“Tri, Tri again”: Finding Triangles and Small Subgraphs in a Distributed Setting [Extended Abstract]

Danny Dolev[1], Christoph Lenzen[2], and Shir Peled[1]

[1] School of Engineering and Computer Science
Hebrew University of Jerusalem, Israel
{dolev,shir.peled}@cs.huji.ac.il
[2] Department for Computer Science and Applied Mathematics
Weizmann Institute of Science, Israel
clenzen@cs.huji.ac.il

Abstract. Let $G = (V, E)$ be an n-vertex graph and M_d a d-vertex graph, for some constant d. Is M_d a subgraph of G? We consider this problem in a model where all n processes are connected to all other processes, and each message contains up to $\mathcal{O}(\log n)$ bits. A simple deterministic algorithm that requires $\mathcal{O}(n^{(d-2)/d}/\log n)$ communication rounds is presented. For the special case that M_d is a triangle, we present a probabilistic algorithm that requires an expected $\mathcal{O}(n^{1/3}/(t^{2/3}+1))$ rounds of communication, where t is the number of triangles in the graph, and $\mathcal{O}(\min\{n^{1/3}\log^{2/3} n/(t^{2/3}+1), n^{1/3}\})$ with high probability.
We also present deterministic algorithms that are specially suited for sparse graphs. In graphs of maximum degree Δ, we can test for arbitrary subgraphs of diameter D in $\mathcal{O}(\Delta^{D+1}/n)$ rounds. For triangles, we devise an algorithm featuring a round complexity of $\mathcal{O}((A^2 \log_{2+n/A^2} n)/n)$, where A denotes the arboricity of G.

1 Introduction

In distributed computing, it is common to represent a distributed system as a graph whose nodes are computational devices (or, more generally, any kind of agents) and whose edges indicate which pairs of devices can directly communicate with each other. Since its infancy, the area has been arduously studying the so-called LOCAL model (cf. [17]), where the devices try to jointly compute some combinatorial structure, such as a maximal matching or a node coloring, of this communication graph. In its most pure form, the local model is concerned with one parameter only: the locality of a problem, i.e., the number of hops up to which nodes need to learn the topology and local portions of the input in order to compute their local parts of the output—for example this could be whether or not an outgoing edge is in the maximal matching or the color of the node.

Considerable efforts have been made to understand the effect of bounding the amount of communication across each edge. In particular, the CONGEST model that demands that in each time unit, at most $\mathcal{O}(\log n)$ bits are exchanged over each edge, has been studied intensively. However, to the best of our knowledge, all known lower bounds rely on “bottlenecks” [10,12,18], i.e., small edge cuts that severely constrain the total number of bits that may be communicated between different parts of the graph. In contrast, very little is known about the possibilities and limitations in case the communication graph is a clique, i.e., the communication bounds are symmetric and *independent* of the structure of the problem we need to solve. The few existing works show that, as one can expect, such a distributed system model is very powerful: A minimum spanning tree can be found in $\mathcal{O}(\log\log n)$ time [13], with randomization nodes can send and receive up to $\mathcal{O}(n)$ messages of size $\mathcal{O}(\log n)$ in $\mathcal{O}(1)$ rounds, without any initial knowledge of which nodes hold messages for which destinations [11], and, using the latter routine, they can sort n^2 keys in $\mathcal{O}(1)$ rounds (where each node holds n keys and needs to learn their index in the sorted sequence) [16]. In general, none of these tasks can be performed fast in the local model, as the communication graph might have a large diameter.

In the current paper, we examine a question that appears to be hard even in a clique if message size is constrained to be $\mathcal{O}(\log n)$. Given that each node initially knows its neighborhood in an input graph, the goal is to decide whether this graph contains some subgraph on $d \in \mathcal{O}(1)$ vertices. In the local model, this can be trivially solved by each node learning the topology up to a constant distance;[3] in our setting, this simple strategy might result in a running time of $\Omega(n/\log n)$, as some (or all) nodes may have to learn about the entire graph and thus need to receive $\Omega(n^2)$ bits. We devise a number of algorithms that achieve much better running times. These algorithms illustrate that efficient algorithms in the contemplated model need to strive for balancing the communication load, and we show some basic strategies to do so. We will see as a corollary that it is possible for all nodes to learn about the entire graph within $\mathcal{O}(|E|/n)$ rounds and therefore locally solve any (computable) problem on the graph; this refines the immediately obvious statement that the same can be accomplished within Δ (where Δ denotes the maximum degree of G) rounds by each node sending its complete list of neighbors to all other nodes. For various settings, we achieve running times of $o(|E|/n)$ by truly distributed algorithms that do not require that (some) nodes obtain full information on the entire input.

Apart from shedding more light on the power of the considered model, the detection of small subgraphs, sometimes referred to as *graphlets* or *network motifs*, is of interest in its own right. Recently, this topic received growing attention due to the importance of recurring patterns in man-made networks as well as natural ones. Certain subgraphs were found to be associated with neurobiological networks, others with biochemical ones, and others still with human-engineered networks [15]. Detecting network motifs is an important part of understanding biological networks, for instance, as they play a key role in information processing mechanisms of biological regulation networks. Even motifs as simple as

[3] In the local model, one is satisfied with at least one node detecting a respective subgraph. Requiring that the output is known by all nodes results in the diameter being a trivial lower bound for any meaningful problem.

triangles are of interest to the biological research community as they appear in gene regulation networks, where what a graph theorist would call a directed triangle is often referred to as a Feed-Forward Loop. In recent years, the network motifs approach to studying networks lead to development of dedicated algorithms and software tools. Being of highly applicative nature, algorithms used in such context are usually researched from an experimental point of view, using naturally generated data sets [9].

Triangles and triangle-free graphs also play a central role in combinatorics. For example, it is long since known that planar triangle-free graphs are 3-colorable [8]. The implications of triangle finding and triangle-freeness motivated extensive research of algorithms, as well as lower bounds, in the centralized model. Most of the work done on these problems falls into one of two categories: subgraph listing and property testing. In subgraph listing, the aim is to list all copies of a given subgraph. The number of copies in the graph, that may be as high as $\Theta\left(n^3\right)$ for triangles, sets an obvious lower bound for the running time of such algorithms, rendering instances with many triangles harder in some sense [4]. Property testing algorithms, on the other hand, distinguish with some probability between graphs that are triangle-free and graphs that are far from being triangle-free, in the sense that a constant fraction of the edges has to be removed in order for the graph to become triangle-free [1,2]. Although soundly motivated by stability arguments, the notion of measuring the distance from triangle-freeness by the minimal number of edges that need to be removed seems less natural than counting the number of triangles in the graph. Consider for instance the case of a graph with n nodes comprised of $n-2$ triangles, all sharing the same edge. From the property testing point of view, this graph is very close to being triangle free, although it contains a linear number of triangles. Some query-based algorithms were suggested in the centralized model, where the parameter to determine is the number of triangles in the graph. The lower bounds for such algorithms assume restrictions on the type of queries[4] that cannot be justified in our model [7].

Detailed Contributions. In Section 3, we start out by giving a family of deterministic algorithms that decide whether the graph contains a d-vertex subgraph within $\mathcal{O}(n^{(d-2)/d})$ rounds. In fact, these algorithms find *all* copies of this subgraph and therefore could be used to count the exact number of occurrences. They split the task among the nodes such that each node is responsible for checking an equal number of subsets of d vertices for being the vertices of a copy of the targeted subgraph. This partition of the problem is chosen independently of the structure of the graph. Note that even the trivial algorithm that lets each node collect its D-hop neighborhood and test it for instances of the subgraph in question does not satisfy this property. Still it exhibits a structure that is simple enough to permit a deterministic implementation of running time $\mathcal{O}(\Delta^{D+1}/n)$, where Δ is the maximum degree of the graph, given in Section 4. For the special case of triangles, we present a more intricate way of checking neighborhoods that

[4] For instance, in [7] the query model requires that edges are sampled uniformly at random.

results in a running time of $\mathcal{O}(A^2/n+\log_{2+n/A^2} n) \subseteq \mathcal{O}(|E|/n+\log n)$, where the arboricity A of the graph denotes the minimal number of forests into which the edge set can be decomposed. While always $A \leq \Delta$, it is possible that $A \in \mathcal{O}(1)$, yet $\Delta \in \Theta(n)$ (e.g. in a graph that is a star). Moreover, any family of graphs excluding a fixed minor has $A \in \mathcal{O}(1)$ [5], demonstrating that the arboricity is a much less restrictive parameter than Δ. Note also that the running time bound in terms of $|E|$ is considerably weaker than the one in terms of A; it serves to demonstrate that in the worst case, the algorithm's running time essentially does not deteriorate beyond the trivial $\mathcal{O}(|E|/n)$ bound.

All our deterministic algorithms systematically check for subgraphs by either considering all possible combinations of d nodes or following the edges of the graph. If there are *many* copies of the subgraph, it can be more efficient to randomly inspect small portions of the graph. In Section 5, we present a triangle-finding algorithm that does just that, yielding that for every $\varepsilon \geq 1/n^{\mathcal{O}(1)}$ and graph containing $t \geq 1$ triangles, one will be found with probability at least $1-\varepsilon$ within $\mathcal{O}((n^{1/3}\log^{2/3}\varepsilon^{-1})/t^{2/3}+\log n)$ rounds; we show this analysis to be tight.

All our algorithms are uniform, i.e., they require no prior knowledge of parameters such as t or A. Interleaving them will result in an asymptotic running time that is bounded by the minimum of all the individual results. Due to lack of space, we only sketch most proof ideas; for details we refer to [6].

2 Model and Problem

Our model separates the computational problem from the communication model. Let $V = \{1, \ldots, n\}$ represent the nodes of a distributed system. With respect to communication, we adhere to the synchronous CONGEST model as described in [17] on the complete graph on the node set V, i.e., in each computational round, each node may send (potentially different) $\mathcal{O}(\log n)$ bits to each other node. We do not consider the amount of computation performed by each node, however, for all our algorithms it will be polynomially bounded. Instead, we measure complexity in the number of rounds until an algorithm terminates.[5] Let $G = (V, E)$ be an arbitrary graph on the same vertex set, representing the computational problem at hand. Initially, every node $i \in V$ has the list $\mathcal{N}_i := \{j \in V \mid \{i, j\} \in E\}$ of its neighbors in G, but no further knowledge of G.

The computational problem we are going to consider throughout this paper is the following: Given a graph M_d on $d \in \mathcal{O}(1)$ vertices, we wish to discover whether M_d is a subgraph of G.

3 Deterministic Algorithms for General Graphs

During our exposition, we will discuss the issues of what to communicate and how to communicate it separately. That is, given sets of $\mathcal{O}(\log n)$-sized messages at all nodes satisfying certain properties, we provide subroutines that deliver

[5] Note that ensuring termination in the same round is easy due to the full connectivity.

all messages quickly and then use these subroutines in our algorithms. We start out by giving a very efficient deterministic scheme provided that origins and destinations of all messages are initially known to *all* nodes. We then will show that this scheme can be utilized to find all triangles or other constant-sized subgraphs in sublinear time.

Full-knowledge message passing. For a certain limited family of algorithms whose communication structure is basically independent of the problem graph, it is possible to exploit the full capacity of the communication system, i.e., provided that no node sends or receives more than n messages, all messages can be delivered in two rounds.

Lemma 1. *Given a bulk of messages, such that:*

1. *The source and destination of each message is known in advance to all nodes, and each source knows the contents of the messages to sent.*
2. *No node is the source of more than n messages.*
3. *No node is the destination of more than n messages.*

A routing scheme to deliver all messages within 2 rounds can be found efficiently.

To get some intuition on why Lemma 1 is true, observe that if every node initially holds a single message for every other node, the task can clearly be completed within a single round. This reduces our problem to finding a message passing scheme such that, after one round, every node will hold a single message for every other node. Now consider the bipartite multi-graph in which the vertices on the left are the nodes in their role as sources, the vertices on the right are the nodes in there role as destinations, and for each message whose source and destination are nodes i and j, respectively, there is an edge from i (on the left hand side) to j (on the right hand sight). Our desired scheme translates to finding n perfect matchings in the graph, since we could have every source send the edge used by it (i.e. the message) in the i^{th} matching to the ith node, resulting in every node holding a single message for every other node in the subsequent round. The graph described is always a disjoint union of n perfect matchings, as a corollary of Hall's Theorem. The required matchings can be found before communication commences, since all sources and destinations are known.

TriPartition - finding triangles deterministically. We now present an algorithm that finds whether there are triangles in G that has a better round complexity than $\mathcal{O}(|E|/n)$ if $|E|$ is large. Apart from a possible final broadcast message informing other nodes of a discovered triangle, it exhibits the very simple communication structure required by Lemma 1.

Let $S \subseteq 2^V$ be a partition of V into equally sized subsets of cardinality $n^{2/3}$. We write $S = \{S_1, ..., S_{n^{1/3}}\}$. To each node $i \in V$ we assign a distinct triplet from S denoted $S_{i,1}, S_{i,2}, S_{i,3}$ (where repetitions are admitted). Clearly, for any subset of three nodes there is a triplet such that each node is element of one

Algorithm 1: TriPartition at node i.

```
1 $E_i := \emptyset$
2 for $1 \leq j < k \leq 3$ do
3     for $l \in S_{i,j}$ do
4         retrieve $\mathcal{N}_l \cap S_{i,k}$
5         for $m \in \mathcal{N}_l \cap S_{i,k}$ do $E_i := E_i \cup \{l, m\}$
6 if there exists a triangle in $G_i := (V, E_i)$ then send "triangle" to all nodes
7 if received "triangle" from some node then return true
8 else return false
```

of the subsets in the triplet. Hence, for each triangle $\{t_1, t_2, t_3\}$ in G, there is some node i such that $t_1 \in S_{i,1}$, $t_2 \in S_{i,2}$, and $t_3 \in S_{i,3}$. Each node checks for triangles that are contained in its triplet of subsets by executing *TriPartition* (see Algorithm 1).

Theorem 1. TriPartition *correctly decides whether there exists a triangle in* G *and can be implemented within* $\mathcal{O}(n^{1/3})$ *rounds.*

Remark 1. The fact that (except for the potential final broadcast) the entire communication pattern of *TriPartition* is predefined enables to refrain from including any node identifiers into the messages. That is, instead of encoding the respective sublist of neighbors by listing their identifiers, nodes just send a $0-1$ array of bits indicating whether a node from the respective set from S is or is not a neighbor in G. The receiving node can decode the message because it is already known in advance which bit stands for which pair of nodes. We may hence improve the round complexity of *TriPartition* to $\mathcal{O}(n^{1/3}/\log n)$.

Generalization for d-cliques. *TriPartition* generalizes easily to an algorithm we call *dClique0* that finds d-cliques (as well as any other subgraph on d vertices). We choose S to be a partition of V into equal size subsets of cardinality $n^{(d-1)/d}$, resulting in $S = \{S_1, ..., S_{n^{1/d}}\}$. Each node now examines the edges between all pairs of some d-sized multisubset of S (as we did for $d = 3$ in *TriPartition*). Since there are exactly $|S|^d = n$ such multisets, all possible d-cliques are examined. Every node needs to receive the list of edges for all $\binom{d}{2}$ pairs, each containing at most $(n^{(d-1)/d})^2$ edges, thus every node needs to send and receive at most $\mathcal{O}(n^{(2d-2)/d})$ messages.

Theorem 2. *dClique0 determines correctly whether there exists a d-clique (or any given d-vertex graph) in* G *within* $\mathcal{O}(n^{(d-2)/d}/\log n)$ *rounds.*

4 Finding triangles in sparse graphs

In graphs that have $o(n^2)$ edges, one might hope to obtain faster algorithms. However, the algorithms from the previous section have congestion at the node

Algorithm 2: TriNeighbors at node i.

```
1 $E_i := \emptyset$
2 for $j \in V$ s.t. $(i,j) \in E$ do
3     retrieve $\mathcal{N}_j$
4     for $k \in \mathcal{N}_j$ do $E_i := E_i \cup \{j,k\}$
5 if there exists a triangle in $G_i := (V, E_i)$ then send "triangle" to all nodes
6 if received "triangle" from some node then return true
7 else return false
```

level, i.e., even if there are few edges in total, some nodes may still have to send or receive lots of messages. Hence, we need different strategies for sparse graphs. In this section, we derive bounds depending on parameters that reflect the sparsity of graphs.

Bounded degree. We start with a simple value, the *maximum degree* $\Delta := \max_{i \in V} \delta_i$, where the *degree of node* i $\delta_i := |\mathcal{N}_i|$. If Δ is relatively small, then every node can simply exchange its neighbors list with all its neighbors. We refer to this as *TriNeighbors* algorithm, whose pseudo-code is given in Algorithm 2. In a graph with bounded Δ, it will be much faster than *dClique0* algorithm.

Since potentially all vertices may have degree Δ, the message complexity per node is in $\mathcal{O}(\Delta^2 + n)$. We use an elegant message-passing technique, suggested to us by Shiri Chechik [3]. Assuming that (i) no node is the source of more than n messages in total, (ii) no node is the destination of more than n messages, and (iii) every node sends the exact same messages to all of the destinations for its messages, it delivers all messages in 3 rounds. This is done by first having each node distribute its messages evenly, in a Round-Robin fashion, to all other nodes in the graph. In the second phase, messages are retrieved in a similar Round-Robin process. This divides the communication load evenly, resulting in an optimal round complexity. Assuming that for each node i we have the set of its $k(i)$ messages $M_i = \{m_{i,1}, \dots, m_{i,k(i)}\}$, let D_i denote its recipients list. With these notations, the code of *Round-Robin Messaging* is given in Algorithm 3.

Lemma 2. *Given a bulk of messages in which:*

1. *Every node is the source of at most n messages.*
2. *Every node is the destination of at most n messages.*
3. *Every source node sends exactly the same information to all of its destination nodes and knows the content of its messages.*

Round-Robin-Messaging *delivers all messages in* 3 *rounds.*

Corollary 1. *Using* Round-Robin-Messaging*, all nodes can learn the complete structure of the graph in $|E|/n$ rounds.*

Algorithm *TriNeighbors* satisfies all the conditions of Lemma 2. We conclude that, employing *Round-Robin-Messaging*, the round complexity of *TriNeighbors*

Algorithm 3: Round-Robin-Messaging at node i.

```
R := ∅ // collects output
S := ∅ // collects source nodes and #messages for i
for $j \in V$ do
    send $m_{i,j \bmod k(i)}$ to $j$
    if $j \in D_i$ then send "notify $k(i)$" to $j$
for "notify $k(j)$" received from $j$ do $S := S \cup (j, k(j))$
$l := 1$
for $(j, k(j)) \in S$ do
    for $k \in \{1, \ldots, k(j)\}$ do
        send "request message from $j$" to $l$
        $l := l + 1$
for received "request message from $j$" do send $m_{j,i \bmod k(j)}$ to $j$
for received message $m$ do $R := R \cup \{m\}$
return $R$
```

becomes $\mathcal{O}(\Delta^2/n)$. If $\Delta \in \mathcal{O}(\sqrt{n})$ then the round complexity is $\mathcal{O}(1)$, and clearly optimal. More generally, any subgraph of diameter[6] $D \in \mathcal{O}(1)$ can be detected by each node exploring its D-hop neighborhood.

Corollary 2. *We can test for subgraphs of diameter D in $\mathcal{O}(\Delta^{D+1}/n)$ rounds.*

Bounded arboricity. The arboricity A of G is defined to be the minimum number of forests on V such that their union is G. Note that always $A \le \Delta$, and for many graphs $A \ll \Delta$. The arboricity bounds the number of edges in any subgraph of G in terms of its nodes. We exploit this property to devise an arboricity-based algorithm for triangle finding that we call *TriArbor*.

An overview of the TriArbor algorithm. We wish to employ the same strategy used by the naive $TriNeighbors$, that is "asking neighbors for their neighbors", in a more careful manner, so as to avoid having high degree nodes send their entire neighbor list to many nodes. This is achieved by having all nodes with degree at most $4A$ send their neighbor list to their neighbors and then shut down. In the next iteration, the nodes that have a degree at most $4A$ in the graph induced by the still active nodes do the same and shut down. As $2An'$ uniformly bounds the sum of degrees of any subgraph of G containing n' nodes, in each iteration at least half of the remaining nodes is shut down. Hence, the algorithm will terminate within $\mathcal{O}(\log n)$ iterations. To control the number of messages, in each iteration we consider triangles involving at least one node of low degree (in the induced subgraph of the still active nodes). This way, any triangle will be found once any of its nodes' degrees becomes smaller than $4A$.

Obviously, no node of low degree will have to send more than $4A$ messages in this scheme. However, it may be the case that a node receives more than $4A$

[6] The diameter of a graph is the maximum shortest path length over all pairs of nodes.

Algorithm 4: One iteration of TriArbor at node i.

```
1  // compute delegates
2  send $\delta'_i$ to all other nodes
3  compute assignment of delegates to high-degree nodes and neighbor sublists
4  // high-degree nodes distribute neighborhood
5  if $\delta'_i > 4A$ then
6      partition $\mathcal{N}'_i$ into $\lceil \delta'_i / 4A \rceil$ lists of length at most $4A$
7      send each sublist to the computed delegate
8      for $j \in \mathcal{N}'_i$ do notify $j$ of delegate assigned to it // only $i$ knows order of $\mathcal{N}'_i$
9  // let all delegates learn about $\mathcal{N}'_j$
10 if $i$ is delegate of some node $j$ then
11     denote by $D_j$ the set of delegates of $j$
12     denote by $L_{j,i} \subset \mathcal{N}'_j$ the sublist of neighbors received from $j$
13     for $k \in D_j$ do send $L_{j,i}$ to $k$
14     for received sublist $L_{j,k}$ do $\mathcal{N}'_j := \mathcal{N}'_j \cup L_{j,k}$
15 // low-degree nodes distribute neighborhoods
16 if $\delta'_i \leq 4A$ then
17     for $j \in \mathcal{N}'_i$ do
18         if $\delta'_j \leq 4A$ then send $\mathcal{N}'_i$ to $j$ // low-degree nodes handle load alone
19         else send $\mathcal{N}'_i$ to $j$'s delegate assigned to $i$
20 // check for triangles
21 for received $\mathcal{N}'_j$ (from $j$ with $\delta'_j \leq 4A$) do
22     if $\mathcal{N}'_i \cap \mathcal{N}'_j \neq \emptyset$ then
23         send "triangle" to all nodes // triangles with two low-degree nodes
24     else if $i$ is delegate of $k$ and $\mathcal{N}'_j \cap \mathcal{N}'_k \neq \emptyset$ then
25         send "triangle" to all nodes // triangle with one low-degree node
26 if received "triangle" then return true
27 else return false
```

messages in case it has many low-degree neighbors. To remedy that, low-degree nodes avoid sending their neighbor list to their high-degree neighbors directly, and instead send it to intermediate nodes we call *delegates*. The delegates share the load of testing their associated high-degree node's neighborhood for triangles involving a low-degree node. We present the algorithm assuming that A is known to the nodes; for details on how to remove this assumption, we refer to [6].

Choosing delegates. In each iteration, every delegate node will be assigned to a unique high-degree node, i.e., a node of degree larger than $4A$ in the subgraph induced by the nodes that are still active. In the following, we will discuss a single iteration of the algorithm. Denote by $G' := (V', E')$ some subgraph of G on n' nodes, where WLOG $V' = \{1, \ldots, n'\}$. Define δ'_i, Δ', $\mathcal{N}'_i$, etc. analogously to δ_i, Δ, $\mathcal{N}_i$, etc., but with respect to G' instead of G. We would like to assign to each node i exactly $\lceil \delta'_i/(4A) \rceil$ delegates such that each delegate is responsible for up to $4A$ of the respective high-degree node's neighbors.

This is feasible because the arboricity provides a bound of the number of nodes having at least a certain degree that is inversely proportional to this threshold, implying the following claim.

Claim. At least $n'/2$ of the nodes have degree at most $4A$ and the number of assigned delegates is bounded by n'.

Moreover, the assignment of delegates to high-degree nodes can be computed locally using a predetermined function of the degrees δ'_i. Thus, if every node communicates its degree δ'_i, all nodes can determine locally the assignment of delegates to high-degree nodes in a consistent manner.

The algorithm. Algorithm 4 shows the pseudocode of one iteration of *TriArbor*. The complete algorithm iterates until for all nodes $\delta'_i = 0$ and outputs "true" if in one of the iterations a triangle was detected and "false" otherwise.

The following claim holds because we are certain that in each iteration half of the nodes are of low degree and therefore eliminated.

Claim. TriArbor terminates within $\lceil \log n \rceil$ iterations.

Since triangles with a low-degree node are detected, correctness is immediate.

Lemma 3. TriArbor *correctly decides whether the graph contains a triangle.*

Round complexity of TriArbor. We examine the round complexity of one iteration of the algorithm. Obviously, announcing degrees takes a single round only. The following claims can be veryfied by carefully applying the communication strategies we already established.

Claim. High-degree nodes' neighborhoods can be distributed in two rounds.

Claim. Exchanging neighborhood sublists between delegates can be performed in four rounds.

Claim. Low-degree nodes' neighborhoods can be communicated in $3\lceil 32A^2/n \rceil$ rounds.

Finally, announcing that a triangle is found takes one round. Recall that in each iteration at least half of the nodes have low degree and are thus eliminated. All in all, we get the following result.

Theorem 3. *Algorithm* TriArbor *is correct. It can be implemented with a running time of $\mathcal{O}(\lceil A^2/n \rceil \log n)$ rounds.*

The correctness here follows from the fact that eventually every node is eliminated, and in the respective phase it has small degree. A triangle with at least two low-degree nodes will be detected because these nodes will exchange their neighborhoods. A triangle with only one low-degree node will be detected by at least one delegate of each of its high-degree nodes.

We note that a more sophisticated implementation of the approach is uniform and slightly faster.

Corollary 3. TriArbor *can be modified to be uniform and run in $\mathcal{O}(A^2/n + \log_{2+n/A^2} n)$ rounds.*

5 Randomization

Our randomized algorithm does not exhibit an as well-structured communication pattern as the presented deterministic solutions, hence it is difficult to efficiently organize the exchange of information by means of a deterministic subroutine. Therefore, we make use of a randomized routine from [11].

Theorem 4 ([11]). *Given a bulk of messages such that:*

1. *No node is the source of more than n messages.*
2. *No node is the destination of more than n messages.*
3. *Each source knows the content of its messages.*

For any predefined constant $c > 0$, all messages can be delivered in $\mathcal{O}(1)$ rounds with high probability (w.h.p.), i.e., with probability at least $1 - 1/n^c$.

The algorithm. When sampling randomly for triangles, we would like to use the available information as efficiently as possible. To this end, on the first iteration of Algorithm 5 all nodes sample randomly chosen induced subgraphs of a certain size and examine them for triangles. On subsequent iterations the size of the checked subgraphs is increased. Checking a subgraph of size s requires to learn about $\mathcal{O}(s^2)$ edges, while it tests for $\Theta(s^3)$ potential triangles. If $s \in \Theta(\sqrt{n})$, it thus takes a linear number of messages to collect such an induced subgraph at a node. Using the subroutine from Theorem 4, each node can sample such a graph in parallel in $\mathcal{O}(1)$ rounds. Intuitively, this means we can sample $\Theta(n^{5/2})$ subsets of three vertices in constant time. As $|\binom{V}{3}| \in \Theta(n^3)$, one therefore can expect to find a triangle quickly if at least $\Omega(\sqrt{n})$ triangles are present in G. If less triangles are in the graph, we need to sample more. In order to do this efficiently, it makes sense to increase s instead of just reiterating the routine with the same set size: The time complexity grows quadratically, whereas the number of sampled 3-vertex-subsets grows cubically. Finally, once the round complexity of an iteration hits $n^{1/3}$, we will switch to deterministic searching to guarantee termination within $\mathcal{O}(n^{1/3})$ rounds. Interestingly, the set size of $s = n^{2/3}$ corresponding to this running time ensures that even a single triangle in the graph is found with constant probability.

Round complexity. The last iteration dominates the running time.

Lemma 4. *If* TriSample *terminates after m iterations, the round complexity is in $\mathcal{O}(2^{2m})$ with high probability.*

Our aim is to bound the number of iterations needed to detect a triangle with probability at least $1 - \varepsilon$, as a function of the number of triangles in the graph. Let $T \subset \binom{V}{3}$ denote the set of triangles in G, where $|T| = t$.

On an intuitive level, the triangles are either scattered (i.e., rarely share edges) or clustered. If the triangles are scattered, then applying the inclusion-exclusion principle of the second order will give us a sufficiently strong bound on the probability of success. If the triangles are clustered, then there exists an

Algorithm 5: TriSample at node i.

```
1  $s := \sqrt{n}$ while $s < n^{1/3}$ do
2      choose uniformly random set of $s$ nodes $C_i$
3      for $j \in C_i$ do send the member list of $C_i$ to $j$
4      for received member list $C_j$ from $j$ do send $\mathcal{N}_i \cap C_j$ to $j$
5      $E_i := \emptyset$
6      for received $\mathcal{N}_j \cap C_i$ from $j$ do
7          for $k \in \mathcal{N}_j \cap C_i$ do $E_i := E_i \cup \{j, k\}$
8      if $G_i := (V, E_i)$ contains a triangle then send "triangle" to all nodes
9      if received "triangle" then return true
10     else $s := 2s$
11 run TriPartition and return its output // switch to deterministic strategy
```

edge that many of them share. Finding that specific edge is more likely than finding any specific triangle, and given this edge is found, the probability to find at least one of the triangles it participates in is large.

Bounding the probability of success using the inclusion-exclusion principle. We know by the inclusion-exclusion principle that

$$Pr[\text{triangle found}] \geq t \cdot Pr[\text{specific triangle found}] - \sum_{a \neq b \in T} Pr[a \text{ and } b \text{ found}].$$

For every $a \neq b \in T$ there are three cases to consider:

1. a and b are disjoint, that is $a \cap b = \emptyset$.
2. a and b share a single vertex, $|a \cap b| = 1$.
3. a and b share an edge, $|a \cap b| = 2$.

Definition 1. *For $r \in \{4, 5, 6\}$, $T_r \subseteq \binom{T}{2}$ is the set of pairs of distinct triangles in G that have together exactly r vertices. Denoting $t_r = |T_r|$, clearly $t_4 + t_5 + t_6 = \binom{t}{2} = |\binom{T}{2}|$.*

We define that $P_m = Pr[\text{triangle found in iteration } m]$ and also that $p_m = Pr[\text{node } i \text{ found triangle in iteration } m]$. For symmetry reasons the latter probability is the same for each node i.

Claim. For $0 < \varepsilon < 2$, if $p_m \geq \ln(2/\varepsilon)/n$ then $P_m \geq 1 - \varepsilon/2$.

With the above notations, from the inclusion-exclusion principle we infer that

$$p_m \geq t \cdot \left(\frac{s_m}{n - s_m + 3}\right)^3 - \sum_{k=4}^{6} t_k \cdot \left(\frac{s_m}{n - s_m}\right)^k . \tag{1}$$

Recall that we distinguish between the cases of "scattered" and "clustered" triangles. We now give these expressions a formal meaning by defining a threshold for t_4 in terms of t and a critical value $s(\varepsilon)$ of s_m. This value is defined as $s(\varepsilon) := \max\{2n^{2/3}t^{-1/3}\ln^{1/3}(2/\varepsilon), 2\sqrt{n\ln(2/\varepsilon)}\}$. The critical value stems from either of the following cases:

1. Scattered triangles - we wish to sample as many triangles as possible, and the number of triangles sampled grows cubically in s_m. The $n^{2/3}$ factor in the numerator reflects the fact that $s_m = n^{2/3}$ would imply that each triangle is sampled with constant probability.[7] Clearly having a lot of triangles in general improves the probability of success, hence the division by $t^{1/3}$.
2. Clustered triangles - it may be the case that all triangles share a single edge, hence we must sample this edge with probability at least $1 - \varepsilon/2$. For $s_m = \sqrt{n}$ each node samples $\Theta(n)$ edges, hence each edge is sampled with constant probability.

The round complexity bound for the case of scattered triangles now follows by an application of the inclusion-exclusion principle. The probability to find a pair of triangles sharing an edge is considerably greater than finding other pairs, as only 4 vertices need to be sampled (unlike 5 or 6 if they only share a vertex or are disjoint, respectively), therefore it dominates the subtracted amount in Equality (1). More specifically, the probability that some node samples $k \in \{3, \ldots, 6\}$ fixed nodes is $\Theta((s_m/n)^k)$. Computations reveal that for the relevant values of s_m, t_5 and t_6 cannot be large enough to compensate for the factors of $(s_m/n)^2$ and $(s_m/n)^3$ by which these probabilities differ, respectively. With regard to t_4, we can give a threshold, below which the respective term can be neglected. This is done by the following Lemma that provides a probabilistic guarantee to find a triangle, assuming that there are not many pairs of triangles sharing an edge.

Lemma 5. *If $t_4 \leq tn/(2s(\varepsilon))$ and n is sufficiently large, then a triangle will be found with probability at least $1 - \varepsilon/2$ in any iteration where $s_m \geq s(\varepsilon)$.*

The strategy employed for clustered triangles is to show that due to the bound on t_4, there exists an edge shared by many triangles. An elegant proof for this fact was given by Brendan McKay [14].

Definition 2. *For each edge $e \in E$, define $\Delta_e = |\{T_i : e \subseteq T_i\}|$. In other words, Δ_e is the number of triangles that e participates in. Denote $\Delta_{\max} = \max_{e \in E} \Delta_e$.*

Lemma 6. $\Delta_{\max} \geq 2t_4/3t$.

Proof. Consider a figure consisting of two triangles sharing an edge (this is basically K_4 with one edge removed). We count the occurrences of this figure in G in two different ways:

1. Observe that t_4 counts just that.
2. Pick one of the t triangles from T, choose one of its 3 edges, denote it e. Choose one of the other $\Delta_e - 1$ triangles that share e to complete the figure. Note that this counts every figure exactly twice, since we may pick either of the two triangles in the figure to be the first one. By definition $\Delta_e - 1 \leq \Delta_{\max} - 1$, hence we count at most $3t(\Delta_{\max} - 1)/2$ occurrences.

[7] *TriPartition* selects $\mathcal{O}(n^{2/3})$ vertices per node so that all sets of 3 nodes are covered.

By comparing 1. and 2., we conclude that indeed $t_4 \leq 3t(\Delta_{\max} - 1)/2$. □

Subsequently the analysis focuses on this special edge, showing that the probability to sample this edge and find a triangle containing it is sufficiently large.

Lemma 7. *If $t_4 > tn/(2 \cdot s(\varepsilon))$ then a triangle will be found with probability at least $1 - \varepsilon/2$ in any iteration where $s_m \geq s(\varepsilon)$.*

Theorem 5. *If G contains at least t triangles, for any $\varepsilon \geq 1/n^{\mathcal{O}(1)}$* TriSample *terminates within $\mathcal{O}(\min\{n^{1/3}t^{-2/3}\ln^{2/3}\varepsilon^{-1} + \ln\varepsilon^{-1}, n^{1/3}\})$ rounds with probability at least $1 - \varepsilon$. It always outputs the correct result.*

Remark 2. The running time bound from Theorem 5 is asymptotically tight, that is, there are graphs for which *TriSample* runs with probability at least ε for $\Omega(n^{1/3}t^{-2/3}\ln^{2/3}\varepsilon^{-1})$ or $\Omega(\ln\varepsilon^{-1})$ rounds, respectively.

6 Conclusions

In this work, we give a number of solutions for the task of checking a distributed input graph for small subgraphs, under the assumption of an underlying fully connected communication network with bandwidth of $\mathcal{O}(\log n)$ per link. Our results show that non-trivial running time bounds can be achieved and present some communication strategies that are successful in deriving such bounds. However, we provide no insight on lower bounds for the problem, although during our work we developed intuition that strong, i.e., super-polylogarithmic, bounds might exist. Due to the very powerful model, we consider it a highly challenging task to devise such a bound. As a first step, we suggest examining the class of *oblivious* algorithms, whose communication pattern is completely predefined. Algorithm 1 is such an algorithm, and we conjecture that its running time is essentially optimal, i.e., that any deterministic oblivious algorithm deciding whether there is a triangle in the input graph must run for $\tilde{\Omega}(n^{1/3})$ rounds.

Acknowledgements

The authors would like to thank Shiri Chechik for suggesting Algorithm 3, and Brendan McKay for his proof of Lemma 6. Danny Dolev is incumbent of the Berthold Badler Chair in Computer Science. Christoph Lenzen has been supported by the Swiss National Science Foundation and the Society of Swiss Friends of the Weizmann Institute of Science. This research project was supported in part by The Israeli Centers of Research Excellence (I-CORE) program, (Center No. 4/11), by the Google Inter-university center for Electronic Markets and Auctions, and by the Kabarnit Consortium, administered by the office of the Chief Scientist of the Israeli ministry of Industry and Trade and Labor.

References

1. Alon, N.: Testing subgraphs in large graphs. Random Structures and Algorithms 21, 359–370 (2002)
2. Alon, N., Kaufman, T., Krivelevich, M., Ron, D.: Testing triangle-freeness in general graphs. SIAM Journal on Discrete Math 22(2), 786–819 (2008)
3. Chechik, S.: Message distribution technique (2011), private communication
4. Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM Journal on Computing 14, 210–223 (1985)
5. Deo, N., Litow, B.: A Structural Approach to Graph Compression. In: Proc. 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS). pp. 91–101 (1998)
6. Dolev, D., Lenzen, C., Peled, S.: "Tri, Tri again": Finding Triangles and Small Subgraphs in a Distributed Setting. Computing Research Repository abs/1201.6652 (2012)
7. Gonen, M., Ron, D., Shavitt, Y.: Counting Stars and Other Small Subgraphs in Sublinear-Time. SIAM Journal on Discrete Mathematics 25(3), 1365–1411 (2011)
8. Grötzsch, H.: Zur Theorie der diskreten Gebilde, VII. Ein Dreifarbensatz fr dreikreisfreie Netze auf der Kugel. In: Math.-Nat. Reihe, vol. 8, pp. 109–120. Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg (1958/59)
9. Kashtan, N., Itzkovitz, S., Milo, R., Alon, U.: Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. Bioinformatics 20(11), 1746–1758 (2004)
10. Kothapalli, K., Scheideler, C., Onus, M., Schindelhauer, C.: Distributed Coloring in $\tilde{\mathcal{O}}(\sqrt{\log n})$ Bit Rounds. In: IPDPS (2006)
11. Lenzen, C., Wattenhofer, R.: Tight Bounds for Parallel Randomized Load Balancing. In: Proc. 43rd Symposium on Theory of Computing (STOC). pp. 11–20 (2011)
12. Lotker, Z., Patt-Shamir, B., Peleg, D.: Distributed MST for Constant Diameter Graphs. Distributed Computing 18(6) (2006)
13. Lotker, Z., Pavlov, E., Patt-Shamir, B., Peleg, D.: MST Construction in $\mathcal{O}(loglogn)$ Communication Rounds. In: Proc. 15th Symposium on Parallel Algorithms and Architectures (SPAA). pp. 94–100 (2003)
14. McKay (mathoverflow.net/users/9025), B.: If many triangles share edges, then some edge is shared by many triangles. MathOverflow, http://mathoverflow.net/questions/83939, http://mathoverflow.net/questions/83939 (version: 2011-12-20)
15. Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U.: Network Motifs: Simple Building Blocks of Complex Networks. Science 298(5594), 824–827 (2002), http://dx.doi.org/10.1126/science.298.5594.824
16. Patt-Shamir, B., Teplitsky, M.: The Round Complexity of Distributed Sorting: Extended Abstract. In: PODC. pp. 249–256 (2011)
17. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. Society for Industrial and Applied Mathematics (2000)
18. Sarma, A.D., Holzer, S., Kor, L., Korman, A., Nanongkai, D., Pandurangan, G., Peleg, D., Wattenhofer, R.: Distributed Verification and Hardness of Distributed Approximation. In: 43rd Symposium on Theory of Computing (STOC) (2011)