, \gamma \in (0, 1)$ that depend only on ε, such that the output of v is correct for the sequence $S_{C,w}$ for every $C \in \mathcal{C}_1$ and every $w \in W_1$.

Now, consider the input of node v during any sequence $S_{C,w}$. During the first stage of building the bipartite graph C, v is isolated and receives no input. Then, it receives a set of messages from the nodes in U, and finally one additional message from w. Note that the messages v receives from the nodes in U depend only on C, and not on w, since the nodes in U cannot know the identity of w until the final round of communication.

Now, on every round, every node in U can send to v any of 2^B possible messages. Altogether, during the entire sequence, the nodes of U send to v a set of $\frac{t(t+3)}{2}$ messages. Hence, the number of possible inputs from the nodes of U to v is $2^{B\cdot\frac{t(t+3)}{2}}$. Therefore, there exists $\mathcal{C}_2 \subseteq \mathcal{C}_1$ of size at least $2^{-B\cdot\frac{t(t+3)}{2}} \cdot |\mathcal{C}_1|$, such that v receives the same input from all nodes in U for every sequence $S_{C,w}$ for $C \in \mathcal{C}_2$ and every $w \in W_1$. Thus, we have:

$$|\mathcal{C}_2| \geq 2^{-B\frac{t(t+3)}{2}}|\mathcal{C}_1| \geq 2^{-B\frac{t(t+3)}{2}}\beta\gamma^{n-t-1}2^{t(n-t-1)} = \beta\gamma^{n-t-1}2^{t(n-t-1)-B\frac{t(t+3)}{2}} \tag{4}$$

On the other hand, we can bound the size of $\mathcal{C}_2$ by considering the number of possible neighbors of w in any $C \in \mathcal{C}_2$, for every $w \in W$. Recall that during the sequence $S_{C,w}$, v receives only one message from w. Also, for every $w \in W_1$, v must determine the set of all neighbors of w in U. Since there are only 2^B possible inputs v can receive from w, every $w \in W_1$ can have at most 2^B possible sets of neighbors in any $C \in \mathcal{C}_2$.

Every $w \in W \setminus W_1$ can have any subset of U as its set of neighbors, hence it can have at most 2^t possible sets of neighbors. Now, since every $C \in \mathcal{C}_2$ is uniquely determined by set of neighbors of every $w \in W$, we have:

$$|\mathcal{C}_2| \leq \left(\prod_{w\in W_1} 2^B\right)\left(\prod_{w\in W\setminus W_1} 2^t\right) = 2^{B\cdot|W_1|}\cdot 2^{t\cdot(|W|-|W_1|)} = 2^{(B-t)\cdot|W_1|+t(n-t-1)} \tag{5}$$

Combining (4) and (5) gives: $\beta\cdot\gamma^{n-t-1}\cdot 2^{t(n-t-1)-B\cdot\frac{t(t+3)}{2}} \leq 2^{(B-t)\cdot|W_1|+t(n-t-1)}$, and setting $t = \lceil\sqrt{n}\rceil$ gives, with some algebraic manipulations, $B \geq \Omega(\sqrt{n})$. ◀

Finally, for node insertions, we show that every r-round algorithm must use at least $\Omega(n/r)$ bits of bandwidth, which is tight by Theorem 2.2.

▶ **Theorem 2.5.** *For every r, the randomized r-round bandwidth complexity of $\mathsf{MemList}(K_3)$ under node insertions is $\Omega\left(\frac{n}{r}\right)$.*

Proof. Let A be a (randomized) r-round algorithm that solves $\mathsf{MemList}(K_3)$ under node insertions using bandwidth B with error probability ε. We show that $r \cdot B = \Omega(n)$.

Let u be any node, and let $\mathcal{C}$ be the set of all possible graphs on the other $n-1$ nodes. For every $C \in \mathcal{C}$ we define the sequence S_C as follows:

- Start with an empty graph (no nodes and no edges).
- During $n-1$ rounds, insert one node on each round and connect it to the nodes which have been already inserted, according to the edges in C. After $n-1$ rounds, the graph is identical to the graph C.
- On round n insert u and connect to all the other nodes.

After $r-1$ quiet rounds u needs to output the list of all triangles that contain u. Since u is connected to all the other nodes, this implies that u needs to know the graph C. For every $C \in \mathcal{C}$, the output of u is guaranteed to be correct for the sequence S_C with probability at least $(1-\varepsilon)$. Therefore, by Yao's lemma, there exists a deterministic algorithm, A', that guarantees that the output of u is correct for at least $(1-\varepsilon)$ of all sequences. That is, there exists a subset $\mathcal{C}_1 \subseteq \mathcal{C}$, whose size is at least $(1-\varepsilon)\cdot|\mathcal{C}|$, such that A' guarantees the correct output of u for sequences S_C for all $C \in \mathcal{C}_1$.

Now, the number of possible graphs on $n-1$ nodes is $2^{\binom{n-1}{2}}$, hence the size of $\mathcal{C}_1$ is at least $(1-\varepsilon)\cdot 2^{\binom{n-1}{2}}$. Since A' is deterministic, and u needs to distinguish correctly between all possible $C \in \mathcal{C}_1$, this implies that the input u receives from its neighbors must have at

least $(1-\varepsilon)\cdot 2^{\binom{n-1}{2}}$ possible values. Every neighbor of u can send any of 2^B messages on every round, thus the number of possible inputs u can receive on a single round is $2^{B\cdot(n-1)}$. Therefore, during r rounds of communication, the number of possible inputs to u is $2^{r\cdot B\cdot(n-1)}$.

Combining the above we get that $2^{r\cdot B\cdot(n-1)} \geq (1-\varepsilon)\cdot 2^{\binom{n-1}{2}}$, which can be simplified to $r\cdot B \geq \frac{n}{2} + \frac{\log(1-\varepsilon)}{n-1}$, and it follows that $r\cdot B = \Omega(n)$. ◀

2.2 Membership detection

Table 3 summarizes the results of this subsection.

Table 3 Bandwidth complexities of MemDetect(K_3).

	Node deletions	Edge deletions	Edge insertions	Node insertions
$r=1$	0	$\Theta(1)$	$O(\log n)$	$\Theta(n)$
$r\geq 2$	0	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$

2.2.1 Upper bounds

The upper bounds for node deletions and edge deletions follow from Observation 1. The following shows that edge insertions can be handled with $O(\log n)$ bits of bandwidth, which can be extended to handle node/edge deletions as well (see full version [5]).

▶ **Theorem 2.6.** *The deterministic* 1*-round bandwidth complexity of* MemDetect(K_3) *under edge insertions is* $O(\log n)$.

Proof. The algorithm works as follows. On every round, every node sends to all its neighbors an indication whether or not it got a new neighbor on the current round, along with the ID of the new neighbor (if any). We denote this information by `NEWID`. Note that with just this information, for every pair of adjacent edges $u-v-w$, at least one of u and w know that these two edges exist (specifically, the first one connected to v knows that the other one is also connected to v).

Additionally, every node u sends to every neighbor v one bit, indicating whether u knows that v is part of some triangle. We denote this bit by `ACCEPT`.

Suppose the edge uv is inserted and creates at least one triangle uvw. As explained above, in this case at least one of u and v knows that the triangle exists, therefore it sends `ACCEPT` $= 1$ to the two other nodes. Thus all three nodes know that they are part of a triangle. It follows that on each round all nodes can determine whether they are in a triangle. ◀

If we only consider sequences of graphs with bounded degree Δ, and only allow edge insertions, the complexity of Theorem 2.6 can be improved to $O(\sqrt{\Delta\log n})$, using an algorithm similar to that of Theorem 2.1, but with $O(\sqrt{\Delta\log n})$ bits instead of $O(\sqrt{n})$, as follows. On every round, every node sends to all its neighbors an indication whether or not it has a new neighbor. We denote this information by `NEW`. Additionally, whenever a new edge uv is inserted, the following happens:

- u sends to v an indication whether or not it knows about a triangle that contains v.
- For every round i within the last $\left\lceil\sqrt{\Delta\log n}\right\rceil$ rounds, u sends to v an indication whether or not it has had a new neighbor on round i.
- For every round i within the last $\left\lceil\sqrt{\Delta\log n}\right\rceil$ rounds, u sends to v an indication whether or not any of its neighbors has sent `NEW` $= 1$ on round i.

- u computes the list of IDs of all its current neighbors, and starts sending it to v, $\left\lceil \sqrt{\Delta \log n} \right\rceil$ bits on every round. Since u has at most Δ neighbors, after $O(\sqrt{\Delta \log n})$ rounds v has the complete list. Note that by the time this process completes, the list may not be up-to-date.

All of this requires a $O(\sqrt{\Delta \log n})$ bandwidth, and it can be shown, similarly to Theorem 2.1, that this allows every node to give the correct output in all cases. The discussion of what happens when a quiet round is promised, is deferred to the full version [5].

2.2.2 Lower bounds

For node deletions and edge deletions, we have trivial constant lower bounds, which are tight, as shown above. For edge insertions, we have shown a general algorithm that uses $O(\log n)$ bits, and also an algorithm that uses $O(\sqrt{\Delta \log n})$ bits for graphs with bounded degree Δ. The latter algorithm implies that showing a lower bound of B bits for the bandwidth complexity of this problem would require looking at sequences of graphs with degree at least $\Omega(\frac{B^2}{\log n})$. In particular, in order to show that the algorithm of Theorem 2.6 is optimal, one has to consider sequences of graphs with degree at least $\log n$. Other than that, it remains an open question whether or not the algorithm of Theorem 2.6 can be improved.

▶ **Open Question 1.** *What is the* 1*-round bandwidth complexity of* MemDetect(K_3) *under edge insertions?*

Finally, for node insertions, the following theorem shows a lower bound of $\Omega(n)$ bits.

▶ **Theorem 2.7.** *The randomized* 1*-round bandwidth complexity of* MemDetect(K_3) *under node insertions is* $\Omega(n)$.

Proof. Let A be a (randomized) 1-round algorithm that solves MemDetect(K_3) under node insertions using bandwidth B with error probability ε. Fix $x \in V$, let $U = V \setminus \{x\}$, and let $\mathcal{C}$ be the set of all possible graphs on the nodes of U. For every $C \in \mathcal{C}$ and every $u, v \in U$, define the sequence $S_{C,u,v}$ as follows:

- Start with an empty graph (no nodes and no edges).
- During $n-1$ rounds, insert one node of U on each round and connect it to the nodes which have been already inserted, according to the edges in C. After $n-1$ rounds, the graph is identical to the graph C.
- On round n insert x and connect it to u and v.

Then, x should output 1 iff the edge uv exists in C. By our assumption, for every C and every $u, v \in U$, the output is correct with probability at least $1 - \varepsilon$. Note that during the final round, x receives only one message from each of u and v. Also, during this final round, u and v do not know the identity of the other neighbor of x. Hence, the message received from u depends only on the identity of u and the graph C, and not on the identity of v, and the message received from v depends only on the identity of v and the graph C.

Now, consider the following experiment for a given graph $C \in \mathcal{C}$. First, we run the sequence $S_{C,u,v}$ for some $u, v \in U$, and stop just before the last round, in which x is connected to u and v. Then, every node $w \in U$ generates a message to be sent to x, as if it has been connected to x on the final round. For every $w \in U$, let m_w be the message generated by w (note that m_w is a random variable).

Note again, that the messages generated by the nodes of U depend only on the graph C, and not on the other nodes that may have been connected to x on the last round. Therefore, for every $u, v \in U$, given the messages generated by u and v, x should be able to determine, with probability at least $1 - \varepsilon$, whether or not the edge uv exists in C.

Next, for every pair of nodes $u, v \in U$, let I_{uv} be the output of x given the two messages m_u and m_v. Note that for nodes u, v, w, the variables I_{uv} and I_{uw} are not necessarily independent. Now, let C' be the graph on the nodes of U, in which the edge uv exists if and only if $I_{uv} = 1$. We consider C' to be the result of the experiment.

Let p_C denote the probability that at the end of the experiment we have $C' = C$. Since, for every $u, v \in U$, the value of I_{uv} corresponds to the edge uv in the graph C with probability at least $1 - \varepsilon$, we have $p_C \geq (1-\varepsilon)^{\binom{n-1}{2}}$. Summing the above for every $C \in \mathcal{C}$ we get:

$$\sum_C p_C \geq |\mathcal{C}| \cdot (1-\varepsilon)^{\binom{n-1}{2}} = (2-2\varepsilon)^{\binom{n-1}{2}} \tag{6}$$

On the other hand, consider the set of messages generated by the nodes of U at the end of the first stage of the experiment. For every possible set of messages M generated by the nodes of U during the first stage of the experiment, denote by $\phi_C(M)$ the probability for generating exactly the messages of M. Also, for every $C' \in \mathcal{C}$, let $\Psi_M(C')$ denote the probability for the result of the experiment to be equal to C', given the set of generated messages M. We have:

$$\sum_C p_C = \sum_C \sum_M \phi_C(M) \cdot \Psi_M(C) \leq \sum_C \sum_M \Psi_M(C) = \sum_M \sum_C \Psi_M(C) = \sum_M 1 = |\mathcal{M}|$$

where $\mathcal{M}$ is the set of all possible values of M. Since every message has exactly 2^B possible values, the number of possible sets of $(n-1)$ messages is $|\mathcal{M}| = 2^{B(n-1)}$, and hence $\sum_C p_C \leq 2^{B(n-1)}$. Combining this with (6) gives $2^{B(n-1)} \geq (2-2\varepsilon)^{\binom{n-1}{2}}$. After some simplifications we get the desired bound:

$$B \geq \log(2-2\varepsilon) \cdot \frac{n-2}{2} = \Omega(n)$$ ◀

▶ Remark 2.8. The discussion of $\mathsf{List}(K_3)$ and $\mathsf{Detect}(K_3)$, as well as the treatment of s-cliques, appear in the full version [5].

References

1. Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Christoph Lenzen. Fooling Views: A New Lower Bound Technique for Distributed Computations under Congestion. *CoRR*, abs/1711.01623, 2017. arXiv:1711.01623.
2. Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 815–826, 2018. doi:10.1145/3188745.3188922.
3. Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully Dynamic Maximal Independent Set with Sublinear in n Update Time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1919–1936, 2019. doi:10.1137/1.9781611975482.116.
4. Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Local Distributed Algorithms in Highly Dynamic Networks. *CoRR*, abs/1802.10199, 2018. arXiv:1802.10199.
5. Matthias Bonne and Keren Censor-Hillel. Distributed Detection of Cliques in Dynamic Networks. *CoRR*, abs/1904.11440, 2019. arXiv:1904.11440.
6. Zvika Brakerski and Boaz Patt-Shamir. Distributed discovery of large near-cliques. *Distributed Computing*, 24(2):79–89, 2011. doi:10.1007/s00446-011-0132-x.
7. Keren Censor-Hillel, Neta Dafni, Victor I. Kolobov, Ami Paz, and Gregory Schwartzman. Fast and Simple Deterministic Algorithms for Highly-Dynamic Networks. *CoRR*, abs/1901.04008, 2019.

8 Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. *Distributed Computing*, 32(1):41–57, 2019. doi:10.1007/s00446-018-0324-8.

9 Keren Censor-Hillel, Elad Haramaty, and Zohar S. Karnin. Optimal Dynamic Distributed MIS. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 217–226, 2016. doi:10.1145/2933057.2933083.

10 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. doi:10.1137/1.9781611975482.51.

11 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. doi:10.4230/LIPIcs.DISC.2018.16.

12 Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

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

14 Yuhao Du and Hengjie Zhang. Improved Algorithms for Fully Dynamic Maximal Independent Set. *CoRR*, abs/1804.08908, 2018. arXiv:1804.08908.

15 P. Erdös. On extremal problems of graphs and generalized graphs. *Israel Journal of Mathematics*, 2(3):183–190, September 1964. doi:10.1007/BF02759942.

16 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, October 16-20, 2017, Vienna, Austria*, pages 15:1–15:30, 2017. doi:10.4230/LIPIcs.DISC.2017.15.

17 Guy Even, Reut Levi, and Moti Medina. Faster and Simpler Distributed Algorithms for Testing and Correcting Graph Properties in the CONGEST-Model. *CoRR*, abs/1705.04898, 2017. arXiv:1705.04898.

18 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. doi:10.1145/3210377.3210401.

19 Orr Fischer, Tzlil Gonen, and Rotem Oshman. Distributed Property Testing for Subgraph-Freeness Revisited. *CoRR*, abs/1705.04033, 2017. arXiv:1705.04033.

20 Pierre Fraigniaud, Pedro Montealegre, Dennis Olivetti, Ivan Rapaport, and Ioan Todinca. Distributed Subgraph Detection. *CoRR*, abs/1706.03996, 2017. arXiv:1706.03996.

21 Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed Testing of Excluded Subgraphs. In *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, pages 342–356, 2016. doi:10.1007/978-3-662-53426-7_25.

22 Tzlil Gonen and Rotem Oshman. Lower Bounds for Subgraph Detection in the CONGEST Model. In *21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, December 18-20, 2017*, pages 6:1–6:16, 2017. doi:10.4230/LIPIcs.OPODIS.2017.6.

23 Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for Maximal Independent Set and other problems. *CoRR*, abs/1804.01823, 2018. arXiv:1804.01823.

24 Juho Hirvonen, Joel Rybicki, Stefan Schmid, and Jukka Suomela. Large Cuts with Local Algorithms on Triangle-Free Graphs. *Electr. J. Comb.*, 24(4):P4.21, 2017. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v24i4p21.

25 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 2017, Washington, DC, USA, July 25-27, 2017*, pages 381–389, 2017. doi:10.1145/3087801.3087811.

26 Michael König and Roger Wattenhofer. On Local Fixing. In *OPODIS*, volume 8304 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2013.

27 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. doi:10.4230/LIPIcs.OPODIS.2017.4.

28 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the Distributed Complexity of Large-Scale Graph Computations. In *SPAA*, pages 405–414. ACM, 2018.

29 Merav Parter, David Peleg, and Shay Solomon. Local-on-Average Distributed Tasks. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 220–239, 2016. doi:10.1137/1.9781611974331.ch17.

30 Seth Pettie and Hsin-Hao Su. Distributed coloring algorithms for triangle-free graphs. *Inf. Comput.*, 243:263–280, 2015. doi:10.1016/j.ic.2014.12.018.

31 Shay Solomon. Fully Dynamic Maximal Matching in Constant Update Time. In *Proceedings of the IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 325–334, 2016. doi:10.1109/FOCS.2016.43.