אלגוריתמים למציאת משולשים ותת-גרפים קטנים במערכת מבוזרת

שיר פלד

הוגש כמילוי חלקי של החובות לתואר מוסמך במדעים

מנחה: פרופ' דני דולב

אדר תשע"ב

בית הספר להנדסה ומדעי המחשב
על שם רחל וסלים בנין
האוניברסיטה העברית בירושלים
ישראל

תקציר

מציאת תת-גרפים קטנים, המכונים גם מוטיבי-רשת (Network Motifs), בתוך גרף קלט גדול יותר, היא בעיה ידועה ונחקרת בתורת הגרפים ובמדעי המחשב בשל שימושיה בתחומים תיאורטיים ומעשיים כאחד. למרות זאת, למיטב ידיעתנו הבעיה טרם נחקרה בהקשר של מערכות מבוזרות. בעבודה זו אנחנו בוחנים את השאלה במודל הבא: יהי $G=(V,E)$ גרף על n קודקודים, ויהי M_d גרף על d קודקודים, עבור $d \in O(1)$ כלשהו. האם M_d הוא תת-גרף של G? אנו בוחנים שאלה זו במודל מבוזר, אשר בו כל n המעבדים מקושרים אלו לאלו, וכל הודעה מכילה לכל היותר $O(\log n)$ ביטים. בעבודה מתואר אלגוריתם דטרמיניסטי פשוט אשר פותר את הבעיה בתוך $O(n^{(d-2)/d}/\log n)$ סיבובי תקשורת. עבור המקרה הפרטי שבו M_d הוא משולש, מוצג אלגוריתם הסתברותי אשר תוחלת סיבובי התקשורת הדרושים לו היא $O(n^{1/3}/(t^{2/3}+1))$, באשר t הוא מספר המשולשים בגרף הקלט, ובהסתברות גבוהה (השואפת פולינומית ל 1) זמן הריצה הוא $O(\min\{n^{1/3}\log^{2/3}n/(t^{2/3}+1),n^{1/3}\})$.

בנוסף, מתוארים אלגוריתמים דטרמיניסטיים המותאמים לגרפים דלילים (Sparse): מוצג אלגוריתם שמסוגל למצוא את כל תתי הגרפים בקוטר D ב $O(\Delta^{D+1}/n)$ סיבובי תקשורת בכל גרף שבו הדרגה המקסימלית היא Δ. עבור המקרה הפרטי של משולשים, אנו מציגים אלגוריתם המבוסס על פרמטר העֲצִיּוּת (arboricity) של גרף הקלט, וסיבוכיות סיבובי התקשורת היא $O(|E|/n+\log n)\subseteq O((A^2)/n+\log n)$, באשר A היא העציות של G.

Finding Triangles and Small Subgraphs in a Distributed Setting

Shir Peled

Submitted in partial fulfilment of the requirements
of the degree of Master of Science

Under the supervision of Prof. Danny Dolev

March 2012

Rachel and Selim Benin
School of Computer Science and Engineering
The Hebrew University of Jerusalem
Israel

Acknowledgments

I thank my advisor, Danny Dolev, for his sure-handed guidance and support. A special thanks goes to Christoph Lenzen, who suggested many of the ideas presented in this work, was the devil's advocate when necessary, and kept me on my mathematical toes. I would also like to thank Shiri Chechik, for suggesting Algorithm 5, and Brendan McKay for his elegant proof of Lemma 6.9 in [14].

Lastly, I would like to thank my family, for nearly everything else.

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 specially suited for sparse graphs. In any graph 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 arboricity-based algorithm, featuring a round complexity of $\mathcal{O}((A^2)/n + \log n) \subseteq \mathcal{O}(|E|/n + \log n)$, where A denotes the arboricity of G.

Contents

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 are locally connected. 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 when the communication graph is 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;[1] 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.

Detailed Contributions. In Chapter 4, 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 Chapter 5. For the special case of triangles, we present a more intricate way of checking neighborhoods that 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 $A \in \mathcal{O}(1)$ [6], 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. For example, a graph of uniform degree $\sqrt{n}$ has arboricity at most $\sqrt{n}$, but $n^{3/2}/2$ edges.

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 available, it can be much more efficient to

[1]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.

randomly inspect small portions of the graph. In Chapter 6, we present a triangle-finding algorithm that does just that, yielding that for every $\varepsilon \geq 1/n$ and a graph containing $t \geq 1$ triangles, a triangle 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 of the algorithms presented 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.

2 Related Work

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, for instance, biological networks, as they play a key role in information processing mechanisms of biological regulation networks.

Even motifs as simple as 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, planar triangle-free graphs are long since known to be 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.

2.1 Subgraph Listing

In subgraph listing, the aim is to list all copies of a given subgraph. Such is the usual approach adopted in the aforementioned biological applications. The number of copies in the graph, that may be as high as $\Theta(n^3)$ for triangles, sets an obvious lower bound for the running time of such algorithms, rendering instances with many triangles harder in some sense. It has been shown by Chiba and Nishizeki in [4] that a graph with bounded arboricity contains $\mathcal{O}(n)$ copies of cliques of a given size. They further describes a linear-time algorithm to list them all, as do Chrobak and Eppstein (though by a different algorithm) in [5].

2.2 Property Testing

Property testing algorithms, as opposed to subgraph listing, 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.

2.3 State of the art

It is hard to compare the approach we adopt with ones used in the centralized model, as we may employ n processors in parallel, whereas the main restriction we adher to is the communication volume. The approach that most resembles ours, is that of query-based algorithms, that are allowed to sample edges or degrees. The query complexity is a plausible analogue of messages passed between processors. Instead of the hamming distance, as used in the Property Testing paradigm, the number of triangles in the graph is the significant parameter. Some such query based algorithms were suggested in the centralized model, where the parameter to determine is the number of triangles in the graph. Sadly, the lower bounds for such algorithms assume restrictions on the type of queries[1] that cannot be justified in our model [7]. In conclusion, though extensive work has been done in the field of finding small subgraphs in a larger graph, to the best of our knowledge little to none of it was in the field of distributed computing, at least partly due to the triviality of this question in the *local* model. We hope that exploring these seemingly simple problems in the CONGEST model, would not only contribute to understanding the

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

problem itself, but also illustrate some generally applicable techniques of distributing a computational task in a highly symmetric way, such that the communication load is adequately balanced.

3 Model and Problem

3.1 Model

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.[1] 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.

3.2 Problem

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 (i) discover whether M_d is a subgraph of G and (ii) characterize the trade-off between the number s of copies of M_d that are present in G and the time required to decide on (i).

[1]Note that it is trivial to make all nodes terminate in the same round due to the full connectivity.

4 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 all messages quickly, and 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.

4.1 Full-Knowledge Message Passing

For a certain limited family of algorithms that we call *oblivious algorithms*, 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.

Definition 4.1. *A distributed algorithm $\mathcal{A}$ in our model is said to be* oblivious *if the sources and destinations of all messages are determined in advance, regardless of the input graph G, and each source can determine the content of its messages from its input.*

For instance, the rather trivial *AllNeighbors* algorithm (see Algorithm 1) is oblivious and results in all nodes having complete knowledge of the structure of G.

Algorithm 1: AllNeighbors at node i.

```
1 for $j \in V$ do
2     send $\mathcal{N}_i$ to $j$
```

As communication is peer-to-peer, in actuality, all iterations of the *For* loop can be executed in parallel. If all nodes execute the above routine, after n rounds every node gets all lists of immediate neighbors of nodes, and can therefore reconstruct the graph locally. We will see later on, in Chapter 5, that a similar algorithm can be realized more efficiently using a more evolved communication strategy.

We now turn to describing our communication pattern for oblivious algorithms. To this end, we will need the following claim that is a corollary of Hall's marriage theorem.

Claim 4.2. *Every d-regular bipartite multigraph is a disjoint union of d perfect matchings.*

Proof. By induction on d. For $d = 1$ the graph is a perfect matching by definition.

Assume that the claim holds for some d, and let $H = (L, R, E)$ be a $(d+1)$-regular bipartite graph. Let $S \subseteq L$ be some set of vertices, and define $\Gamma(S) := \{u \in R : \exists v \in S \text{ s.t. } (v, u) \in E\}$. By regularity, the sum of degrees in S is exactly $(d+1)|S|$, and by the pigeonhole principle and regularity $|\Gamma(S)| \geq (d+1)|S|/(d+1) = |S|$, satisfying Hall's marriage condition thus implying that a perfect matching exists. Removing the perfect matching found from the graph leaves a d-regular bipartite graph that is a disjoint union of d perfect matchings by the induction hypothesis. Adding those d perfect matchings to the one just obtained completes the proof. □

Lemma 4.3. *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.

Proof. WLOG we assume every node is the source of exactly n messages, and it is the destination of exactly n messages as well (having a node "sending message to itself" is not a problem). We will label every message to node i with a different $j \in \{1, ..., n\}$ and denote the messages to node i according to this labeling by $m_{i,1}, m_{i,2}, ..., m_{i,n}$.

We define a *good* labeling to be such that no node initially holds two messages labeled $m_{j,k}$ and $m_{l,k}$ for some l, j, k with $l \neq j$. Assuming we start with a good labeling, we argue that the message passing algorithm whose pseud-code is given in Algorithm 2 terminates successfully after two rounds. We will later show that a good labeling is always attainable.

Algorithm 2: Deterministic Message Passing at node i holding message set S.

```
1  $S' := \emptyset$, $S'' := \emptyset$
2  // first stage (distribution)
3  for $m_{j,k} \in S$ do
4      send $m_{j,k}$ to node $k$
5  for received message $m$ do
6      $S' := S' \cup \{m\}$
7  // second stage (delivery)
8  for $m_{j,k} \in S'$ do
9      send $m_{j,k}$ to node $j$
10 for received message $m$ do
11     $S'' := S'' \cup \{m\}$
12 return $S''$
```

If our labeling is indeed good, then during the first stage every node sends at most a single message to each of the other nodes, and therefore can dispose of all the messages in S within the first round. Due to the unique labeling of the messages, after the first stage node i holds all messages of type $m_{k,i}$, and since there is at most one such message for each k, all of them are emitted within a single round in the second stage. Clearly, the labeling also ensures that the returned set S'' will contain exactly the messages whose destination is i.

It remains to show that we can find a good labeling. Recall that sources and destinations are known in advance to all nodes, so each node can compute the labeling locally. If all nodes use the same deterministic algorithm, this will result in all nodes using the exact same labeling.

Let $B = (L, R, E)$ be a bipartite multigraph, where $|L| = |R| = n$. We denote $L = \{l_1, ..., l_n\}$ and $R = \{r_1, ..., r_n\}$. For every message in the initial bulk with source i and destination k we add an edge (l_i, r_k) to E. B is clearly an n-regular multigraph, and by Claim 4.2 it is a disjoint union of n perfect matchings. We now choose a perfect matching in this graph, remove its edges and label the messages represented by those edges thus: for every edge (l_i, r_k) in the matching we label its corresponding message $m_{k,1}$. After removing those edges we find another perfect matching, and for every edge (l_i, r_k) in it we label the corresponding message $m_{k,2}$ and so on, until we remove the nth perfect matching from the graph. Since a perfect matching is easy to find (using maximal-flow algorithms), a good labeling can be found efficiently. □

Corollary 4.4. *An oblivious algorithm in which each node sends and receives at most $T(n)$ messages can be completed within $2\lceil T(n)/n\rceil$ rounds, by repeatedly using* *the message passing routine described above.*

4.2 TriPartition - Finding triangles deterministically

Next, we present an algorithm that finds whether there are triangles in G. The algorithm is not oblivious, since in the final step every node broadcasts whether it found a triangle or not to all other nodes. This last broadcast message is obviously dependent on other messages transferred throughout the algorithm, therefore it violates the obliviousness requirement that the order of the messages will not matter. However, having every node broadcast its results takes a single round only. The first part of the algorithm is oblivious, allowing us to apply the message passing algorithm previously stated to it. As the oblivious part of the algorithm terminates, we run the final broadcasting round.

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 (ordered) 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 of the subsets in the triplet, showing the following claim.

Claim 4.5. *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 $t_3 \in S_{i,3}$.*

Proof. Each node checks for triangles that are contained in its triplet of subsets by executing *TriPartition*, whose pseudo-code is given in Algorithm 3. □

Algorithm 3: TriPartition at node i.

```
1  $E_i := \emptyset$
2  for $1 \le j < k \le 3$ 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
6              $E_i := E_i \cup \{l, m\}$
7  if there exists a triangle in $G_i := (V, E_i)$ then
8      send "triangle" to all nodes
9  if received "triangle" from some node then
10     return true
11 else
12     return false
```

Theorem 4.6. *Using* Deterministic Message Passing *algorithm as communication subroutine,* TriPartition *determines correctly whether there exists a triangle in G within $\mathcal{O}(n^{1/3})$ rounds.*

Proof. Correctness follows from Claim 4.5, as node i collects exactly the edges between pairs of subsets in its triplet. The round complexity is deduced as follows. Since the assignment of set triplets is static, each node i knows which nodes need to learn about which of its neighbors. Since there are $n^{1/3}$ subsets of size $n^{2/3}$, each of which participates in $n^{1/3}$ triplets involving the subset containing i, the node needs to transmit at most $n^{4/3}$ messages.[1] On the other hand, each node needs to learn about less than $\binom{3}{2}n^{4/3}$ edges, one for each pair of nodes from two of its subsets. By Corollary 4.4, this information can thus be communicated within $\mathcal{O}(n^{1/3})$ rounds. The algorithm terminates one additional round later, completing the proof. □

Remark 4.7. *The partial obliviousness of* TriPartition *allows us to avoid sending IDs of nodes. That is, instead of encoding the respective sublist of neighbors by listing their IDs, 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. Repeating the argument from the proof of Theorem 4.6 we may hence improve the round complexity of* TriPartition *to $\mathcal{O}(n^{1/3}/\log n)$.*

4.3 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 4.8. *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.*

Proof. Similarly to the 3-vertex case, we apply Corollary 4.4, and due to obliviousness, we may assume all messages are sequences of bits as in Remark 4.7. □

[1]Clearly, a neighbor can be encoded using $\log n$ bits.

5 Deterministic Algorithms for 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 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.

5.1 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, the trivial *TriNeighbors* algorithm, whose pseudo-code is given in Algorithm 4 may be much faster than *dClique0* algorithm.

Algorithm 4: 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
5          $E_i := E_i \cup \{j, k\}$
6  if there exists a triangle in $G_i := (V, E_i)$ then
7      send “triangle” to all nodes
8  if received “triangle” from some node then
9      return true
10 else
11     return false
```

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 by Shiri Chechik [3]. Assuming that (i) no node is source of more than n messages in total, (ii) no node is 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. Given for each node i the sets of its messages $M_i = \{m_{i,1}, \ldots, m_{i,k(i)}\}$ and destinations D_i, the algorithm's pseudo-code is given in Algorithm 5.

Algorithm 5: Round-Robin-Messaging at node i.

```
1  $R := \emptyset$ // collects output
2  $S := \emptyset$ // collects source nodes and #messages for $i$
3  for $j \in V$ do
4      send $m_{i,j \bmod k(i)}$ to $j$
5      if $j \in D_i$ then
6          send “notify $k(i)$” to $j$
7  for “notify $k(j)$” received from $j$ do
8      $S := S \cup (j, k(j))$
9  $l := 1$
10 for $(j, k(j)) \in S$ do
11     for $k \in \{1, \ldots, k(j)\}$ do
12         send “request message from $j$” to $l$
13         $l := l + 1$
14 for received “request message from $j$” do
15     send $m_{j,i \bmod k(j)}$ to $j$
16 for received message $m$ do
17     $R := R \cup \{m\}$
18 return $R$
```

Lemma 5.1. *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.*

Proof. In the first loop of the algorithm every node sends one message to every other node; note that it is feasible to send both the message $m_{i,j \bmod k(i)}$ and a potential notification at the same time. The cyclic nature of the message distribution in this first loop assures that any consecutive $k(i)$ nodes together hold all $k(i)$ messages of node i, exactly one at each node. By Condition 1, $k(i) \leq n$ for each node i, i.e., each node indeed sends out all its messages. By Condition 2, for each node the querying loop will request at most one message from each node. Since exactly $k(j)$ messages are requested from a node j, the set of messages retrieved in the second last loop contains M_j. By Condition 3 and due to the previous notification of destination nodes, this is exactly the set of messages to be received from j. This shows correctness of the algorithm. As we also argued that in total three communication rounds are required, this shows the statement of the lemma. □

Algorithm *TriNeighbors* satisfies all the conditions of Lemma 5.1. We conclude that, employing *Round-Robin-Messaging*, the round complexity of *TriNeighbors* 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[1] $D \in \mathcal{O}(1)$ can be detected by each node exploring its D-hop neighborhood.

Corollary 5.2. *We can test for subgraphs of diameter D in $\mathcal{O}(\Delta^{D+1}/n)$ rounds.*

5.2 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 $A \leq \Delta$, 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*.

5.2.1 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. In order to control the number of messages sent in each iteration, we consider triangles involving at least one node of low degree (in the induced subgraph of the still active nodes). As we will find a triangle once any of its nodes' degrees becomes smaller than $4A$, all triangles are will be detected.

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 $4A$ messages

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

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 them 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.

Note that in the presented form, the algorithm is not uniform, i.e., it is assumed that A is known. We will later discuss how to remove this assumption and slightly improving its round complexity at the same time.

5.2.2 TriArbor algorithm

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 the respective values without a prime, 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.

Claim 5.3. *At least $n'/2$ of the nodes have degree at most $4A$ and the number of assigned delegates is bounded by n'.*

Proof. We have that

$$|\{i \in V' \mid \delta'_i > 4A\}| \leq \frac{1}{4A} \sum_{i \in V'} \delta'_i \leq \frac{|E'|}{2A} < \frac{n'}{2}.$$

Therefore,

$$\sum_{\substack{i \in V' \\ \delta'_i > 4A}} \left\lceil \frac{\delta'_i}{4A} \right\rceil \leq \frac{n'}{2} + \frac{1}{4A} \sum_{i=1}^{n'} \delta'_i \leq \frac{n'}{2} + \frac{|E'|}{2A} < n',$$

i.e., less than n' delegates are required. □

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 6 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.

Algorithm 6: One iteration of TriArbor at node i.

```
// compute delegates
send $\delta'_i$ to all other nodes
compute assignment of delegates to high-degree nodes and neighbor sublists
// high-degree nodes distribute their neighborhood
if $\delta'_i > 4A$ then
    partition $\mathcal{N}'_i$ into $\lceil \delta'_i / 4A \rceil$ lists of length at most $4A$
    send each sublist to the computed delegate
    for $j \in \mathcal{N}'_i$ do
        notify $j$ of the delegate assigned to it // only $i$ knows the order of $\mathcal{N}'_i$,
        hence communication required
// let all delegates learn about $\mathcal{N}'_j$
if $i$ is delegate of some node $j$ then
    denote by $D_j$ the set of delegates of $j$
    denote by $L_{j,i} \subset \mathcal{N}'_j$ the sublist of neighbors received from $j$
    for $k \in D_j$ do
        send $L_{j,i}$ to $k$
    for received sublist $L_{j,k}$ do
        $\mathcal{N}'_j := \mathcal{N}'_j \cup L_{j,k}$
// low-degree nodes distribute their neighborhoods
if $\delta'_i \leq 4A$ then
    for $j \in \mathcal{N}'_i$ do
        if $\delta'_j \leq 4A$ then
            send $\mathcal{N}'_i$ to $j$ // low-degree nodes can handle load themselves
        else
            send $\mathcal{N}'_i$ to the delegate of $j$ assigned to $i$
// check for triangles
for received $\mathcal{N}'_j$ (from $j$ with $\delta'_j \leq 4A$) do
    if $\mathcal{N}'_i \cap \mathcal{N}'_j \neq \emptyset$ then
        send "triangle found" to all nodes // detected triangles involving two
        low-degree nodes
    else if $i$ is delegate of $k$ and $\mathcal{N}'_j \cap \mathcal{N}'_k \neq \emptyset$ then
        send "triangle found" to all nodes // detected triangle involving one
        low-degree node
if received "triangle found" then
    return true
else
    return false
```

Claim 5.4. TriArbor *terminates within* $\lceil \log n \rceil$ *iterations.*

Proof. Follows directly from Claim 5.3, as in each iteration at least half of the nodes are eliminated. □

Lemma 5.5. TriArbor *correctly decides whether the graph contains a triangle or not.*

Proof. Clearly, there are no false positives, as in each iteration, nodes will only claim that a triangle is found if they learned about an edge connecting two nodes in the same neighborhood (either their own or the node whose delegate they are).

Recall that by Claim 5.3, there are sufficiently many delegates available, and we observed that the assignment can be computed as the same function of the (current) degrees.

Now, assume the graph contains some triangle $\{i_1, i_2, i_3\}$. There must be some iteration in which one of the nodes, say i_1, has degree $\delta'_{i_1} \leq 4A$ and the triangle is still in the subgraph induced by active nodes: By Claim 5.4, eventually all nodes get eliminated, while each edge connecting two high-degree nodes will still be present in the subgraph induced by the active nodes of the next iteration.

We distinguish two cases. If in the respective iteration it also holds that $\delta'_{i_2} \leq 4A$, then i_1 will send i_2 its neighbor list (with respect to the induced subgraph), and i_2 will detect the triangle. Otherwise, i_1 will send its current neighbor list to one of i_2's delegates. As i_2 splits its neighbor list and distributes it among its delegates, which share their sublist with all other delegates, this delegate will detect the triangle. Hence, in both cases, the triangle will eventually be discovered, this information be spread among the nodes, and all nodes will compute the correct output. □

Round Complexity of TriArbor

We examine the time complexity of one iteration of the algorithm. Obviously, announcing degrees takes a single round only.

Claim 5.6. *The distribution of high-degree nodes' neighborhoods can be performed in two rounds.*

Proof. Every node i with $\delta'_i > 4A$ partitions its neighbor list and sends it, totalling in at most $\delta'_i < n'$ messages. As each node is delegate of at most one node, no more than $4A$ messages need to be received. Observe that since all nodes are aware of the assignment of delegates as well as all node degrees, we can apply Lemma 4.3 to see that all messages can be delivered in two rounds. Notifying neighbors of their assigned delegates takes one message for each neighbor. However, both tasks are independent, therefore we can merge the respective messages, resulting in a total of two rounds. □

Claim 5.7. *Exchanging neighborhood sublists between delegates can be implemented in four rounds.*

Proof. Every delegate holds a sublist of at most $4A$ of the neighbors of the node i it has been assigned to. Hence, it needs to send at most $\lceil \delta_i'/4A \rceil 4A < 2\delta_i' < 2n'$ messages. Similarly, it receives less than $2n'$ messages. As delegates are aware of the number of messages to exchange, Lemma 4.3 shows that we can implement this communication in four rounds. □

Claim 5.8. *The distribution of low-degree nodes' neighborhoods can be performed in $3\lceil 32A^2/n \rceil$ rounds.*

Proof. Every node i with $\delta_i' \leq 4A$ sends δ_i' messages to each of its low-degree neighbors and to one delegate of each high-degree neighbor, i.e., at most $16A^2$ messages. Similarly, both low-degree nodes and delegates receive at most $16A^2$ messages. As the low-degree nodes send their entire neighborhood to all destinations, applying Lemma 5.1 repeatedly yields that this communication can be performed in $3\lceil 32A^2/n \rceil$ rounds (note that nodes may have to receive $32A^2/n$ messages because they may have low degree and be delegate at the same time). □

Finally, announcing a found triangle takes one more round. All in all, we get the following result.

Theorem 5.9. *Algorithm* TriArbor *is correct. Using our* Deterministic Message Passing *and* Round-Robin-Messaging *algorithms, it can be implemented with a running time of $\mathcal{O}(\lceil A^2/n \rceil \log n)$ rounds.*

Proof. Correctness was shown in Lemma 5.5. Combining Claims 5.6, 5.7, and 5.8, we see that a single iteration of the algorithm can be implemented with running time $\mathcal{O}(A^2/n)$. By Claim 5.4, the total running time is thus bounded by $\mathcal{O}(\lceil A^2/n \rceil \log n)$ rounds. □

Corollary 5.10. *The iterations of TriArbor can be parallelized, reducing the round complexity to $\mathcal{O}(A^2/n + \log n)$.*

Proof. We first let all nodes execute the a short announcement phase, whose pseudocode is given in Algorithm 7, storing all received values.

Algorithm 7: QuickDecomposition at node i.

```
1 $V' := V$
2 for $\lceil \log n \rceil$ iterations do
3     send $\delta'_i := |\mathcal{N}_i \cap V'|$ to all nodes
4     $V' := V' \setminus \{j \in V' \,|\, \delta'_j \leq 4A\}$
```

The aim of this "announcement phase" is that for all iterations, the nodes will know in advance which nodes are of high degree, which are of low degree, and which nodes are the delegates of which other nodes. As all this information can be inferred from the degree distributions at the beginning of each iteration, which by itself is also a function of the degrees in the previous iteration, the above routine performs this task.

Our goal is now to show that we can "merge" the further communication of all iterations such that the total running time is bounded by $\mathcal{O}(A^2/n)$. Note that nodes satisfy up to three roles during the execution of the algorithm: they may act as (i) high-degree nodes, (ii) delegates, and (iii) low-degree nodes. However, according to Claim 5.3, during the entire execution of the algorithm, the total number of delegates is bounded by

$$\sum_{i=1}^{\infty} \frac{n}{2^{i-1}} = 2n.$$

We conclude that we can assign delegates in a way such that each node acts as delegate in at most two iterations. Furthermore, each node is a low-degree node in exactly one iteration, as afterwards it is eliminated from the subgraph induced by active nodes. Therefore, the asymptotic bounds from Claims Claim 5.7 and Claim 5.8 can be shown analogously also for the merged execution. Regarding Claim 5.6 observe that since the number of active nodes decreases exponentially, no node sends more than $2n$ messages in its role as high-degree node during the course of the algorithm. Overall, we obtain the same asymptotic running time bound of $\mathcal{O}(A^2/n)$ for the communication performed by all iterations of the algorithm as we did before for a single one. Adding the initial $\mathcal{O}(\log n)$ rounds for determining the active nodes in each iteration, the claimed running time bound follows. $\square$

Furthermore, we can utilize the "excess capacity" of the communication system in case $A^2 \ll n$ to further reduce the number of iterations.

Corollary 5.11. TriArbor *can be modified to run in* $\mathcal{O}(A^2/n + \log_{2+n/A^2} n)$ *rounds.*

Proof. Instead of choosing the threshold for low-degree nodes to be $4A$, we pick $\max\{4A, \lceil\sqrt{n}\rceil\}$. If $4A \geq \lceil\sqrt{n}\rceil$ the algorithm behaves as before. Otherwise, we have that in each iteration at most

$$\frac{1}{\sqrt{n}} \sum_{i \in V'} \delta_i' \leq \frac{2An'}{\sqrt{n}}$$

remain active, implying that all nodes are eliminated in $\mathcal{O}(\log_{2+n/A^2} n)$ rounds.

It remains to show that if $\lceil \sqrt{n} \rceil \geq 4A$, all iterations can be executed in parallel in $\mathcal{O}(1)$ rounds. Observe that Claims 5.6 and 5.7 hold for any choice of the threshold. Hence, as the number of nodes decreases exponentially also if $\lceil \sqrt{n} \rceil \geq 4A$, the distribution of high-degree nodes' neighborhoods and the communication among delegates can be performed in $\mathcal{O}(1)$ rounds in total. Regarding the messages sent by low-degree nodes, in total less than $\lceil \sqrt{n} \rceil^2 \leq 2n$ (instead of $16A^2$) messages need to be conveyed, and each delegate receives at most $\lceil \sqrt{n} \rceil^2 \leq 2n$ messages. As each node is delegate at most twice, this requires $\mathcal{O}(1)$ rounds as well. Hence, taking into account Theorem 5.9 and Corollary 5.10, the statement follows. $\square$

It remains to remove the dependence of the algorithm on knowledge on A.

Corollary 5.12. *A variant of* TriArbor *can be executed successfully in $\mathcal{O}(A^2/n + \log_{2+n/A^2} n)$ rounds with no prior knowledge of A.*

Proof. Denote by $\bar{\delta}' := (\sum_{i \in V'} \delta_i')/n'$ the average degree of the graph of currently active nodes G'. Instead of setting the threshold for high-degree nodes to $\max\{4A, \lceil \sqrt{n} \rceil\}$ as in Corollary 5.11, we pick $\max\{2\bar{\delta}', \lceil \sqrt{n} \rceil\}$. We have that

$$\frac{1}{2\bar{\delta}'} \sum_{i \in V'} \delta_i' = \frac{n'}{2},$$

i.e., still at least half of the active nodes are eliminated in each iteration. Moreover,

$$2\bar{\delta}' = \frac{2}{n'} \sum_{i \in V'} \delta_i' \leq 4A,$$

hence, arguing analogously to Corollary 5.10, we can perform all iterations together in $\mathcal{O}(A^2/n)$ rounds. $\square$

Finally, we can bound the running time of the algorithm in terms of the number of edges of G.

Corollary 5.13. *The uniform algorithm from Corollary 5.12 successfully decides whether G contains a triangle within $\mathcal{O}(|E|/n + \log n)$ rounds.*

Proof. In [4] it is shown that, for any graph, $A \in \mathcal{O}(\sqrt{|E| + n})$. Plugging this bound into the running time guaranteed by Corollary 5.12 yields the claim. $\square$

6 Randomized Algorithm for General Graphs

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 6.1 ([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 $1 - 1/n^c$.

We try to give some intuition on why this theorem is true. A key idea is that, using randomization, it is possible to first distribute a fairly large fraction of the messages in a roughly balanced manner, i.e., such that $n - o(n)$ messages for each destination can be delivered by each node sending at most $\mathcal{O}(1)$ messages to the respective destination. Subsequently, we can make "more effort" to distribute the remaining $o(n^2)$ messages evenly. To this end, these messages are duplicated and sent redundantly to different randomly chosen relay nodes. The number of copies is limited in order to not overload the network. This results in an exponentially amplified probability to succeed in delivering each message. Hence, after one iteration of this scheme, we will have much less messages to deliver, enabling to increase the number of redundant copies used for each message further, and so on. Repeated application results in delivery of all messages in $\mathcal{O}(1)$ rounds.

6.1 The Algorithm

When sampling randomly for triangles, we would like to use the available information as efficiently as possible. To this end, the algorithm samples induced subgraphs of increasing size. Checking a subgraph of size s requires to learn about $\mathcal{O}(s^2)$, 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 the induced subgraph at some node and test for triangles. Using the subroutine from Theorem 6.1, each node can sample such a graph in parallel in $\mathcal{O}(1)$ rounds. Intuitively, this means to 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 running time 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.

Algorithm 8: TriSample at node i.

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

6.2 Round complexity

Our first observation is that the last iteration dominates the running time of the algorithm.

Lemma 6.2. *If* TriSample *terminates after m iterations, the round complexity is in $\mathcal{O}(2^{2m})$ with high probability.*

Proof. Let s_k denote s in the k^{th} iteration, hence $s_1 = \sqrt{n}, s_2 = 2\sqrt{n}, ..., s_m = 2^{m-1}\sqrt{n}$. Clearly, in the k^{th} iteration, every node i sends out exactly s_k^2 messages to nodes j informing them about C_i. Since the sets C_i are chosen independently, by Chernoff's bound with high probability every node j in the k^{th} iteration is a member of $\mathcal{O}(s_k)$ sets C_i, and therefore receives $\mathcal{O}(s_k)$ such subsets. It follows that, w.h.p., it will respond with in total at most $\mathcal{O}(s_k^2)$ messages telling the respective nodes i about $\mathcal{N}_j \cap C_i$. The recipients of this messages will have to bear a load of at most $|C_i|^2 = s_k^2$. By Theorem 6.1, these message exchanges may be accomplished in $\mathcal{O}(s_k^2/n)$ rounds w.h.p. If the algorithm terminates after the m^{th} iteration, the overall round complexity is therefore in $\mathcal{O}(\sum_{k=1}^{m} s_k^2/n) = \mathcal{O}(s_m^2)$. □

Corollary 6.3. *If* TriSample *is guaranteed to find a triangle with probability $1-\varepsilon/2$ once s passes some threshold $s(\varepsilon)$, then the round complexity to find a triangle with probability $1-\varepsilon$ is $\mathcal{O}(s(\varepsilon)^2/n)$.*

Proof. By Lemma 6.2 and the union bound. □

Remark 6.4. *Note that $s_m \leq n^{2/3}$ by the loop condition and afterwards the algorithm simply executes* TriPartition. *The round complexity is therefore always in $\mathcal{O}(n^{1/3})$ with high probability.*

6.2.1 Proof overview

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 $|T| = t$. WLOG, assume that $t \in o(n^{2/3})$ (otherwise we consider only a subset of 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 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.

6.2.2 Bounding the probability of success using the inclusion-exclusion principle

We know by the inclusion-exclusion principle that

$$Pr[\text{a triangle is found}] \geq t \cdot Pr[\text{exactly one triangle is found}] - \sum_{a \neq b \in T} Pr[\text{at least } a \text{ and } b \text{ are found}]. \tag{6.1}$$

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, $|a \cap b| = 2$.

Observe that for every constant r and set of vertices V_0 s.t. $|V_0| = r$, it holds that:

$$(s_m/(n-s_m+r))^r \leq Pr\left[V_0 \text{ is chosen in the } m^{th} \text{ iteration}\right] \leq (s_m/(n-s_m))^r. \tag{6.2}$$

Definition 6.5. *$T_r \in \binom{T}{2}$ is the set of pairs of distinct triangles in G that have together exactly r vertices. Denoting $t_r = |T_r|$, clearly $t_4 + t_5 + t_6 = \binom{t}{2} = |\binom{T}{2}|$.*

For brevity we denote $p_m = Pr[\text{a specific node found a triangle in the } m^{th} \text{ iteration}]$. For symmetry reasons this probability is the same for each individual node executing *TriSample*. We also denote $P_m = Pr[\text{a triangle is found in the } m^{th} \text{ iteration}]$.

Claim 6.6. *For $0 < \varepsilon < 2$, if $p_m \geq \ln(2/\varepsilon)/n$ then $P_m \geq 1 - \varepsilon/2$.*

Proof. Recall that each node i chooses C_i independently. Consequently, the probability of no triangle being found in the m^{th} iteration is at most

$$\begin{aligned}\left(1 - \frac{\ln(2/\varepsilon)}{n}\right)^n &= \left(\left(1 - \frac{1}{n\ln^{-1}(2/\varepsilon)}\right)^{n\ln^{-1}(2/\varepsilon)}\right)^{\ln(2/\varepsilon)} \\ &\leq e^{-\ln(2/\varepsilon)} \\ &= \frac{\varepsilon}{2}.\end{aligned}$$

□

With the above notations, we combine Equality (6.2) with the inclusion-exclusion principle to obtain:

$$p_m \geq t \cdot \left(\frac{s_m}{n - s_m + 3}\right)^3 - \sum_{k=4}^{6} t_k \cdot \left(\frac{s_m}{n - s_m}\right)^k . \tag{6.3}$$

We distinguish between the cases of "scattered" and "clustered" triangles. We now give these expressions a formal meaning via a threshold for t_4 in terms of t and a critical value $s(\varepsilon)$ of s_m that is $s(\varepsilon) := \max\{2n^{2/3}t^{-1/3}\ln^{1/3}(2/\varepsilon), 2\sqrt{n\ln(2/\varepsilon)}\}$. 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.[1] 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.

6.2.3 Scattered triangles

Assume $t_4 \leq tn/(2s(\varepsilon))$.

Lemma 6.7. *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 $s_m \geq s(\varepsilon)$.*

Proof. We rewrite Equality (6.3) as

$$p_m \geq \frac{t \cdot s_m^3(n - s_m)^3 - t_4 \cdot s_m^4 \cdot (n - s_m)^2 - t_5 s_m^5 \cdot (n - s_m) - t_6 s_m^6}{(n - s_m)^6}.$$

Due to the loop condition in *TriSample*, $s_m \leq n^{2/3} \in (1 - o(1))n$, therefore

$$\begin{aligned} p_m &\geq \frac{(1 - o(1))ts_m^3 n^3 - t_4 s_m^4 n^2 - t_5 s_m^5 n - t_6 s_m^6}{n^6} \\ &= \frac{s_m^3((1 - o(1))tn^3 - t_4 s_m n^2 - t_5 s_m^2 n - t_6 s_m^3)}{n^6}. \end{aligned}$$

[1]Observe that *TriPartition* samples exactly $n^{2/3}$ vertices per node in a way covering all subsets of 3 nodes.

As $t_4 \leq tn/(2s(\varepsilon)) \leq tn/(2s_m)$, this can be estimated further by

$$p_m \geq \frac{s_m^3((\frac{1}{2} - o(1))tn^3 - t_5 s_m^2 n - t_6 s_m^3)}{n^6}.$$

By definition, $t_4 + t_5 + t_6 = \binom{t}{2} \leq t^2/2$, therefore there exist $\beta, \gamma \geq 0$ such that $t_5 = \beta t^2, t_6 = \gamma t^2$ and $\beta + \gamma \leq 1/2$. Using this notation,

$$\begin{aligned} p_m &\geq \frac{s_m^3((\frac{1}{2} - o(1))tn^3 - \beta t^2 s_m^2 n - \gamma t^2 s_m^3)}{n^6} \\ &= \frac{s_m^3 t((\frac{1}{2} - o(1))n^3 - \beta t s_m^2 n - \gamma t s_m^3)}{n^6}. \end{aligned}$$

By the loop condition in *TriSample*, $s_m \leq n^{2/3}$. Recalling that we assume $t \in o(n^{2/3})$, this becomes

$$\begin{aligned} p_m &\geq \frac{s_m^3 t((\frac{1}{2} - o(1))n^3 - \beta t n^{\frac{7}{3}} - \gamma t n^2)}{n^6} \\ &\geq \frac{s_m^3 t(\frac{1}{2} - o(1))n^3}{n^6}. \end{aligned}$$

Given that $s_m \geq s(\varepsilon) \geq 2n^{2/3} \ln^{1/3}(2/\varepsilon)/t^{1/3}$ and, we have for sufficiently large n that

$$\begin{aligned} p_m &\geq \frac{s_m^3 t(\frac{1}{2} - o(1))n^3}{n^6} \\ &= \frac{(\frac{1}{2} - o(1))s_m^3 t}{n^3} \\ &\geq \frac{2tn^2 \ln(2/\varepsilon)}{tn^3} \\ &= \frac{2\ln(2/\varepsilon)}{n}. \end{aligned}$$

By Claim 6.6 this implies that the probability of finding a triangle in iteration m is at least $1 - \varepsilon/2$. $\square$

6.2.4 Clustered triangles

Assume $t_4 > t \cdot n(2 \cdot s_m)$. The strategy employed here is to show that due to the bound on t_4, there exists an edge shared by many triangles. Subsequently the analysis focuses on this edge, showing that the probability to sample this edge and find a triangle containing it is sufficiently large.

Definition 6.8. *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 6.9. $\Delta_{\max} \geq 2t_4/3t$.

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.

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

Remark 6.10. *The tightness of this bound can be confirmed by examining a complete graph.*

Lemma 6.11. *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 $s_m \geq s(\varepsilon)$.*

Proof. Assume WLOG that $e_{\max} = \{x, y\}$ is an edge shared by $\Delta_{\max}$ triangles. The probability of a node choosing both endpoints of $e_{\max}$ is $s_m(s_m - 1)/(n(n-1)) \geq 0.99 s_m^2/n^2$ (for large values of n, as $s_m \geq \sqrt{n}$). Given that this edge is chosen, the probability of missing all of the $\Delta_{\max}$ vertices that complete a triangle with $e_{\max}$ is at most $(1 - \Delta_{\max}/n)^{s_m - 2}$. By Lemma 6.9 and our assumption on t_4, we deduce that $\Delta_{\max} \geq n/(3s_m)$, therefore the probability of a specific node missing all triangles comprising $e_{\max}$, conditional to $e_{\max}$ being chosen, is at most $(1 - 1/(3s_m))^{s_m - 2} \leq e^{-1/3}/0.99$ (for large values of n). Fixing some node i, we obtain that

$$\begin{aligned}
p_m &\geq Pr\left[i \text{ finds a triangle with } e_{\max} | x, y \in C_i\right] \cdot Pr[x, y \in C_i] \\
&\geq \frac{(0.99 - e^{-1/3}) s_m^2}{n^2} \\
&\geq \frac{s(\varepsilon)^2}{4n^2} \\
&\geq \frac{\ln(2/\varepsilon)}{n}.
\end{aligned}$$

Applying Claim 6.6, we conclude $P_m \geq 1 - \varepsilon/2$. □

6.2.5 Deriving the Bound on the Round Complexity

Definition 6.12. *$m(n,t,\varepsilon)$ is the minimal integer such that $s_{m(n,t,\varepsilon)} \geq s(\varepsilon)$.*

Corollary 6.13. *Given that G contains at least t triangles, for every $\varepsilon > 0$, with probability at least $1-\varepsilon/2$* TriSample *terminates at the latest in iteration $m(n,t,\varepsilon)$.*

Proof. Combine Lemmas 6.7 and 6.11. □

Theorem 6.14. *Given that G contains at least t triangles, for every $\varepsilon \geq 1/n$, with probability at least $1-\varepsilon$* TriSample *terminates within $\mathcal{O}(\min\{n^{1/3}t^{-2/3}\ln^{2/3}\varepsilon^{-1} + \ln\varepsilon^{-1}, n^{1/3}\})$ rounds. It always outputs the correct result.*

Proof. By Corollary 6.13, the algorithm terminates with probability $1-\varepsilon/2$ after no more than $m(n,t,\varepsilon)$ iterations. By Corollary 6.3, it thus terminates with probability $1-\varepsilon$ within $\mathcal{O}(2^{2m(n,t,\varepsilon)}$ rounds. By Remark 6.4 the round complexity is always in $\mathcal{O}(n^{1/3})$ with high probability, altogether showing the stated bound.

Correctness follows from the fact that the algorithm terminates if it either finds a triangle, or after executing *TriPartition*, according to Theorem 4.6 with the correct output. □

Corollary 6.15. *Algorithm* TriSample *terminates within $\mathcal{O}(n^{1/3}/t^{2/3})$ rounds in expectation and within $\mathcal{O}(\max\{n^{1/3}\ln^{2/3}n/t^{2/3} + \ln n, n^{1/3}\})$ rounds w.h.p.*

Remark 6.16. *We can make sure the algorithm* always *terminates within $\mathcal{O}(n^{1/3})$ rounds by stopping it after $n^{1/3}$ rounds and switching to* TriPartition *even if $s_m < n^{2/3}$.*

Corollary 6.17. *For every $\varepsilon \geq 1/n^{\mathcal{O}(1)}$, it is possible to distinguish with probability $1-\varepsilon$ between the cases that G is triangle-free and that G has at least t_0 triangles within $\mathcal{O}(t_0^{-2/3}n^{1/3}\ln^{2/3}(1/\varepsilon) + \ln(1/\varepsilon))$ rounds.*

Proof. Set $s \leq 2\max\{t_0^{-1/3}2n^{2/3}\ln^{1/3}(1/\varepsilon), 2\sqrt{n\ln(1/\varepsilon)}\}$ to be the loop condition *TriSample*. If no triangle has been found during the loop, we output that G is triangle-free. The running time bound follows from Corollary 6.3, and correctness with probability $1-\varepsilon$ is due to Theorem 6.14. □

6.3 Tightness of the Analysis

Consider a graph G with $t < n-2$ triangles, all sharing a specific edge e_0. To find a triangle, some node must sample both ends of e_0, and this happens with probability

$s_m(s_m - 1)/(n(n-1))$ per node. The probability that all nodes miss e_0 is at least $(1 - s_m(s_m - 1)/(n(n-1)))^n$. If $s_m \in o(\sqrt{n \ln(1/\varepsilon)})$ then this probability is in $1 - \omega(\varepsilon)$.

Consider a graph G with t disjoint triangles $t < n/3$. The probability of a specific node to miss a specific triangle is at least $1-(s_m(s_m-1)(s_m-2))/(n(n-1)(n-2)) \geq 1-((s_m-2)/n)^3$. By the union bound, the probability of a specific node missing all triangles is at least $1-t((s_m-2)/n)^3$. The probability that all nodes miss all triangles is therefore at least $(1-t((s_m-2)/n)^3)^n$. Assuming that $s_m \in o(t^{-1/3}n^{2/3}\ln^{1/3}(1/\varepsilon))$, this is in $(1 - o(n^{-1}\ln(1/\varepsilon))^n \subseteq \omega(\varepsilon)$.

Bibliography

[1] N. Alon. Testing subgraphs in large graphs. *Random Structures and Algorithms*, 21:359–370, 2002.

[2] N. Alon, T. Kaufman, M. Krivelevich, and D. Ron. Testing triangle-freeness in general graphs. *SIAM Journal on Discrete Math*, 22(2):786–819, 2008.

[3] S. Chechik. Message distribution technique, 2011. Private communication.

[4] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14:210–223, 1985.

[5] M. Chrobak and D. Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theoretical Computer Science*, 86(2):243 – 266, 1991.

[6] N. Deo and B. Litow. A Structural Approach to Graph Compression. In *Proc. 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 91–101, 1998.

[7] M. Gonen, D. Ron, and Y. Shavitt. Counting Stars and Other Small Subgraphs in Sublinear-Time. *SIAM Journal on Discrete Mathematics*, 25(3):1365–1411, 2011.

[8] H. Grötzsch. Zur Theorie der diskreten Gebilde, VII. Ein Dreifarbensatz fr dreikreisfreie Netze auf der Kugel. In *Math.-Nat. Reihe*, volume 8, pages 109–120. Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg, 1958/59.

[9] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004.

[10] K. Kothapalli, C. Scheideler, M. Onus, and C. Schindelhauer. Distributed Coloring in $\tilde{\mathcal{O}}(\sqrt{logn})$ Bit Rounds. In *IPDPS*, 2006.

[11] C. Lenzen and R. Wattenhofer. Tight Bounds for Parallel Randomized Load Balancing. In *Proc. 43rd Symposium on Theory of Computing (STOC)*, pages 11–20, 2011.

[12] Z. Lotker, B. Patt-Shamir, and D. Peleg. Distributed MST for Constant Diameter Graphs. *Distributed Computing*, 18(6), 2006.

[13] Z. Lotker, E. Pavlov, B. Patt-Shamir, and D. Peleg. MST Construction in $\mathcal{O}(loglogn)$ Communication Rounds. In *Proc. 15th Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 94–100, 2003.

[14] B. McKay (mathoverflow.net/users/9025). If many triangles share edges, then some edge is shared by many triangles. MathOverflow. http://mathoverflow.net/questions/83939 (version: 2011-12-20).

[15] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network Motifs: Simple Building Blocks of Complex Networks. *Science*, 298(5594):824–827, 2002.

[16] B. Patt-Shamir and M. Teplitsky. The Round Complexity of Distributed Sorting: Extended Abstract. In *PODC*, pages 249–256, 2011.

[17] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* Society for Industrial and Applied Mathematics, 2000.

[18] A. D. Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed Verification and Hardness of Distributed Approximation. In *43rd Symposium on Theory of Computing (STOC)*, 2011.