# Three Notes on Distributed Property Testing

**Guy Even*[1], Orr Fischer[2], Pierre Fraigniaud†[3], Tzlil Gonen[2], Reut Levi[4], Moti Medina[4], Pedro Montealegre‡[5], Dennis Olivetti[6], Rotem Oshman§[2], Ivan Rapaport¶[7], and Ioan Todinca[8]**

1 **Tel-Aviv University, School of Electrical Engineering, Israel**
`guy@eng.tau.ac.il`
2 **Tel Aviv University, Computer Science Department, Israel**
`orrfischer@mail.tau.ac.il,tzlilgon@mail.tau.ac.il,roshman@tau.ac.il`
3 **CNRS and University Paris Diderot, France**
**pierre.fraigniaud@irif.fr**
4 **Max Planck Institute for Informatics, Saarland Informatics Campus, Germany**
`rlevi@mpi-inf.mpg.de, mmedina@mpi-inf.mpg.de`
5 **Facultad de Ingeniería y Ciencias, Universidad Adolfo Ibáñez, Chile**
**p.montealegre@uai.cl**
6 **Gran Sasso Science Institute, L'Aquila, Italy**
**dennis.olivetti@gssi.infn.it**
7 **DIM-CMM (UMI 2807 CNRS), Universidad de Chile, Chile**
**rapaport@dim.uchile.cl**
8 **Université d'Orléans, INSA Centre Val de Loire, LIFO EA 4022, France**
**ioan.todinca@univ-orleans.fr**

## Abstract

In this paper we present distributed property-testing algorithms for graph properties in the CONGEST model, with emphasis on testing subgraph-freeness. Testing a graph property $\mathcal{P}$ means distinguishing graphs $G = (V, E)$ having property $\mathcal{P}$ from graphs that are $\varepsilon$-far from having it, meaning that $\varepsilon|E|$ edges must be added or removed from $G$ to obtain a graph satisfying $\mathcal{P}$.

We present a series of results, including:

- Testing $H$-freeness in $O(1/\varepsilon)$ rounds, for any constant-sized graph $H$ containing an edge $(u, v)$ such that any cycle in $H$ contain either $u$ or $v$ (or both). This includes all connected graphs over five vertices except $K_5$. For triangles, we can do even better when $\varepsilon$ is not too small.
- A deterministic CONGEST protocol determining whether a graph contains a given tree as a subgraph in constant time.
- For cliques $K_s$ with $s \geq 5$, we show that $K_s$-freeness can be tested in $O(m^{\frac{1}{2}-\frac{1}{s-2}} \cdot \varepsilon^{-\frac{1}{2}-\frac{1}{s-2}})$ rounds, where $m$ is the number of edges in the network graph.
- We describe a general procedure for converting $\varepsilon$-testers with $f(D)$ rounds, where $D$ denotes the diameter of the graph, to work in $O((\log n)/\varepsilon) + f((\log n)/\varepsilon)$ rounds, where $n$ is the number of processors of the network. We then apply this procedure to obtain an $\varepsilon$-tester for testing whether a graph is bipartite and testing whether a graph is cycle-free. Moreover, for cycle-freeness, we obtain a *corrector* of the graph that locally corrects the graph so that the corrected graph is acyclic. Note that, unlike a tester, a corrector needs to mend the graph in many places in the case that the graph is far from having the property.

* Work done while visiting Max Planck Institute for Informatics.
† Additional support from ANR Project DESCARTES, and from INRIA Project GANG.
‡ This work was partially supported by CONICYT via Basal in Applied Mathematics
§ Orr Fischer, Tzlil Gonen and Rotem Oshman are supported by the Israeli Centers of Research Excellence (I-CORE) program, (Center No.4/11) and by BSF Grant No. 2014256.
¶ This work was partially supported by Fondecyt 1170021, Núcleo Milenio Información y Coordinación en Redes ICM/FIC RC130003

© Guy Even, Orr Fischer, Pierre Fraigniaud , Tzlil Gonen, Reut Levi, Moti Medina, Pedro Montealegre, Dennis Olivetti, Rotem Oshman, Ivan Rapaport and Ioan Todinca;
licensed under Creative Commons License CC-BY
31st International Symposium on Distributed Computing (DISC 2017).
Editor: Andréa Richa; Article No. 15; pp. 15:1–15:30
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

These protocols extend and improve previous results of [Censor-Hillel et al. 2016] and [Fraigniaud et al. 2016].

**1998 ACM Subject Classification** F.2 ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY

**Digital Object Identifier** 10.4230/LIPIcs.DISC.2017.15

# 1 General Introduction

Distributed decision refers to tasks in which the computing elements of a distributed system have to collectively decide whether the system satisfies some given boolean predicate on system states. If the system state is legal, i.e., it satisfies the given predicate, then all computing elements must accept; if the system state is illegal, then at least one computing element must reject. Distributed property testing is a relaxed variant of distributed decision, which only requires distinguishing legal states from states that are "far from" being legal. (The notion of "farness" depends on the context.)

In the context of distributed network computing, one is interested in deciding or testing whether the actual network, modeled as a simple connected graph, satisfies some given predicate on graphs; e.g., bipartiteness, cycle-freeness, subgraph-freeness, etc. For a positive distance parameter $\varepsilon \leq 1$, a graph $G$ with $m$ edges is said to be $\varepsilon$-far from satisfying a given property $P$ if removing and/or adding up to $\varepsilon\, m$ edges from/to $G$ cannot result in a graph satisfying $P$.

In this paper we study distributed decision in general, and distributed property testing in particular, in the framework of distributed network computing, under the standard CONGEST model. This paper is the result of merging the three papers [16], [17], and [18] that were concurrently submitted to the 31st International Symposium on Distributed Computing (DISC 2017), which independently showed overlapping results, using different methods and ideas. To highlight the different approaches to the problem, we chose to present a short version of each of the three papers in the form of three notes.

**The Subgraph-Freeness Problem.** Each of the three notes presented here gives results on subgraph-freeness: we are given a constant-size graph $H$, and we wish to determine whether the network graph contains $H$ as a subgraph or not. In the *property testing* relaxation of the problem, we only need to distinguish the case where the network graph is $H$-free from the case where it is *$\varepsilon$-far* from $H$-free, in the sense that at least an $\varepsilon$-fraction of the graph's edges must be removed to eliminate all copies of $H$.

Hereafter, we provide a summary of the results and methods in each paper.

## 1.1 Summary of the Results and Techniques

**Note #1: Color-Coding Based Algorithms for Testing Subgraph-Freeness.** This note uses a technique called *color-coding* [4] to design randomized algorithms for property-testing subgraph-freeness in $O(1/\varepsilon)$ rounds, for any subgraph $H$ that contains an edge $(u, v)$ such that any cycle in $H$ contains at least one of $u$ and $v$. In the case of trees, the color-coding technique yields an $O(1)$-round algorithm for testing exactly whether the graph contains the given tree or not. In addition, for cliques $K_s$ with $s \geq 3$, we show that $K_s$-freeness can be tested in $O(m^{\frac{1}{2}-\frac{1}{s-2}} \cdot \varepsilon^{-\frac{1}{2}-\frac{1}{s-2}})$ rounds, where $m$ is the number of edges in the network graph.

In the special case of triangles, $K_3$-freeness can actually be solved in $O(1)$ rounds in $n$-node networks, i.e., in a number of rounds independent from $\varepsilon$, assuming $\varepsilon \geq \min\{m^{-\frac{1}{3}}, \frac{n}{m}\}$.

**Note #2: Deterministic Tree Detection and Applications in Distributed Property Testing.** In this note, we propose a generic construction for designing deterministic distributed algorithms detecting the presence of any given tree $T$ as a subgraph of the input network, performing in a constant number of rounds in the CONGEST model. Our construction also provides randomized algorithms for testing $H$-freeness for every graph pattern $H$ that can be decomposed into an edge and a tree, with arbitrary connections between them, also running in $O(1)$ rounds in the CONGEST model. This generalizes the results in [9, 19, 20], where algorithms for testing $K_3$, $K_4$, and $C_k$-freeness for every $k \geq 3$ were provided.

**Note #3: Algorithms for Testing and Correcting Graph Properties in the CONGEST Model.** In Section 4, we design and analyze distributed testers in the distributed CONGEST model all of which work in the general model. Stating that our testers work in the general model means that our measure of distance between two graphs is measured by adding or removing $\varepsilon \cdot m$ edges.

*Diameter dependency reduction and its Applications.* In Section 4.2 we describe a general procedure for converting $\varepsilon$-testers with $f(D)$ rounds, where $D$ denotes the diameter of the graph, to work in $O((\log n)/\varepsilon) + f((\log n)/\varepsilon)$ rounds, where $n$ is the number of processors of the network. We then apply this procedure to obtain an $\varepsilon$-tester for testing whether a graph is bipartite. The improvement of this tester over state of the art is twofold: (a) the round complexity is $O(\varepsilon^{-1} \log n)$, which improves over the $\text{Poly}(\varepsilon^{-1} \log n)$-round algorithm by Censor-Hillel et al. [9, Thm. 5.2], and (b) our tester works in the general model while [9] works in the more restrictive bounded degree model. Moreover, the number of rounds of our bipartiteness tester meets the $\Omega(\log n)$ lower bound by [9, Thm. 7.3], hence our tester is asymptotically optimal in terms of $n$. We then apply this "compiler" to obtain a cycle-free tester with number of rounds of $O(\varepsilon^{-1} \cdot \log n)$, thus revisiting the result by [9, Thm. 6.3]. The last application that we consider is "how to obtain a *corrector* of the graph by using this machinery?" Namely, how to design an algorithm that locally corrects the graph so that the corrected graph satisfies the property. For cycle-freeness, we are able to obtain also an $\varepsilon$-corrector. Note that, unlike a tester, a corrector needs to mend the graph in many places in the case that the graph is far from having the property.

*Testers for $H$-freeness for $|V(H)| \leq 4$.* In Section 4.3 we design algorithms for testing (in the general model) whether the network is $H$-free for any connected $H$ of size up to four with round complexity of $O(\varepsilon^{-1})$.

*Testers for tree-freeness.* In Section 4.4, we first generalize the global tester by Iwama and Yoshida [25] of testing $k$-path freeness to testing the exclusion of any tree, $T$, of order $k$. Our tester has a one sided error and it works in the general graph model with random edge queries. This algorithm can be simulated in the CONGEST model in $O(k^{k^2+1} \cdot \varepsilon^{-k})$ rounds.

# 2 Note #1: Color-Coding Based Algorithms for Testing Subgraph-Freeness

## 2.1 Introduction

The aim of this part of the paper is to improve our understanding of the question: "which types of excluded subgraphs can be tested in constant time?". We also explore several related questions, such as whether limiting the maximum degree in the graphs helps (by analogy to the bounded-degree model in sequential property testing), whether we can test $H$-freeness in *sublinear* time for some subgraphs $H$ for which no constant-time algorithm is known, and whether there are cases where we can test $H$-freeness with *no dependence on the distance parameter* $\varepsilon$, even when $\varepsilon$ is sub-constant (e.g., $\varepsilon = O(1/\sqrt{n})$). Using new ideas and combining them with previous techniques, we are able to simplify and extend prior work, and point out some surprising answers to the questions above, which point to several aspects where distributed property testing for subgraph-freeness differs from the sequential analogue.

**Our Results.** First we give simple randomized algorithms for testing subgraph-freeness, using the *color-coding* technique from [4], originally used to find cycles of fixed constant size sequentially.

We begin by giving a color-coding based algorithm which tests $k$-cycle freeness in $O(1/\varepsilon)$ rounds (for constant $k$). Next we show that for any tree $T$, we can test $T$-freeness exactly (without the property-testing relaxation) in constant time. Both of the results extend to directed graphs in the directed version of the CONGESTmodel. Combining the two algorithms, we give a class of graphs $\mathcal{H}$ such that for any constant-sized $H \in \mathcal{H}$, we can test $H$-freeness in $O(1/\varepsilon)$ rounds. The class $\mathcal{H}$ consists of all graphs $H$ containing an edge $\{u, v\}$ such that each cycle in $H$ includes either $u$ or $v$ (or both). This includes all graphs of size 5 except for the 5-clique, $K_5$.

We then turn our attention to the special case of cliques. We present a different approach for detecting triangles, which allows us to *eliminate* the dependence of the running time on $\varepsilon$ when $\varepsilon$ is not too small: if $n$ is the number of nodes and $m$ is the number of edges, and we are promised that $\varepsilon \geq \min\left\{m^{-1/3}, n/m\right\}$, then we can test triangle-freeness in $O(1)$ rounds (independent of the actual value of $\varepsilon$). We extend this approach to cliques of any size $s \geq 3$, and show that $K_s$-freeness can be tested in $O\left(\varepsilon^{-1/2-1/(s-2)} m^{1/2-1/(s-2)}\right)$ rounds. In particular, for constant $\varepsilon$ and $s = 5$, we can test $K_5$-freeness in $O(m^{1/6})$ rounds. We also modify the algorithm to work in *constant time* in graphs whose maximum degree $\Delta$ is not too large with respect to the total number of edges, $\Delta = O((\varepsilon m)^{1/(s-2)})$.

## 2.2 Preliminaries

We generally work with undirected graphs, unless indicated otherwise. We let $N(v)$ denote the neighbors of $v$, and $d(v)$ the degree of $v$. Note that throughout the paper, when we use the term *subgraph*, we do not mean *induced subgraph*; we say that $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V, E' \subseteq E$.

We say that a graph $G = (V, E)$ is *$\varepsilon$-far from property $\mathcal{P}$* if at least $\varepsilon|E|$ edges need to be added to or removed from $E$ to obtain a graph satisfying $\mathcal{P}$.

The goal in distributed property testing for $H$-freeness is to solve the following problem: if the network graph $G$ is $H$-free, then with probability 2/3, all nodes should accept. On the other hand, if $G$ is $\varepsilon$-far from being $H$-free, then with probability 2/3, some node should reject.

We rely on the following fundamental property, which serves as the basis for most sequential property testers for $H$-freeness:

▶ Property 1. Let $G$ be $\varepsilon$-far from being $H$-free, then $G$ has $\varepsilon m/|E(H)|$ edge-disjoint copies of $H$.

Our algorithms assign random colors to vertices of the graph, and then look for a copy of the forbidden subgraph $H$ which received the "correct colors". Formally we define:

▶ **Definition 1 Properly-colored subgraphs.** Let $G = (V, E)$ and $H = ([k], F)$ be graphs, and let $G' = (V', E')$ be a subgraph of $G$ that is isomorphic to $H$. We say that $G'$ is *properly colored* with respect to a mapping $color_V : V \to [k]$ if there is an isomorphism $\varphi : V' \to [k]$ from $G'$ to $H$ such that for each $u \in V'$ we have $color_V(v) = \varphi(v)$.

## 2.3 Detecting Constant-Size Cycles

In this section we show that $C_k$-freeness can be tested in $O(1/\varepsilon)$ rounds in the CONGEST model for any constant integer $k > 2$.

▶ **Theorem 2.** *For any constant $k > 2$, there is a 1-sided error distributed algorithm for testing $C_k$-freeness which uses $O(1/\varepsilon)$ rounds.*

The key idea of the algorithm is to assign each node $u$ of the graph a random color $color(u) \in [k]$. The node colors induce a coloring of both orientations of each edge, where $color(u, v) = (color(u), color(v))$. We discard all edges that are not colored $(i, (i+1) \bmod k)$ for some $i \in [k]$; this eliminates all cycles of size less than $k$, while preserving a constant fraction of $k$-cycles with high probability.

Next, we look for a properly-colored $k$-cycle by choosing a random directed edge $(u_0, u_1)$[1] and carrying out a $k$-round *color-coded BFS* from node $u_0$: in each round $r = 0, \ldots, k-1$, the BFS only explores edges colored $(r, (r+1) \bmod k)$. After $k$ rounds, if the BFS reaches node $u_0$ again, then we have found a $k$-cycle.

Next we describe the implementation of the algorithm in more detail. We do not attempt to optimize the constants. To simplify the analysis, fix a set $\mathcal{C}$ of $\varepsilon m/k$ edge-disjoint $k$-cycles (which we know exist if the graph is $\varepsilon$-far from being $C_k$-free). We abuse notation by also treating $\mathcal{C}$ as the set of edges participating in the cycles in $\mathcal{C}$.

For the analysis, it is helpful to think of the algorithm as first choosing a random edge and then choosing random colors, and this is the way we describe it below.

**Choosing a Random Edge.** It is not possible to get all nodes of the graph to explicitly agree on a uniformly random directed edge in constant time (unless the graph has constant diameter), but we can emulate the effect as follows: each node $u \in V$ chooses a uniformly random weight $w(e) \in [n^4]$ for each of its edges $e$. (Note that each edge has two weights, one for each of its orientations.) Implicitly, the directed edge we selected is the edge with the smallest weight in the graph, assuming that no two directed edges have the same weight.

▶ Observation 1. With probability at least $1 - 1/n^2$, all weights in the graph are unique.

Let $\mathcal{E}_U$ be the event that all edge weights are unique. Conditioned on $\mathcal{E}_U$, the directed edge with the smallest weight is uniformly random. Let $e_0$ be this edge; implicitly, $e_0$ is the edge we select. (However, nodes do not initially know which edge was selected, or even if a single edge was selected.)

[1] What we really want to do is choose a random node with probability proportional to its degree; choosing random edge is a simple way to do that.

Since the set $\mathcal{C}$ contains $\varepsilon m/k$ edge-disjoint $k$-cycles, and the graph has a total of $m$ edges, we have:

▶ Observation 2. We have $\Pr[e_0 \in \mathcal{C} \mid \mathcal{E}_U] = \varepsilon$.

Let $\mathcal{E}_{Cyc}$ be the event that $e_0 \in \mathcal{C}$, and let $C_0 = \{u_0, u_1, \ldots, u_{k-1}\}$ be the cycle to which $e_0$ belongs given $\mathcal{E}_C$, where $e_0 = (u_0, u_1)$.

**Color Coding.** In order to eliminate cycles of length less than $k$, we assign to each node $u$ a uniformly random color $color(u) \in [k]$. Node $u$ then broadcasts $color(u)$ to its neighbors.

Since the colors are independent of the edge weights, we have:

▶ Observation 3. $\Pr[\forall i \in [k] \ : \ color(u_i) = i \mid \mathcal{E}_C, \mathcal{E}_U] = \frac{1}{k^k}$.

Let $\mathcal{E}_{Col}$ be the event that each $u_i$ received color $i$. Combining our observations above yields:

▶ **Corollary 3.** $\Pr[\mathcal{E}_U \cap \mathcal{E}_{Cyc} \cap \mathcal{E}_{Col}] > \frac{\varepsilon}{2k^k}$.

Next we show thatn when $\mathcal{E}_U, \mathcal{E}_{Cyc}$ and $\mathcal{E}_{Col}$ all occur, we find a $k$-cycle.

**Color-Coded BFS.** Each node $u$ stores the weight $wgt_u$ associated with the lightest edge it has heard of so far, and the root $root_u$ of the BFS tree to which it currently belongs. Initially, $wgt_u$ is set to the weight of the lightest of $u$'s outgoing edges, and $root_u$ is set to $u$.

In each round $r = 0, \ldots, k-1$ of the BFS, nodes $u$ with color $r$ send $(u, wgt_u, root_u)$ to their neighbors, and nodes $v$ with color $r+1$ update their state: if they received a message $(u, wgt_u, root_u)$ from a neighbor $u$, they set $wgt_v$ to the lightest weight they received, and $root_v$ to the root associated with that weight.

After $k$ rounds, if some node colored 0 receives a message $(v, wgt_v, root_v)$ where $root_v = u$, then it has found a $k$-cycle, and it rejects.

In Section 2.5, we will use the same $C_k$-freeness algorithms, but some nodes will be prohibited from taking certain colors. We incorporate this in Algorithm 1 by having some nodes whose state is abort. These nodes do not forward BFS messages and do not participate in the algorithm.

▶ **Lemma 4.** *If $\mathcal{E}_U, \mathcal{E}_{Cyc}$ and $\mathcal{E}_{Col}$ all occur, and if in addition the cycle $C_0$ contains no nodes whose state is* abort*, then $u_0$ returns 1 and Algorithm 1 finds a $k$-cycle (i.e., returns 1).*

**Proof of Theorem 2.** Suppose that $G$ is $\varepsilon$-far from being $C_k$-free. We have no nodes whose state is abort (as we said, the abort state will be used in Section 2.5). Drawing random colors and weights, the probability that $\mathcal{E}_U, \mathcal{E}_{Cyc}$ and $\mathcal{E}_{Col}$ all occur is at least $\frac{\varepsilon}{2k^k}$; therefore, the probability that we fail to detect a $k$-cycle after $\lceil 20k^k/\varepsilon \rceil$ attempts is at most $1/10$. ◀

## 2.4 Detecting Constant-Size Trees

In this section we show that for any constant-size tree $T$, we can test $T$-freeness *exactly* (that is, without the property-testing relaxation) in $O(1)$ rounds. Let the nodes of $T$ be $0, \ldots, k-1$. We arbitrarily assign node 0 to be the root of $T$, and orient the edges of the tree upwards toward node 0. Let $R$ be the depth of the tree, that is, the maximum number of hops from any leaf of $T$ to node 0. Finally, let $children(x)$ be the children of node $x$ in the tree.

In the algorithm, we map each node of the network graph $G$ onto a random node of $T$ by assigning it a random color from $[k]$. Then we check if there is a copy of $T$ in $G$ that was mapped "correctly", with each node receiving the color of the vertex in $T$ it corresponds to.

**Algorithm 1:** Procedure `ColorCodedBFS`, code for node $u$

```
root ← u
wgt ← min {w(u,v) | v ∈ N(u)}
for r = 0,...,k−1 do
    if color = r and state ≠ abort then send (wgt, root) to neighbors
    receive (w_1, r_1),...,(w_t, r_t) from neighbors
    if color = (r+1) mod k then
        i ← argmin {w_1,...,w_t}
        if w_i < wgt then
            root ← r_i, wgt ← w_i
    if r = k−1 and u ∈ {r_1,...,r_t} then return 1
return 0
```

Initially the state of each node is "open" if it is an inner node of $T$, and "closed" if it is a leaf. The algorithm has $R$ rounds, in each of which all nodes broadcast their state and their color to their neighbors. When a node with color $j$ hears "closed" messages from nodes with colors matching all the children of node $j$ in $T$, it changes its status to "closed". After $R$ rounds, if node 0's state is "closed", we reject.

Let $x \in \{0, \ldots, k-1\}$ be a node of $T$, let $T'$ be the sub-tree rooted at $x$, and let $G' = (U, E')$ be a subgraph of $G = (V, E)$ isomorphic to $T'$. We say that $G'$ is *properly colored* if there is an isomorphism $\varphi$ from $G'$ to $T'$, such that $color(u) = \varphi(u)$. (There may be more than one possible isomorphism from $G'$ to $T'$.)

▶ **Lemma 5.** *Let $u$ be a node with color $color(u) = x$, and let $T'$ be the sub-tree of $T$ rooted at $x$. Let $h_x$ be the height of $x$, that is, the length of the longest path from a leaf of $T'$ to $x$. Then at any time $t \geq h_x$ in the execution of Algorithm 2, we have $state_u(t) = \mathsf{closed}$ iff there is a subgraph $G'$ containing $u$, which is isomorphic to $T'$ and properly colored.*

▶ **Corollary 6.** *For any node $u \in V$, at time $h$ we have $\mathsf{state}_u(h) = \mathsf{closed}$ iff $u$ is the root of a properly-colored copy of $T$.*

▶ **Corollary 7.** *If $G$ contains a copy of $T$, then repeating Algorithm 2 yields an constant probability one sided-error algorithm for detecting a copy of $T$.*

**Proof.** Fix a subgraph $G'$ which is isomorphic to $T$. Each time we pick a random coloring, the probability that $G'$ is properly colored is at least $1/k^k$ (perhaps more, if there is more than one isomorphism mapping the nodes of $G'$ to $T$). By Corollary 6, if $G'$ is properly colored, the root of the tree will discover this and return 1. Therefore the probability that we fail $\lceil 10k^k \rceil$ times is at most $1/10$. ◀

## 2.5 Detecting Constant-Size Complex Graphs

In this section we define a class $\mathcal{H}$ of graphs, and give an algorithm for detecting those graphs in constant number of round (taking the size of the graph as a constant). The class $\mathcal{H}$ includes all graphs of size 5 except $K_5$ (see full paper).

**Algorithm 2:** Procedure `CheckTree`, code for node $u$

```
if children(color) = ∅ then
    state ← closed
else
    state ← open
missing ← children(color)
for r = 1, ..., R do
    send (color, state) to neighbors
    receive (c_1, s_1), ..., (c_t, s_t) from neighbors
    foreach i = 1, ..., t do
        if c_i ∈ missing and s_i = closed then
            missing ← missing \ {c_i}
            if missing = ∅ then state ← closed
if color = 0 and state = closed then
    return 1
else
    return 0
```

### Definition of the Class $\mathcal{H}$

The class $\mathcal{H}$ contains all graphs that have the following property: there exists an edge $(u, v)$ such that any cycle in the graph contain at least one of $u$ and $v$. Equivalently, the class $\mathcal{H}$ contains all connected graphs that can be constructed using the following procedure:

1. We start with two nodes, 0 and 1, with an edge between them
2. Add any number of cycles $C_1, \ldots, C_\ell$ using new nodes, such that:
   - Each cycle $C_i$ contains either node 0 or node 1 or both; and
   - With the exception of nodes $0, 1$, the cycles are node-disjoint.
3. Select a subset $R$ of the nodes added so far, and for each node $x$ selected, attach a tree $T_x$ rooted at $x$ using "fresh" nodes (that is, with the exception of node $x$, each tree $T_x$ that we attach is node-disjoint from the graph constructed so far, including trees $T_y$ added for other nodes $y \neq x$).
4. For each $x \in \{0, 1\}$, add edges $E_x$ between node $x$ and some subset of nodes added in the previous steps.

The class $\mathcal{H}$ includes all connected nodes over five vertices, except the clique $K_5$. In Figure 1 we show a labeling consistent with the construction above (once the "correct" edge to label as $(0, 1)$ is identified, the rest is easy to verify).

Our algorithm for testing $H$-freeness for $H \in \mathcal{H}$ combines the ideas from the previous sections. We begin by color-coding the nodes of $G$, mapping each node onto a random node of $H$. Next, we choose a random directed edge $(u_0, u_1)$ from among the edges mapped to $(0, 1)$, and begin the task of verifying that the various components of $H$ are present and attached properly.

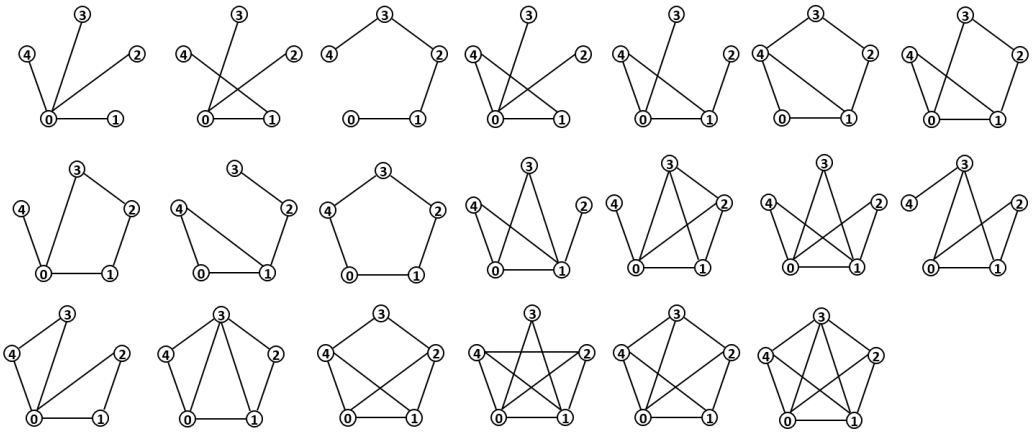

**Figure 1** A good labeling for each connected graph over five vertices, except $K_5$

For simplicity, below we describe the verification process assuming that we really do choose a unique random edge, and all nodes know what it is; however, we cannot really do this, so we substitute using random edge weights as in Section 2.3.

(I) Nodes $u_0$ and $u_1$ broadcast the chosen edge $(u_0, u_1)$ for $\mathrm{diam}(H)$ rounds.

(II) Any node whose color is 0 or 1 but which is not $u_0$ or $u_1$ (resp.) sets its state to abort.

(III) For each edge $\{b, x\} \in E_b$, where $b \in \{0, 1\}$, nodes colored $x$ verify that they have an edge to node $u_b$; if they do not, they set their state to abort.

(IV) For each tree $T_x$ added in stage 3 of the construction, we verify that a properly-colored copy of $T_x$ is present, by having nodes colored $x$ call Algorithm 2, with the colors replaced by the names of the nodes in $T_x$. We denote this by `CheckTree`$(T_x)$. If a node colored $x$ fails to detect a copy of $T_x$ for which it is the root, it sets its state to abort for the rest of the current attempt.

(V) For each $i = 1, \ldots, \ell$, we test for a properly-colored copy of $C_i$. We define the *owner* of $C_i$, denoted $owner(C_i)$, to be node 0 if $C_0$ contains 0, and otherwise node 1. To verify the presence of $C_i$, we call Algorithm 1, using the names of the nodes in $C_i$ as colors: instead of color 0 we use $owner(C_i)$, and the remaining colors are mapped to the other nodes of $C_i$ in order (in a arbitrary orientation of $C_i$). We denote this call by `ColorBFS`$(C_i)$. (As indicated in Alg. 1, nodes whose state is abort do not participate.)

(VI) If both $u_0$ and $u_1$ are not in state abort, $u_0$ rejects, otherwise it accepts. All other nodes accept.

The analysis and pseudo-code of this algorithm appear in the full paper.

## 2.6 Testing $K_s$-Freeness

In previous sections it was shown how to test $K_3$ and $K_4$ freeness in $O(1/\varepsilon)$ rounds of communication. In this section we describe how to test $K_s$-freeness for cliques of any constant size $s$, in a sublinear number of rounds. Moreover, we show that triangle-freeness can be tested in $O(1)$ rounds, with no dependence on $\varepsilon$, when $\min\left(\frac{n}{m}, m^{-1/3}\right) \leq \varepsilon \leq 1$. Finally, we show that if the maximal degree is bounded by $O((\varepsilon m)^{\frac{1}{s-2}})$ then $K_s$-freeness can be tested in $O(1)$ rounds, but due to lack of space, this appears in the full paper version only.

### 2.6.1 Algorithm Overview

The basic idea is the following simple observation: suppose that each node $u$ could learn the entire subgraph induced by $N(u)$, that is, node $u$ knew for any two $v_1, v_2 \in N(u)$ whether they are neighbors or not. Then $u$ could check if there is a set of $s$ neighbors in $N(u)$ that are all connected to each other, and thus know if it participates in an $s$-clique or not. How can we leverage this observation?

For nodes $u$ with high degree, we cannot afford to have $u$ learn the entire subgraph induced by $N(u)$, as this requires of $N(u)^2$ bits of information. But fortunately, if $G$ is $\varepsilon$-far from $K_s$-free, then there are many copies of $K_s$ that contain some fairly low-degree nodes, as observed in [20]:

▶ **Lemma 8 [20].** *Let $I(G)$ be the set of edges in some maximum set of edge-disjoint copies of $H$, and let $g(G) = \{(u,v) \mid d(u)d(v) \leq 2m|E(H)|/\varepsilon\}$. Then $|I(G) \cap g(G)| \geq \varepsilon m/(4|E(H)|)$.*

▶ Remark. [20] considers only subgraphs $H$ with 4 vertices and constant $\varepsilon$, but their proof works for any subgraph $H$ and any $0 < \varepsilon \leq 1$.

The focus in [20] is on *good edges*, which are edges satisfying the condition in Lemma 8, but here we need to focus on the *endpoints* of such edges. We call $u \in V$ a *good vertex* if its degree is at most $\sqrt{2m|E(H)|/\varepsilon}$, and we say that a copy of $H$ in $G$ is a *good copy* if it contains a good vertex. Since each copy of $H$ in $I(G)$ contributes at most $|E(H)|$ edges to $g(G)$,

▶ **Corollary 9.** *If $G$ is $\varepsilon$-far from $H$-free, then $G$ contains at least $\varepsilon m/(4|E(H)|^2)$ edge-disjoint good copies of $H$.*

Because there are many good edge-disjoint copies of $K_s$, we can *sparsify* the graph and still retain at least one good copy of $K_s$.

We partition $G$ into many edge-disjoint sparse subgraphs, by having each vertex $u$ choose for each neighbor $v \in N(u)$ a random color $color(v) \in \{1, \ldots, C(u)\}$, where the size of the color range, $C(u)$, will be fixed later. This induces a partition of $G$'s edges into $C(v)$ color classes; let $N_c(u)$ denote the set of neighbors $v \in N(u)$ with $color(v) = c$. The expected size of $N_c(u)$ is $d(v)/C(v)$.

With this partition in place, we begin by showing how to solve triangle-freeness in constant time, and then extend the algorithm to other cliques $K_s$ with $s > 3$.

### 2.6.2 Triangle-Freeness for $\varepsilon \in [\min\left\{m^{-1/3}, n/m\right\}, 1]$ in $O(1)$ rounds

Assume that $\varepsilon$ is not too small with respect to $n$ and $m$: $\varepsilon \geq \min\left\{m^{-1/3}, n/m\right\}$. Then we can improve the algorithm from Section 2.3 and test triangle-freeness in *constant* time that does not depend on $\varepsilon$.

To test triangle-freeness, we set $C(v) = \lceil d(v)/200 \rceil$. Each node chooses a random color for each neighbor from the range $\{1, \ldots, C(v)\}$. Then, we go through the color classes $c = 1, \ldots, C(v)$ *in parallel*, and for each color class $c$, we look for a triangle containing two edges from $N_c(u)$: let $N_c(u) = \{v_1, \ldots, v_{t_c}\}$. for $R = 202e^2$ rounds $r = 1, \ldots, R$, node $u$ sends $v_r$ to all neighbors $v_1, \ldots, v_r$ in $N_c(u)$, and each neighbor $v_i$ responds by telling $u$ whether it is also connected to $v_r$, that is, whether $v_r \in N(v_i)$ (note that we do not insist on the edge $(v_r, v_i)$ having color $c$). If $v_r \in N(v_i)$, then node $u$ has found a triangle, and it rejects. If after $202e^2$ attempts node $u$ has not found a triangle in any color class, it accepts.

▶ **Lemma 10.** *If $G$ is $\varepsilon$-far from being triangle-free, then with probability $2/3$, at least one vertex detects a triangle.*

**Proof.** Let $\mathcal{T}$ be a set of edge-disjoint good triangles in $G$, of size $|\mathcal{T}| \geq \varepsilon m/(4|E(T)|^2) = \varepsilon m/36$. By Corollary 9 we know that there is such a set.

Assume that $\mathcal{T} = \{T_1, \ldots, T_t\}$. By definition, each good triangle has a *good vertex*; let $v_i$ be a good vertex from the $i$'th triangle $T_i$.

For each $i \in \{1, \ldots, t\}$, let $A_i$ be the event that $v_i$ assigned the same color, $c_i$, to the other two vertices in $T_i$, and let $X_i$ be an indicator for $A_i$. We have $\Pr[X_i = 1] = 1/C(v_i) = 200/d(v_i)$. Also, since the triangles in $\mathcal{T}$ are edge-disjoint, $X_1, \ldots, X_t$ are independent. Now let $X = \sum_{i=1}^{t} X_i$ be their sum; then

$$\Pr[X = 0] = \Pr[\bigcap_{i=1}^{t}(X_i = 0)] = \prod_{i=1}^{t}\left(1 - \frac{1}{C(v_i)}\right) = \prod_{i=1}^{t}\left(1 - \frac{200}{d(v_i)}\right).$$

We divide into two cases:

I. $m < n^{3/2}$: then $\varepsilon \geq \min\left(\frac{n}{m}, m^{-1/3}\right) = m^{-1/3}$. Recall $v_i$ is a *good vertex*, which means $d(v_i) \leq \sqrt{6m/\varepsilon}$, and therefore

$$\prod_{i=1}^{t}\left(1 - \frac{200}{d(v_i)}\right) \leq \left(1 - \frac{200}{\sqrt{6m/\varepsilon}}\right)^t \leq e^{-\frac{200t}{\sqrt{6m/\varepsilon}}} \leq e^{-\frac{200\varepsilon m}{36\sqrt{6m/\varepsilon}}} = e^{-\frac{200\varepsilon^{3/2}\sqrt{m}}{36\sqrt{6}}} \leq e^{-2}.$$

II. $m \geq n^{3/2}$: then $\varepsilon \geq \min\left(\frac{n}{m}, m^{-1/3}\right) = \frac{n}{m}$. The degree of each vertex is no more than $n$, and hence

$$\prod_{i=1}^{t}\left(1 - \frac{200}{d(v_i)}\right) \leq \left(1 - \frac{200}{n}\right)^t \leq e^{-\frac{200t}{n}} \leq e^{-\frac{200\varepsilon m}{36n}} \leq e^{-2}.$$

So in any case we get $Pr[X = 0] \leq e^{-2}$.

Conditioned on $X \geq 1$, there is at least one vertex $v_i$ which put two of its triangle neighbors in the same color class $c_i$, which means that if $N_c(v_i)$ is no larger than $200e^2$, node $v_i$ will go through all neighbors in $N_c(v_i)$ and find the triangle. Because the colors of the edges are independent of each other, conditioning on $A_i$ does not change the expected size of $N_{c_i}(v_i)$ by much: we know that the other two vertices in $T_i$ received color $c_i$, but the remaining neighbors are assigned to a color class independently. The expected size of $|N_{c_i}(v_i)|$ is therefore $(d(v_i)-2)/C(v_i)+2 < 202 = R/e^2$, and by Markov, $\Pr[|N_{c_i}(v_i)| > R] \leq 1/e^2$.

To conclude, by union bound, the probability that no node $v_i$ has $X_i = 1$, or that the smallest node $v_i$ with $X_i = 1$ has $|N_c(v_j)| > 200e^2$ for the smallest color class $c$ containing two triangle neighbors, is at most $1/e^2 + 1/e^2 < 1/3$. ◀

### 2.6.3 General Tester for $K_s$-Freeness

Use the same algorithm but with a different setting of the parameters, we can test $K_s$-freeness for any $s \geq 3$.

▶ **Theorem 11.** *There is a 1-sided error distributed property-testing algorithm for $K_s$-freeness, for any constant $s \geq 4$, with running time $O(\varepsilon^{\frac{-s}{2(s-2)}} m^{\frac{s-4}{2(s-2)}})$.*

▶ **Corollary 12.** *There is a 1-sided error distributed property-testing algorithm for $K_5$-freeness, with running time $O(m^{1/6})$.*

We set

$$C(u) = \left\lceil \left( \frac{1}{2s^4} \varepsilon m \right)^{\frac{1}{s-2}} \right\rceil$$

to be the number of color classes at node $u$, and

$$R = 2s^4 e^2 \left[ \varepsilon^{-1/2-1/(s-2)} m^{1/2-1/(s-2)} + s - 1 \right]$$

to be the timeout. For $R$ rounds, each node $u$ sends the next node $v_r$ from each color class to all neighbors $v_1, \ldots, v_{t_c}$ in that color class, and each neighbor $v_i$ responds by telling $u$ whether $v_r$ is its neighbor or not. Node $u$ remembers this information; if at any point it knows of a subset $S \subseteq N_c(u)$ of $|S| = s$ nodes that are all neighbors of each other, then it has found an $s$-clique, and it rejects. After $R$ rounds $u$ gives up and accepts.

▶ **Lemma 13.** *If $G$ is $\varepsilon$-far from $K_s$-free, then with probability at least $2/3$, at least one vertex detects a copy of $K_s$.*

▶ Remark. For $s \geq 5$, the algorithm requires a linear estimate of $m$ to get good running time. If $m$ is unknown, then the vertices may run the algorithm $\log n$ times for exponentially-increasing guesses $m = [n, 2n, ...n^2]$ , and as the protocol has one sided error, correctness is maintained; however, the running time increases to $O(\varepsilon^{\frac{-s}{2(s-2)}} n^{\frac{s-4}{(s-2)}})$ rounds.

# 3 Note #2: Deterministic Tree Detection and Applications in Distributed Property Testing

## 3.1 Context and Objective

Given a fixed graph $H$ (e.g., a triangle, a clique on four nodes, etc.), a graph $G$ is $H$-free if it does not contain $H$ as a subgraph[2]. Detecting copies of $H$ or deciding $H$-freeness has been investigated in many algorithmic frameworks, including classical sequential computing [2], parametrized complexity [32], streaming [8], property-testing [3], communication complexity [27], quantum computing [5], etc. In the context of distributed network computing, deciding $H$-freeness refers to the task in which the processing nodes of a network $G$ must collectively detect whether $H$ is a subgraph of $G$, according to the following decision rule:

- if $G$ is $H$-free then every node outputs accept;
- otherwise, at least one node outputs reject.

In other words, $G$ is $H$-free if and only if all nodes output accept.

Recently, deciding $H$-freeness for various types of graph patterns $H$ has received lots of attention (see, e.g., [9, 10, 12, 13, 26, 19, 20]) in the CONGEST model [36], and in variants of this model. (Recall that the CONGEST model is a popular model for analyzing the impact of limited link bandwidth on the ability to solve tasks efficiently in the context of distributed network computing). In particular, it has been observed that deciding $H$-freeness may require nodes to consume a lot of bandwidth, even for very simple graph patterns $H$. For instance, it has been shown in [13] that deciding $C_4$-freeness requires $\widetilde{\Omega}(\sqrt{n})$ rounds in $n$-node networks in the CONGEST model. Intuitively, the reason why so many rounds of

[2] Recall that $H$ is a subgraph of $G$ if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

computation are required to decide $C_4$-freeness is that the limited bandwidth capacity of the links prevents every node with high degree from sending the entire list of its neighbors through one link, unless consuming a lot of rounds. The lower bound $\widetilde{\Omega}(\sqrt{n})$ rounds for $C_4$-freeness can be extended to larger cycles $C_k$, $k \geq 4$, obtaining a lower bound of $\Omega(\text{poly}(n))$ rounds, where the exponent of the polynomial in $n$ depends on $k$ [13]. Hence, not only "global" tasks such as minimum-weight spanning tree [11, 28, 34], diameter [1, 21], and all-pairs shortest paths [24, 30, 33] are bandwidth demanding, but also "local" tasks such as deciding $H$-freeness are bandwidth demanding, at least for some graph patterns $H$.

In this note, we focus on a generic set of $H$-freeness decision tasks which includes several instances deserving full interest on their own right. In particular, deciding $P_k$-freeness, where $P_k$ denotes the $k$-node path, is directly related to the NP-hard problem of computing the longest path in a graph. Also, detecting the presence of large complete binary trees, or of large binomial trees, is of interest for implementing classical techniques used in the design of efficient parallel algorithms (see, e.g., [29]). Similarly, detecting large Polytrees in a Bayesian network might be used to check fast belief propagation [35]. Finally, as it will be shown in this note, detecting the presence of various forms of trees can be used to tests the presence of graph patterns of interest in the framework of distributed property-testing [9]. Hence, this note addresses the following question:

> For which tree $T$ is it possible to decide $T$-freeness efficiently in the CONGEST model, that is, in a number of rounds independent from the size $n$ of the underlying network?

At a first glance, deciding $T$-freeness for some given tree $T$ may look simpler than detecting cycles, or even just deciding $C_4$-freeness. Indeed, the absence of cycles enables to ignore the issue of checking that a path starts and ends at the *same* node, which is bandwidth consuming because it requires maintaining all possible partial solutions corresponding to growing paths from all starting nodes. When detecting cycles, discarding even just a few starting nodes may result in missing the unique cycle including these nodes. However, even deciding $P_k$-freeness requires to overcome many obstacles. First, as mentioned before, finding a longest simple path in a graph is NP-hard, which implies that it is unlikely that an algorithm deciding $P_k$-freeness exists in the CONGEST model, with running time polynomial in $k$ at every node. Second, and more importantly, there exists potentially up to $\Theta(n^k)$ paths of length $k$ in a network, which makes impossible to maintain all of them in partial solutions, as the overall bandwidth of $n$-node networks is at most $O(n^2 \log n)$ in the CONGEST model.

## 3.2 Our Results

We show that, in contrast to $C_k$-freeness, $P_k$-freeness can be decided in a constant number of rounds, for any $k \geq 1$. In fact, our main result is far more general, as it applies to *any* tree. Stated informally, we prove the following:

**Theorem A.** *For every tree $T$, there exists a deterministic algorithm for* deciding *$T$-freeness performing in a constant number of rounds under the* CONGEST *model.*

For establishing Theorem A, we present a distributed implementation of a pruning technique based on a combinatorial result due to Erdős et al. [15] that roughly states the following. Let $k > t > 0$. For any set $V$ of $n$ elements, and any collection $F$ of subsets of $V$, all with cardinality at most $t$, let us define a *witness* of $F$ as a collection $\widehat{F} \subseteq F$ of subsets of $V$ such that, for any $X \subseteq V$ with $|X| \leq k - t$, the following holds:

$$\big(\exists Y \in F : Y \cap X = \emptyset\big) \implies \big(\exists \widehat{Y} \in \widehat{F} : \widehat{Y} \cap X = \emptyset\big).$$

Of course, every $F$ is a witness of itself. However, Erdős et al. have shown that, for every $k$, $t$, and $F$, there exists a *compact* witness $\widehat{F}$ of $F$, that is, a witness whose cardinality depends on $k$ and $t$ only, and hence is independent of $n$. To see why this result is important for detecting a tree $T$ in a network $G$, consider $V$ as the set of nodes of $G$, $k$ as the number of nodes in $T$, and $F$ as a collection of subtrees $Y$ of size at most $t$, each isomorphic to some subtree of $T$. The existence of compact witnesses allows an algorithm to keep track of only a small subset $\widehat{F}$ of $F$. Indeed, if $F$ contains a partial solution $Y$ that can be extended into a global solution isomorphic to $T$ using a set of nodes $X$, then there is a *representative* $\widehat{Y} \in \widehat{F}$ of the partial solution $Y \in F$ that can also be extended into a global solution isomorphic to $T$ using the same set $X$ of nodes. Therefore, there is no need to keep track of all partial solutions $Y \in F$, it is sufficient to keep track of just the partial solutions $\widehat{Y} \in \widehat{F}$. This pruning technique has been successfully used for designing fixed-parameter tractable (FPT) algorithms for the longest path problem [32], as well as, recently, for searching cycles in the context of distributed property-testing [19]. Using this technique for detecting the presence of a given tree however requires to push the recent results in [19] much further. First, the detection algorithm in [19] is anchored at a fixed node, i.e., the question addressed in [19] is whether there is a cycle $C_k$ passing through a given node. Instead, we address the detection problem in its full generality, and we do not restrict ourselves to detecting a copy of $T$ including some specific node. Second, detecting trees requires to handle partial solutions that are not only composed of sets of nodes, but that offer various shapes, depending on the structure of the tree $T$, representing all possible combinations of subtrees of $T$.

Theorem A, which establishes the existence of distributed algorithms for detecting the presence of trees, has important consequences on the ability to *test* the presence of more complex graph patterns in the context of *distributed property-testing*. Recall that, for $\varepsilon \in (0, 1)$, a graph $G$ is $\varepsilon$-far from being $H$-free if removing less than a fraction $\varepsilon$ of its edges cannot result in an $H$-free graph. A (randomized) distributed algorithm *tests* $H$-freeness if it decides $H$-freeness according to the following decision rule:

- if $G$ is $H$-free then Pr[every node outputs accept] $\geq 2/3$;
- if $G$ is $\varepsilon$-far from being $H$-free then Pr[at least one node outputs reject] $\geq 2/3$.

That is, a testing algorithm separates graphs that are $H$-free from graphs that are far from being $H$-free. So far, the only non-trivial graph patterns $H$ for which distributed algorithms testing $H$-freeness are known are:

- the complete graphs $K_3$ and $K_4$ (see [9, 20]), and
- the cycles $C_k$, $k \geq 3$ (see [19]).

Using our algorithm for detecting the presence of trees, we show the following (stated informally):

**Theorem B.** *For every graph pattern $H$ composed of an edge and a tree with arbitrary connections between them, there exists a (randomized) distributed algorithm for* testing *$H$-freeness performing in a constant number of rounds under the* CONGEST *model.*

At a first glance, the family of graph patterns $H$ composed of an edge and a tree with arbitrary connections between them (like, e.g., the graph depicted on the top-left corner of Fig. 2) may look quite specific and artificial. This is not the case. For instance, every cycle $C_k$ for $k \geq 3$ is a "tree plus one edge". This also holds for 4-node complete graph $K_4$. In fact, all known results about testing $H$-freeness for some graph $H$ in [9, 19, 20] are just direct consequence of Theorem B. Moreover, Theorem B enables to test the presence of other graph

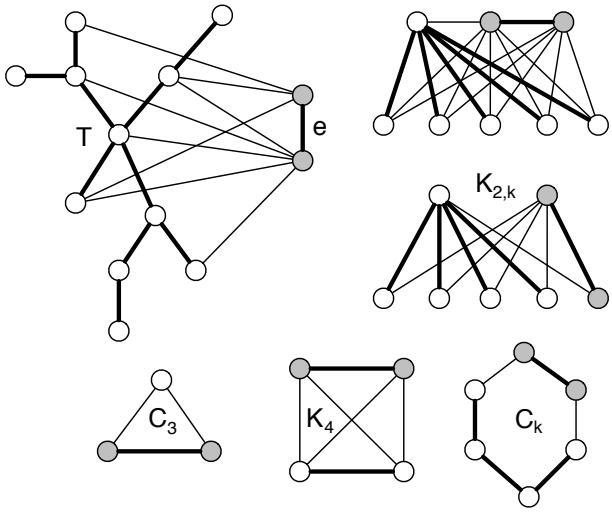


**Figure 2** All these graphs are composed of a tree $T$ and edge $e$ with arbitrary connections between them.

patterns, like the complete bipartite graph $K_{2,k}$ with $k+2$ nodes, for every $k \geq 1$, or the graph pattern depicted on the top-right corner of Fig. 2, in $O(1)$ rounds. It also enables to test the presence of connected 1-factors as a subgraph in $O(1)$ rounds. (Recall that a graph $H$ is a 1-factor if its edges can be directed so that every node has out-degree 1).

In fact, our algorithm is 1-sided, that is, if $G$ is $H$-free, then all nodes output accept with probability 1.

## 3.3 Detecting the Presence of Trees

In this section, we establish our main result:

▶ **Theorem 14.** *For every tree $T$, there exists an algorithm performing in $O(1)$ rounds in the* CONGEST *model for detecting whether the given input network contains $T$ as a subgraph.*

**Proof.** Let $k$ be the number of nodes in tree $T$. The nodes of $T$ are labeled arbitrarily by $k$ distinct integers in $[1,k]$. We arbitrarily choose a vertex $r \in [1,k]$ of $T$, and view $T$ as rooted in $r$. For any vertex $\ell \in V(T)$, let $T_\ell$ be the subtree of $T$ rooted in $\ell$. We say that $T_\ell$ is a *shape* of $T$. Our algorithm deciding $T$-freeness proceeds in $\text{depth}(T_r)+1$ rounds. At round $t$, every node $u$ of $G$ constructs, for each shape $T_\ell$ of depth at most $t$, a set of subtrees of $G$ all rooted at $u$, denoted by $\text{SOS}_u(T_\ell)$, such that each subtree in $\text{SOS}_u(T_\ell)$ is isomorphic to the shape $T_\ell$. The isomorphism is considered in the sense of rooted trees, i.e., it maps $u$ to $\ell$. If we were in the LOCAL model[3], we could afford to construct the set of all such subtrees of $G$. However, we cannot do that in the CONGEST model because there are too many such subtrees. Therefore, the algorithm acts in a way which guarantees that:

1. the set $\text{SOS}_u(T_\ell)$ is of constant size, for every node $u$ of $G$, and every node $\ell$ of $T$;

[3] The LOCAL model is similar to the CONGEST model, but it has no restriction on the size of the messages [36].

2. for every set $C \subseteq V$ of size at most $k - |V(T_\ell)|$, if there is some subtree $W$ of $G$ rooted at $u$ that is isomorphic to $T_\ell$, and that is not intersecting $C$, then $\text{SOS}_u(T_l)$ contains at least one such subtree $W'$ not intersecting $C$. (Note that $W'$ might be different from $W$).

The intuition for the second condition is the following. Assume that there exists some subtree $W$ of $G$ rooted at $u$, corresponding to some shape $T_\ell$, which can be extended into a subtree isomorphic to $T$ by adding the vertices of a set $C$. The algorithm may well not keep the subtree $W$ in $\text{SOS}_u(T_l)$. However, we systematically keep at least one subtree $W'$ of $G$, also rooted at $u$ and isomorphic to $T_\ell$, that is also extendable to $T$ by adding the vertices of $C$. Therefore the sets $\text{SOS}_u(T_\ell)$, over all shapes $T_\ell$ of depth at most $t$, are sufficient to ensure that the algorithm can detect a copy of $T$ in $G$, if it exists. Our approach is described in Algorithm 3. (Observe that, in this algorithm, if we omit Lines 13 to 15, which prune the set $\text{SOS}_u(T_\ell)$, we obtain a trivial algorithm detecting $T$ in the LOCAL model). Implementing the pruning of the sets $\text{SOS}_u(T_\ell)$ for keeping them compact, we make use of the following combinatorial lemma, which has been rediscovered several times, under various forms (see, e.g., [19, 32]).

▶ **Lemma 15 Erdős, Hajnal, Moon [15].** *Let $V$ be a set of size $n$, and consider two integer parameters $p$ and $q$. For any set $F \subseteq \mathcal{P}(V)$ of subsets of size at most $p$ of $V$, there exists a* compact $(p, q)$-representation *of $F$, i.e., a subset $\hat{F}$ of $F$ satisfying:*

1. *For each set $C \subseteq V$ of size at most $q$, if there is a set $L \in F$ such that $L \cap C = \emptyset$, then there also exists $\hat{L} \in \hat{F}$ such that $\hat{L} \cap C = \emptyset$;*
2. *The cardinality of $\hat{F}$ is at most $\binom{p+q}{p}$, for any $n \geq p + q$ .*

By Lemma 15, the sets $\text{SOS}_u(T_\ell)$ can be reduced to constant size (i.e., independent of $n$), for every shape $T_\ell$ and every node $u$ of $G$. Moreover, the number of shapes is at most $k$, and, for each shape $T_\ell$, each element of $\text{SOS}_u(T_\ell)$ can be encoded on $k \log n$ bits. Therefore each vertex communicates only $O(\log n)$ bits per round along each of its incident edges. So, the algorithm does perform in $O(1)$ rounds in the CONGEST model[4].

**Proof of correctness.** First, observe that if $\text{SOS}_u(T_\ell)$ contains a graph $W$, then $W$ is indeed a tree rooted at $u$, and isomorphic to $T_\ell$. This is indeed the case at round $t = 0$, and we can proceed by induction on $t$. Let $T_\ell$ be a shape of depth $\ell$. Each graph $W$ added to $\text{SOS}_u(T_\ell)$ is obtained by gluing vertex-disjoint trees at the root $u$. These latter trees are isomorphic to the shapes $T_{j_1}, \ldots, T_{j_s}$, where $j_1, \ldots, j_s$ are the children of node $j$ in $T$. Therefore $W$ is isomorphic to $T_\ell$. In particular, if the algorithm rejects at some node $u$, it means that there exists a subtree of $G$ isomorphic to $T$.

We now show that if $G$ contains a subgraph $W$ isomorphic to $T$, then the algorithm rejects in at least one node. For this purpose, we prove a stronger statement:

▶ **Lemma 16.** *Let $u$ be a node of $G$, $T_\ell$ be a shape of $T$, and $C$ be a subset of vertices of $G$, with $|C| \leq k - |V(T_u)|$. Let us assume that there exists a subgraph $W_u$ of $G$, satisfying the following two conditions: (1) $W_u$ is isomorphic to $T_\ell$, and the isomorphism maps $u$ on $\ell$, and (2) $W_u$ does not contain any vertex of $C$. Then $\text{SOS}_u(T_\ell)$ contains a tree $W'_u$ satisfying these two conditions.*

[4] We may assume that, for compacting a set $\text{SOS}_u(T_\ell)$ in Lines 13-15, every node $u$ applies Lemma 15 by brute force (e.g., by testing all candidates $\hat{F}$). In [32], an algorithmic version of Lemma 15 is proposed, producing a set $\hat{F}$ of size at most $\sum_{i=1}^{q} p^i$ in time $O((p+q)! \cdot n^3)$, i.e., in time poly$(n)$ for fixed $p$ and $q$.

**Algorithm 3:** Tree-detection, for a given tree $T$. Algorithm executed by node $u$.

1 **for** *each leaf* $\ell$ *of* $T$ **do**
2 let $\text{SOS}_u(T_\ell)$ be the unique tree with single vertex $u$
3 exchange the sets SOS with all neighbors
4 **for** $t = 1$ ***to*** $\text{depth}(T)$ **do**
5 **for** *each node* $\ell$ *of* $T$ *with* $\text{depth}(T_\ell) = t$ **do**
6 $\text{SOS}_u(T_\ell) \leftarrow \emptyset$
7 let $j_1, \ldots, j_s$ be the children of $\ell$ in $T$
8 **for** *every* $s$*-uple* $(v_1, \ldots, v_s)$ *of nodes in* $N(u)$ **do**
9 **for** *every* $(W_1, \ldots, W_s) \in \text{SOS}_{v_1}(T_{j_1}) \times \cdots \times \text{SOS}_{v_s}(T_{j_s})$ **do**
10 **if** $\{u\}$ *and* $W_1, \ldots, W_s$ *are pairwise disjoint* **then**
11 let $W$ be the tree with root $u$, and subtrees $W_1, \ldots, W_s$
12 add $W$ to $\text{SOS}_u(T_\ell)$ *// each* $W_i$ *is glued to* $u$ *by its root*
13 let $F = \{V(W) \mid W \in \text{SOS}_u(T_l)\}$
*// collection of vertex sets for trees in* $\text{SOS}_u(T_l)$
14 construct a $(|V(T_\ell)|, k - |V(T_\ell)|)$-compact representation $\hat{F} \subseteq F$
*// cf. Lemma 15*
15 remove from $\text{SOS}_u(T_\ell)$ all trees $W$ with vertex set not in $\hat{F}$
16 exchange $\text{SOS}_u(T_\ell)$ with all neighbors
17 **if** $\text{SOS}_u(T_r) = \emptyset$ **then** *// r denotes the root of T*
18
19 **return** *accept*
20 **else**
21 **return** *reject*

We prove the lemma by induction on the depth of $T_\ell$. If $\text{depth}(T_\ell) = 0$ then $\ell$ is a leaf of $T_\ell$, and $\text{SOS}_u(T_\ell)$ just contains the tree formed by the unique vertex $u$. Il particular, it satisfies the claim. Assume now that the claim is true for any node of $T$ whose subtree has depth at most $t-1$, and let $\ell$ be a node of depth $t$. Let $j_1, \ldots, j_s$ be the children of $\ell$ in $T$. For every $i$, $1 \leq i \leq s$, let $v_i$ be the vertex of $W_u$ mapped on $j_i$. By induction hypothesis, $\text{SOS}_{v_1}(T_{j_1})$ contains some tree $W'_{v_1}$ isomorphic to $T_{j_1}$ and avoiding the nodes in $C \cup \{u\}$, as well as all the nodes of $W_{v_2}, \ldots W_{v_s}$. Using the same arguments, we proceed by increasing values of $i = 2, \ldots, s$, and we choose a tree $W'_{v_i} \in \text{SOS}_{v_i}(T_{j_i})$ isomorphic to $T_{j_i}$ that avoids $C \cup \{u\}$, as well as all the nodes in $W'_{v_1}, \ldots, W'_{v_{i-1}}$ and the nodes of $W_{v_{i+1}}, \ldots, W_{v_s}$. Now, observe that the tree $W''$ obtained from gluing $u$ to $W'_{v_1}, \ldots, W'_{v_s}$ has been added to $\text{SOS}_u(T_\ell)$ before compacting this set, by Line 11 of Algorithm 3. Since $W''$ does not intersect $C$, we get that, by compacting the set $\text{SOS}_u(T_\ell)$ using Lemma 15, the algorithm keeps a representative subtree $W'$ of $G$ that is isomorphic to $T_l$ and not intersecting $C$. This completes the proof of the lemma. ◇

To complete the proof of Theorem 14, let us assume there exists a subtree $W$ of $G$ isomorphic to $T$, and let $u$ be the vertex that is mapped to the root $r$ of $T$ by this isomorphism. By Lemma 16, $\text{SOS}_u(T_r) \neq \emptyset$, and thus the algorithm rejects at node $u$. ◀

## 3.4 Distributed Property Testing

In this section, we show how to construct a distributed tester for $H$-freeness in the sparse model, based on Algorithm 3. This tester is able to test the presence of every graph pattern $H$ composed of an edge $e$ and a tree $T$ connected in an arbitrary manner, by distinguishing graphs that include $H$ from graphs that are $\varepsilon$-far[5] from being $H$-free.

Specifically, we consider the set $\mathcal{H}$ of all graph patterns $H$ with node-set $V(H) = \{x, y, z_1, \ldots, z_k\}$ for $k \geq 1$, and edge-set $E(H) = \{f\} \cup E(T) \cup \mathcal{E}$, where $f = \{x, y\}$, $T$ is a tree with node set $\{z_1, \ldots, z_k\}$, and $\mathcal{E}$ is some set of edges with one end-point equal to $x$ or $y$, and the other end-point $z_i$ for $i \in \{1, \ldots, k\}$. Hence, a graph $H \in \mathcal{H}$ can be described by a triple $(f, T, \mathcal{E})$ where $\mathcal{E}$ is a set of edges connecting a node in $T$ with a node in $f$.

We now establish our second main result, i.e., Theorem B, stated formally below as follows:

▶ **Theorem 17.** *For every graph pattern $H \in \mathcal{H}$, i.e., composed of an edge and a tree connected in an arbitrary manner, there exists a randomized 1-sided error distributed property testing algorithm for $H$-freeness performing in $O(1/\varepsilon)$ rounds in the* CONGEST *model.*

**Proof.** Let $H = (f, T, \mathcal{E})$, with $f = \{x, y\}$. Let us assume that there are $\nu$ copies of $H$ in $G$, and let us call these copies $H_1 = (f_1, T_1, \mathcal{E}_1), \ldots, H_\nu = (f_\nu, T_\nu, \mathcal{E}_\nu))$. Let $\boldsymbol{E} = \{f_1, \ldots, f_\nu\}$. Our tester algorithm for $H$-freeness is composed by the following two phases:

1. determine a candidate edge $e$ susceptible to belong to $\boldsymbol{E}$;
2. checking the existence of a tree $T$ connected to $e$ in the desired way.

In order to find the candidate edge, we exploit the following lemma:

▶ **Lemma 18 [20].** *Let $H$ be any graph. Let $G$ be an $m$-edge graph that is $\varepsilon$-far from being $H$-free. Then $G$ contains at least $\varepsilon m/|E(H)|$ edge-disjoint copies of $H$.*

Hence, if the actual $m$-edge graph $G$ is $\varepsilon$-far from being $H$-free, we have $|\boldsymbol{E}| \geq \varepsilon m/|E(H)|$. Thus, by randomly choosing an edge $e$ and applying Lemma 18, $e \in \boldsymbol{E}$ with probability at least $\varepsilon/|E(H)|$.

As shown in [19], the first phase can be computed in the following way. First, every edge is assigned to the endpoint having the smallest identifier. Then, every node picks a random integer $r(e) \in [1, m^2]$ for each edge $e$ assigned to it. The candidate edge of Phase 1 is the edge $e_{min}$ with minimum rank, and indeed $\Pr[e_{min} \in \boldsymbol{E}] \geq \varepsilon/|E(H)|$.

It might be the case that $e_{min}$ is not unique though. However: $\Pr[e_{min} \text{ is unique}] \geq 1/e^2$ where $e$ denotes here the basis of the natural logarithm. Also, every node picks, for every edge $e = \{v_1, v_2\}$ assigned to it, a random bit $b$. Assume, w.l.o.g., that $\mathrm{ID}(v_1) < \mathrm{ID}(v_2)$. If $b = 0$, then the algorithm will start Phase 2 for testing the presence of $H$ with $(x, y) = (v_1, v_2)$, and if $b = 1$, then the algorithm will start Phase 2 for testing the presence of $H$ with $(x, y) = (v_2, v_1)$. We have $\Pr[e_{min} \text{ is considered in the right order}] \geq 1/2$. It follows that the probability $e_{min}$ is unique, considered in the right order, and part of $\boldsymbol{E}$ is at least $\frac{\varepsilon}{2|E(H)|e^2}$.

Using a deterministic search based on Algorithm 3, $H$ will be found with probability at least $\frac{\varepsilon}{2|E(H)|e^2}$. To boost the probability of detecting $H$ in a graph that is $\varepsilon$-far from being $H$-free, we repeat the search $2e^2|E(H)| \ln 3/\varepsilon$ times. In this way, the probability that $H$ is detected in at least one search is at least $2/3$ as desired.

[5] For $\varepsilon \in (0, 1)$, a graph $G$ is $\varepsilon$-far from being $H$-free if removing less than a fraction $\varepsilon$ of its edges cannot result in an $H$-free graph.

During the second phase, the ideal scenario would be that all the nodes of $G$ search for $H = (f, T, \mathcal{E})$ by considering only the edge $e_{min}$ as candidate for $f$, to avoid congestion. Obviously, making all nodes aware of $e_{min}$ would require diameter time. However, there is no needs to do so. Indeed, the tree-detection algorithm used in the proof of Theorem 14 runs in $depth(T)$ rounds. Hence, since only the nodes at distance at most $depth(T)+1$ from the endpoints of $e_{min}$ are able to detect $T$, it is enough to broadcast $e_{min}$ at distance up to $2\,(depth(T)+1)$ rounds. This guarantees that all nodes participating to the execution of the algorithm for $e_{min}$ will see the same messages, and will perform the same operations that they would perform by executing the algorithm for $e_{min}$ on the full graph. So, every node broadcasts its candidate edge with the minimum rank, at distance $2\,(depth(T)+1)$. Two contending broadcasts, for two candidate edges $e$ and $e'$ for $f$, resolve contention by discarding the broadcast corresponding to the edge $e$ or $e'$ with largest rank. (If $e$ and $e'$ have the same rank, then both broadcast are discarded). After this is done, every node is assigned to one specific candidate edge, and starts searching for $T$. Similarly to the broadcast phase, two contending searches, for two candidate edges $e$ and $e'$, resolve contention by aborting the search corresponding to the edge $e$ or $e'$ with largest rank. From now on, one can assume that a single search in running, for the candidate edge $e_{min}$.

It remains to show how to adapt Algorithm 3 for checking the presence of a tree $T$ connected to a *fixed* edge $e = \{x, y\} \in E(G)$ as specified in $\mathcal{E}$. Let us consider Instruction 5 of Algorithm 3, that is: "**for** each node $\ell$ of $T$ with $\text{depth}(T_\ell) = t$ **do**". At each step of this for-loop, node $u$ tries to construct a tree $W$ that is isomorphic to the subtree of $T$ rooted at $\ell$. In order for $u$ to add $W$ to $\text{SOS}_u(T_\ell)$, we add the condition that:

- if $\{\ell, x\} \in E(H)$ then $\{u, x\} \in E(G)$, and
- if $\{\ell, y\} \in E(H)$ then $\{u, y\} \in E(G)$.

Note that this condition can be checked by every node $u$. If this condition is not satisfied, then $u$ sets $\text{SOS}_u(T_\ell) = \emptyset$.

This modification enables to test $H$-freeness. Indeed, if the actual graph $G$ is $H$-free, then, since at each step of the modified algorithm, the set $\text{SOS}_u(T_\ell)$ is a subset of the set $\text{SOS}_u(T_\ell)$ generated by the original algorithm, the acceptance of the modified algorithm is guaranteed from the correctness of the original algorithm.

Conversely, let us show that, in a graph $G$ that is $\varepsilon$ far of being $H$-free, the algorithm rejects $G$ as desired. In the first phase of the algorithm, it holds that $e_{min} \in \boldsymbol{E}$ happens in at least one search whenever $G$ is $\varepsilon$-far from being $H$-free, with probability at least $2/3$. Following the same reasoning of the proof of Lemma 16, since the images of the isomorphism satisfy the condition of being linked to nodes $\{x, y\}$ in the desired way, the node of $G$ that is mapped to the root of $T$ correctly detects $T$, and rejects, as desired. ◀

## 3.5 Conclusion

In this note, we have proposed a generic construction for designing deterministic distributed algorithms detecting the presence of any given tree $T$ as a subgraph of the input network, performing in a constant number of rounds in the CONGEST model. Therefore, there is a clear dichotomy between cycles and trees, as far as efficiently solving $H$-freeness is concerned: while every cycle of at least four nodes requires at least a polynomial number of rounds to be detected, every tree can be detected in a constant number of rounds. It is not clear whether one can provide a simple characterization of the graph patterns $H$ for which $H$-freeness can be decided in $O(1)$ rounds in the CONGEST model. Indeed, the lower bound $\widetilde{\Omega}(\sqrt{n})$ for $C_4$-freeness can be extended to some graph patterns containing $C_4$ as induced subgraphs.

However, the proof does not seem to be easily extendable to all such graph patterns as, in particular, the patterns containing many overlapping $C_4$ like, e.g., the 3-dimensional hypercube $Q_3$, since this case seems to require non-trivial extensions of the proof techniques in [13]. An intriguing question is to determine the round-complexity of deciding $K_k$-freeness in the CONGEST model for $k \geq 3$, and in particular to determine the exact round-complexity of deciding $C_3$-freeness.

Our construction also provides randomized algorithms for testing $H$-freeness (i.e., for distinguishing $H$-free graphs from graphs that are far from being $H$-free), for every graph pattern $H$ that can be decomposed into an edge and a tree, with arbitrary connections between them, also running in $O(1)$ rounds in the CONGEST model. This generalizes the results in [9, 19, 20], where algorithms for testing $K_3$, $K_4$, and $C_k$-freeness for every $k \geq 3$ were provided. Interestingly, $K_5$ is the smallest graph pattern $H$ for which it is not known whether testing $H$-freeness can be done in $O(1)$ rounds, and this is also the smallest graph pattern that cannot be decomposed into a tree plus an edge. We do not know whether this is just coincidental or not.

# 4 Note #3: Algorithms for Testing and Correcting Graph Properties in the CONGEST Model

## 4.1 Computational Models

**Notations.** Let $G = (V, E)$ denote a graph, were $V$ is the set of vertices and $E$ is the set of edges. Let $n \triangleq |V(G)|$, and let $m \triangleq |E(G)|$. For every $v \in V$, let $N_G(v) \triangleq \{u \in V \mid \{u, v\} \in E\}$ denote the neighborhood of $v$ in $G$. For every $v \in V$, let $d_G(v) \triangleq |N_G(v)|$ denote the degree of $v$. When the graph at hand is clear from the context we omit the subscript $G$.

### 4.1.1 Distributed CONGEST Model

Computation in the distributed CONGEST [36] model is done as follows. Let $G = (V, E)$ denote a network where each vertex is a processor and each edge is a communication link between its two endpoints. Each processor is given a local input. Each processor $v$ has a distinct ID - for brevity we say that the ID of processor $v$ is simply $v$.[6] The computation is synchronized and is measured in terms of *rounds*. In each round, each processor performs the following steps: (1) Receive the messages that were sent by its neighbors in the previous round. (2) Execute a local (randomized) computation. (3) Sends (different) messages of $O(\log n)$ bits to every neighbor neighbors (or a possible "empty message"). In the last round all the processors stop and output a local output.

### 4.1.2 (Global) Testing Model

Graph property testing [22, 23] is defined as follows. A graph property $\mathcal{P}$ is a subset of all (undirected and unlabeled) graphs e.g., the graph is cycle-free, the graph is bipartite, etc. We focus on edge monotone (with respect to deletions) properties.

▶ **Definition 19.** A graph property $\mathcal{P}$ is *edge-monotone* if $G \in \mathcal{P}$ and $G'$ is obtained from $G$ by the removal of edges, then $G' \in \mathcal{P}$.

[6] In this paper we focus on randomized algorithms. Note that, with high probability, distinct IDs can be randomly generated using $O(\log n)$ bits.

We define the *edge-distance* between two graphs $G = (V, E)$ and $G' = (V, E')$ as the number edges in the symmetric difference $E \triangle E'$. We say that a graph $G$ is *$\varepsilon$-far (in the general model)* from having the property $\mathcal{P}$ if $|E \triangle E'| \geq \varepsilon \cdot |E|$, for every $G' = (V, E') \in \mathcal{P}$.

The tester accesses the graph via queries. The type of queries we consider are: (1) what is the degree of $v$ for $v \in V$? (2) who is the $i$th neighbor of $v \in V$?

We say that an algorithm is a *one sided $\varepsilon$-tester for property $\mathcal{P}$ in the general model* if given query access to the graph $G$ the algorithm ACCEPTS the graph $G$ if $G$ has the property $\mathcal{P}$, i.e, completeness, and REJECTS the graph $G$ with probability at least $2/3$ if $G$ is $\varepsilon$-far from having the property $\mathcal{P}$, i.e., soundness.

We note that since the tester must accept graphs $G \in \mathcal{P}$, a reject occurs if only if the tester has a proof that that $G \notin \mathcal{P}$. Such a proof is called a *witness* against $G \in \mathcal{P}$. In fact, in [9], it is required that the witness is an induced proper subgraph of $G$.

The complexity measure of this model is the number of queries made to $G$. The goal is to design an $\varepsilon$-tester with as few as possible queries. In particular, the number of queries should be sublinear in the size of the graph.

In Section 4.4 an additional query type is allowed; this query is called a *random edge query*, and enables on to pick an edge $e$ u.a.r. from $E$.

### 4.1.3 Distributed Testing in the CONGEST model

Let $G = (V, E)$ be a graph and let $\mathcal{P}$ denote a graph property. We say that a randomized distributed CONGEST algorithm is an *$\varepsilon$-tester for property $\mathcal{P}$ in the general model* [9] if when $G$ has the property $\mathcal{P}$ then *all* the processors $v \in V$ output ACCEPT, and if $G$ is $\varepsilon$-far from having the property $\mathcal{P}$, then there is a processor $v \in V$ that outputs REJECT with probability at least $2/3$.

### 4.1.4 Distributed Correcting

In this section we define correction in the distributed setting. We then explain how to obtain correction for the property of cycle-freeness. We focus here on edge-monotone properties, and therefore, consider only corrections that delete edges. One can view an $\varepsilon$-corrector as an approximation algorithm to the distance to property $\mathcal{P}$, where the approximation is additive.

▶ **Definition 20.** In the distributed CONGEST model, we say that an algorithm is an $\varepsilon$-corrector for an edge-monotone property $\mathcal{P}$ if the following holds.

1. Let $G = (V, E)$ denote the network's graph. When the algorithm terminates, each processor $v$ knows which edges in $E$ that intersect with $v$ are in the set of *deleted edges* $E' \subseteq E$.
2. $G(V, E \setminus E')$ is in $\mathcal{P}$.
3. $|E'| \leq dist(G, \mathcal{P}) + \varepsilon|E|$, where $dist(G, \mathcal{P})$ denotes the minimum number of edges that should be removed from $G$ in order to obtain the property $\mathcal{P}$.

## 4.2 Reducing the Dependency on the Diameter and Applications

In this section we present a general technique that reduces the dependency of the round complexity on the diameter. The technique is based on graph decompositions defined below.

▶ **Definition 21 [31].** Let $G = (V, E)$ denote an undirected graph. A $(\beta, d)$-decomposition of $G$ is a partition of $V$ into disjoint subsets $V_1, \ldots, V_k$ such that (i) For all $1 \leq i \leq k$,

$\mathrm{diam}(G[V_i]) \leq d$, where $G[V_i]$ is the vertex induced subgraph of of $G$ that is induced by $V_i$. (ii) The number of edges with endpoints belonging to different subsets is at most $\beta \cdot |E|$. We refer to these as *cut-edges* of the decomposition.

Note that the diameter constraint refers to strong diameter, in particular, each induced subgraph $G[V_i]$ must be connected.

Algorithms for $(\varepsilon, (\log n)/\varepsilon)$-decompositions were developed in many contexts (e.g., parallel algorithms [6, 7, 31]). An implementation in the CONGEST-model is presented in [14].

▶ **Theorem 22 [14].** *A $(\varepsilon, O(\log n/\varepsilon))$-decomposition can be computed in the randomized* CONGEST*-model in $O((\log n)/\varepsilon)$ rounds with probability at least $1 - 1/\mathrm{Poly}(n)$.*

A nice feature of the algorithm based on random exponential shifts is that at the end of the algorithm, there is a spanning BFS-like rooted tree $T_i$ for each subset $V_i$ in the decomposition. Moreover, each vertex $v \in V_i$ knows the center of $T_i$ as well as its parent in $T_i$. In addition, every vertex knows which of the edges incident to it are cut-edges.

The following definition captures the notion of connected witnesses against a graph satisfying a property.

▶ **Definition 23 [9].** [7] A graph property $\mathcal{P}$ is *non-disjointed* if for every witness $G'$ against $G \in \mathcal{P}$, there exists an induced subgraph $G''$ of $G'$ that is connected such that $G''$ is also a witness against $G \in \mathcal{P}$.

The main result of this section is formulated in the following theorem. We refer to a distributed algorithm in which all vertices accept iff $G \in \mathcal{P}$ as a *verifier* for $\mathcal{P}$.

▶ **Theorem 24.** *Let $\mathcal{P}$ be an edge-monotone non-disjointed graph property that can be verified in the* CONGEST*-model in $O(\mathrm{diam}(G))$ rounds, where $G$ is the input graph. Then there is an $\varepsilon$-tester for $\mathcal{P}$ in the randomized* CONGEST*-model with $O((\log n)/\varepsilon)$ rounds.*

**Proof.** The algorithm tries to "fix" the input graph $G$ so that it satisfies $\mathcal{P}$ by removing less than $\varepsilon \cdot m$ edges. The algorithm consists of two phases. In the first phase, an $(\varepsilon', O((\log n)/\varepsilon'))$ decomposition is computed in $O((\log n)/\varepsilon')$ rounds, for $\varepsilon' = \varepsilon/2$. The algorithm removes all the cut-edges of the decomposition. (There are at most $\varepsilon \cdot m/2$ such edges.) In the second phase, in each subgraph $G[V_i]$, an independent execution of the verifier algorithm for $\mathcal{P}$ is executed. The number of rounds of the verifier in $G[V_i]$ is $O(\mathrm{diam}(G[V_i])) = O((\log n)/\varepsilon)$.

We first prove completeness. Assume that $G \in \mathcal{P}$. Since $\mathcal{P}$ is an edge-monotone property, the deletion of the cut-edges does not introduce a witness against $\mathcal{P}$. This implies that each induced subgraph $G[V_i]$ does not contain a witness against $\mathcal{P}$, and hence the verifiers do not reject, and every vertex accepts.

We now prove soundness. If $G$ is $\varepsilon$-far from $\mathcal{P}$, then after the removal of the cut-edges (at most $\varepsilon m/2$ edges) property $\mathcal{P}$ is still not satisfied. Let $G'$ be a witness against the remaining graph satisfying $\mathcal{P}$. Since property $\mathcal{P}$ is non-disjointed, there exists a connected witness $G''$ in the remaining graph. This witness is contained in one of the subgraphs $G[V_i]$, and therefore, the verifier that is executed in $G[V_i]$ will reject, hence at least one vertex rejects, as required. ◀

[7] An alternative (nonequivalent) definition which suffices for proving Theorem 24 is as follows. A property $\mathcal{P}$ is non-disjointed if, for every nonconnected graph $G$, the following holds:

$$G \in \mathcal{P} \iff \text{for every connected component } G' \text{ of } G\text{: } G' \in \mathcal{P}.$$

We remark that if the round complexity of the verifier is $f(\text{diam}(G), n)$ (e.g., $f(\Delta, n) = \Delta + \log n$), then the round complexity of the $\varepsilon$-tester is $O((\log n)/\varepsilon) + f((\log n)/\varepsilon, n)$. This follows directly from the proof.

#### Extensions to $\varepsilon$-Testers

The following "bootstrapping" technique can be applied. If there exists an $\varepsilon$-tester in the CONGEST-model with round complexity $O(\text{diam}(G))$, then there exists an $\varepsilon$-tester with round complexity $O((\log n)/\varepsilon)$. The proof is along the same lines, expect that instead of a verifier, an $\varepsilon'$-tester is executed in each subgraph $G[V_i]$. Indeed, if $G$ is $\varepsilon$-far from $\mathcal{P}$, then there must exist a subset $V_i$ such that $G[V_i]$ is $\varepsilon'$-far from $\mathcal{P}$. Otherwise, we could "fix" all the parts by deleting at most $\varepsilon' \cdot m$ edges, and thus "fix" $G$ by deleting at most $2\varepsilon' \cdot m = \varepsilon m$ edges, a contradiction.

### 4.2.1 Testing Bipartiteness

Theorem 24 can be used to test whether a graph is bipartite or $\varepsilon$-far from being bipartite. A verifier for bipartiteness can be obtained by attempting to 2-color the vertices (e.g., BFS that assigns alternating colors to layers). In our special case, each subgraph $G[V_i]$ has a root which is the only vertex that initiates the BFS. In the general case, one would need to deal with "collisions" between searches, and how one search "kills" the other searches initiated by vertices of lower ID.

### 4.2.2 Testing Cycle-Freeness

Theorem 24 can be used to test whether a graph is acyclic or $\varepsilon$-far from being acyclic. As in the case of bipartiteness, any scan (e.g., DFS, BFS) can be applied. A second visit to a vertex indicates a cycle, in which case the vertex rejects.

▶ **Corollary 25.** *There exists an $\varepsilon$-tester in the randomized* CONGEST*-model for bipartiteness and cycle-freeness with round complexity $O((\log n)/\varepsilon)$.*

### 4.2.3 Corrector for Cycle-Freeness

Our $\varepsilon$-testers for testing cycle freeness can be easily converted into $\varepsilon$-correctors as follows:: (1) All the cut-edges are removed. (2) In each $G[V_i]$, all the edges which are not in the BFS-like spanning tree $T_i$ are removed.

▶ **Theorem 26.** *There exists an $\varepsilon$-corrector for cycle-freeness in the randomized* CONGEST*-model with round complexity $O((\log n)/\varepsilon)$.*

**Proof sketch.** The remaining edges form a forest of disjoint trees, and are therefore acyclic. The proof that the number of deleted edges is at most $\text{dist}(G, \mathcal{P}) + \varepsilon \cdot |\mathcal{E}|$ is based on the following two observations. Let $G'$ denote the graph obtained from $G$ by deleting all the cut-edges. $\text{dist}(G', \mathcal{P}) \leq \text{dist}(G, \mathcal{P})$ and $\text{dist}(G'[V_i], \mathcal{P}) = |E(G[V_i]) \setminus E(T_i)|$. ◀

## 4.3 Testing $H$-Freeness in $\Theta(1/\varepsilon)$ Rounds for $|V(H)| \leq 4$

### 4.3.1 Testing Triangle-Freeness

In this section we present an $\varepsilon$-tester for triangle-freeness that works in the CONGEST-model. The number of rounds is $O(1/\varepsilon)$.

Consider a triangle $ABC$ in the input graph $G = (V, E)$. This triangle can be detected if $A$ tells $B$ about a neighbor $C \in N(A)$ with the hope that $C$ is also a neighbor of $B$. Vertex $B$ checks that $C$ is also its neighbor, and if it is then the triangle $ABC$ is detected. Hence, $A$ would like to send to $B$ the name of a vertex $C$ such that $C \in N(A) \cap N(B)$. Since $A$ can discover $N(A)$ in a single round, it proceeds by telling $B$ about a neighbor $C \in N(A) \setminus \{B\}$ chosen uniformly at random. Let $M_{A\to B}$ denote the random neighbor that $A$ reports to $B$. A listing of the distributed $\varepsilon$-tester for triangle-freeness appears as Algorithm 4. Note that all the messages $\{M_{A\to B}\}_{(A,B)\in E}$ are independent, and that the messages are rechosen for each iteration.

▶ Claim 27. For every triangle $x$, the probability that triangle $x$ is detected is at least $1/m$.

**Proof.** Label the vertices of $x$ arbitrarily by $A, B, C$. The event that triangle $x$ is detected is contained in the event that $M_{A\to B} \in N(B)$. Since $ABC$ is a triangle, $C \in N(A) \cap N(B)$, and $\Pr[M_{A\to B} \in N(B)] \geq 1/d(A) \geq 1/m$. ◀

▶ Claim 28. If a graph $G$ is $\varepsilon$-far from being triangle-free, then it contains at least $\varepsilon \cdot m/3$ edge-disjoint triangles.

**Proof.** Consider the following procedure for "covering" all the triangles: while the graph contains a triangle, delete all three edges of the triangle. When the procedure ends, the remaining graph is triangle-free, hence at least $\varepsilon m$ edges were removed. The set of deleted triangles is edge disjoint and hence contains at least $\varepsilon m/3$ triangles. ◀

▶ **Theorem 29.** *Algorithm 4 is an $\varepsilon$-tester for triangle-freeness.*

**Proof.** *Completeness:* If $G$ is triangle free then Line 4 is never satisfied, hence for every $v$ Algorithm 4 terminates at Line 5.

*Soundness:* Let $G = (V, E)$ be a graph which is $\varepsilon$-far from being triangle free. By Claim 28 there are $\varepsilon \cdot m/3$ edge disjoint triangles in $G$. Edge disjointness implies that the detection of these triangles are independent events[8]. Hence, the probability of not detecting any of these triangles in a single iteration is at most $(1 - 1/m)^{\varepsilon m/3}$. The reject probability is amplified to $2/3$ by setting the number of iterations to be $t = \Theta(1/\varepsilon)$. ◀

**Algorithm 4:** Triangle-free-test($v$)

```
1 Send v to all u ∈ N(v)                    // 1st round: each v learns N(v)
2 for t ≜ Θ(1/ε) times do
3     For all u ∈ N(v), simultaneously: send u the message M_{v→u} ~ U(N(v) \ {u}).
4     If ∃w ∈ N(v) such that M_{w→v} ∈ N(v) then return REJECT
5 return ACCEPT
```

### 4.3.2 Testing $C_4$-Freeness in $\Theta(1/\varepsilon)$ Rounds

In this section we present an $\varepsilon$-tester in the CONGEST-model for $C_4$-freeness that runs in $O(1/\varepsilon)$ rounds.

[8] In fact, the events are independent even for triangles which are not edge disjoint.

### Uniform Sampling of 2-paths

Let $P_2(v)$ denote the set of all paths of length 2 that start at $v$. The algorithm is based on the ability of each vertex $v$ to uniformly sample a path from $P_2(v)$. How many paths in $P_2(v)$ start with the edge $(v, w)$? Clearly, there are $(d(w) - 1)$ such paths. Hence the first edge should be chosen according to the degree distribution over $N(v)$ defined by $\pi^v(w) \triangleq (d(w) - 1)/\sum_{x \in N(v)}(d(x) - 1)$. Moreover, for each $x \in N(w) \setminus \{v\}$, the (directed) edge $(w, x)$ appears exactly once as the second edge of a path in $P_2(v)$. Hence, given the first edge, the second edge is chosen uniformly.

This implies that $v$ can pick a random path $p \in P_2(v)$ as follows: (1) Each neighbor $w \in N(v)$ sends $v$ its degree and a uniformly randomly chosen neighbor $B_v(w) \in N(w) \setminus \{v\}$. The edge $(w, B_v(w))$ is a candidate edge for the second edge of $p$. (2) $v$ picks a neighbor $A(v) \in N(v)$ where $A(v) \sim \pi^v$. The random path $p$ is $p = \langle v, A(v), B_v(A(v))\rangle$, and it is uniformly distributed over $P_2(v)$.

In the algorithm, vertex $v$ reports a path to each neighbor. We denote by $p_u(v)$ the path in $P_2(v)$ that $v$ reports to $u \in N(v)$. This is done by independently picking neighbors $A_u(v) \in N(v)$, where each $A_u(v) \sim \pi^v$. Hence, the path that $v$ reports to $u$ is $p_u(v) \triangleq \langle v, A_u(v), B_v(A_u(v))\rangle$ Algorithm 5 uses this process for reporting paths of length 2. Interestingly, these paths are not independent, however for the case of edge disjoint copies of $C_4$, their "usefulness" in detecting copies of $C_4$ turns out to be independent (see Lemma 31).

### Detecting a Cycle

Consider a copy $C = (v, w, x, u)$ of $C_4$ in $G$. If the 2-path $p_u(v)$ that $v$ reports to $u$ is $p_u(v) = (v, w, x)$, then $u$ can check whether the last vertex $x$ in $p_u(v)$ is also in $N(u)$. If $x \in N(u)$, then the copy $C$ in $G$ of $C_4$ is detected. (The vertex $u$ also needs to verify that $w \neq u$.)

### Description of the Algorithm

The $\varepsilon$-tester for $C_4$-freeness is listed as Algorithm 5. In the first round, each vertex $v$ learns its neighborhood $N(v)$ and the degree of each neighbor. The for-loop repeats $t = O(1/\varepsilon)$ times. Each iteration consists of three rounds. In the first round, $v$ independently draws fresh values for $A_u(v)$ and $B_u(v)$ for each of its neighbors $u \in N(v)$, and sends $B_u(v)$ to $u$. In the second round, for each neighbor $u \in N(v)$, $v$ sends the path $\langle v, A_u(v), B_v(A_u(v))\rangle$. In the third round, $v$ checks if it received a path $\langle w, a, b\rangle$ for a neighbor $w \in N(v)$ where $a \neq v$ and $b \in N(v)$. If this occurs, then $(v, w, a, b)$ is a copy of $C_4$, and vertex $v$ rejects. If $v$ did not reject in all the iterations, then it finally accepts.

### Analysis of the Algorithm

▶ **Definition 30.** We say that $p_u(v)$ is a *success* (wrt $C = (v, w, x, u)$) if $p_u(v) = (v, w, x)$. Let $I_{v,u}$ denote the indicator variable of the event that $p_u(v)$ is a success.

▶ **Lemma 31.** *Let $\{C^j(v_j, w_j, x_j, u_j)\}_{j \in J}$ denote a set of edge-disjoint copies of $C_4$ in $G$. Then the random variables $I_{v_j,u_j}$ are independent.*

**Proof.** The event $I_{v,u} = 1$ occurs iff $A_u(v) = w$ and $B_v(w) = x$. Both $A_u(v)$ and $B_v(w)$ are random variables assigned to (directed) edges. By construction, all the random variables $\{A_u(v)\}_{(u,v \in E} \cup \{B_v(w)\}_{(v,w) \in E}$ are independent. Since the cycles are edge-disjoint, the lemma follows. ◀

▶ Claim 32. $\Pr[I_{v,u} = 1 \mid C] \geq 1/(2m)$.

**Proof.** The path $p_u(v)$ equals $(v, w, x)$ iff $A_u(v) = w$ and $B_v(w) = x$. As $A_u(v)$ and $B_v(w)$ are independent, we obtain

$$\begin{aligned}\Pr\left[I_{v,u} = 1 \mid C\right] &= \Pr\left[A_u(v) = w|C\right] \cdot \Pr\left[B_v(w) = x \mid C\right] \\ &= \frac{d(w) - 1}{\sum_{x \in N(v)}(d(x) - 1)} \cdot \frac{1}{d(w) - 1} \geq \frac{1}{2m}.\end{aligned}$$

◀

▶ Claim 33. If a graph $G$ is $\varepsilon$-far from being $C_4$-free, then it contains at least $\varepsilon \cdot m/4$ edge-disjoint copies of $C_4$.

▶ **Theorem 34.** *Algorithm 5 is an $\varepsilon$-tester for $C_4$-freeness. The round complexity of the algorithm is $\Theta(1/\varepsilon)$ and in each round no more than $O(\log n)$ bits are communicated along each edge.*

**Proof.** *Completeness:* If $G$ is $C_4$-free then Line 7 is never satisfied, hence for every $v$ Algorithm 5 terminates at Line 8.

*Soundness:* Let $G = (V, E)$ be a graph which is $\varepsilon$-far from being $C_4$-free. Therefore, there exist $\ell \triangleq \varepsilon m/4$ edge disjoint copies of $C_4$ in $G$. Denote these copies by $\{C^1, \ldots, C^\ell\}$, where $C^j = (v_j, w_j, w_j, u_j)$. In each iteration, the cycle $C^j$ is detected if $I_{v_j,u_j} = 1$, which (by Claim 32) occurs with probability at least $1/(2m)$. The cycles $\{C^j\}_j$ are edge-disjoint, hence, by Lemma 31, the probability that none of these cycles is detected is at at most $(1-1/(2m))^\ell$. The iterations are independent, and hence the probability that all the iterations fail to detect one of these cycles is at most $(1 - 1/(2m))^{\ell \cdot t}$. Since $\ell = \varepsilon m/4$, setting $t = \Theta(1/\varepsilon)$ reduces the probability of false accept to at most $1/3$, as required. ◀

**Algorithm 5:** $C_4$-free-test$(v)$

1 Send $v$ and $d(v)$ to all $u \in N(v)$ `// v learns` $N(v)$ `and` $d(u)$ `for every` $u \in N(v)$
2 Define the following distribution $\pi^v$ over $N(v)$: For every $w \in N(v)$,
  $\pi^v(w) \triangleq d(w)/\sum_{x \in N(v)} d(x)$ .
3 **for** $t \triangleq \Theta(1/\varepsilon)$ *times* **do**
4  For every neighbor $u \in N(v)$ independently draw $A_u(v) \sim \pi^v$ and
   $B_u(v) \sim U(N(v) \setminus \{u\}$, send $B_u(v)$ to $u$.
5  For every neighbor $u \in N(v)$ send the path $\langle v, A_u(v), B_v(A_u(v))\rangle$ to $u$.
6  **if** $\exists w \in N(v)$ *s.t. $v$ received the path $\langle w, a, b\rangle$ from $w$, where $v \neq a$ and $b \in N(v)$* **then**
7  **return** *REJECT* `// A cycle` $C = (v, w, a, b)$ `was found.`
8 **return** *ACCEPT*

## Extending Algorithm 5

The algorithm can be easily extended to test $H$-freeness for any connected $H$ over four nodes. If $H$ is a $K_{1,3}$ then clearly $H$-freeness can be tested in one round. Otherwise, $H$ is Hamiltonian and can be tested by simply sending an additional bit in the message sent in Line 5 of the algorithm. The additional bit indicates whether $v$ is connected to $B_v(A_u(v))$. Given this information, $u$ can determine the subgraph induced on $\{u, v, A_u(v), B_v(A_u(v))\}$, and hence rejects if $H$ is a subgraph of this induced subgraph. Therefore we obtain the following theorem.

▶ **Theorem 35.** *There is an algorithm which is an $\varepsilon$-tester for $H$-freeness for any connected $H$ over 4 vertices. The round complexity of the algorithm is $\Theta(1/\varepsilon)$ and in each round no more than $O(\log n)$ bits are communicated along each edge.*

## 4.4 Testing $T$-Freeness for any Tree $T$

In this section we generalize the tester by Iwama and Yoshida [25] of testing $k$-path freeness to testing the exclusion of any tree, $T$, of order $k$. We assume that the vertices of $T$ are labeled by $v_0, \ldots, v_{k-1}$. Our tester has a one sided error and it works in the general graph model with random edge queries. This algorithm can be simulated in the CONGEST model. The complexities of the algorithms are stated in the next theorems.

▶ **Theorem 36.** *Algorithm 6 is a global $\varepsilon$-tester, one-sided error for $T$-freeness. The query complexity of the algorithm is $O\left(k^{k^2+1} \cdot \varepsilon^{-k}\right)$. The algorithm works in the general graph model augmented with random edge samples.*

▶ **Theorem 37.** *There is an $\varepsilon$-tester in the CONGEST model that on input $T$, where $T$ is a tree, tests $T$-freeness. The round complexity of the tester is $O\left(k^{k^2+1} \cdot \varepsilon^{-k}\right)$ where $k$ is the order of $T$.*

### Global Algorithm Description.

The algorithm by Iwama and Yoshida [25] for testing $k$-path freeness proceeds as follows. An edge is picked u.a.r. and an endpoint, $v$, of the selected edge,is picked u.a.r. A random walk of length $k$ is performed from $v$, if a simple path of length $k$ is found then the algorithm rejects. The analysis in [25] shows that this process has a constant probability (depends only on $k$ and $\varepsilon$) to find a $k$-path in an $\varepsilon$-far from $k$-path freeness graph.

We generalize this tester in the following straightforward manner. We pick a random vertex $v$ as in the above-mentioned algorithm. The vertex $v$ is a candidate for being the root of a copy of $T$. For the sake of brevity we denote the (possible) root of the copy of $T$ also by $v_0$. From $v$ we start a "DFS-like" revealing of a tree which is a possible copy of $T$ with the first random vertex acting as its root. DFS-like means that we scan a subgraph of $G$ starting from $v$ as follows: the algorithm independently and randomly selects $d_T(v_0)$ neighbors (out of the possible $d_G(v)$) and recursively scans the graph from each of these randomly chosen neighbors. While scanning, if we encounter any vertex more than once then we abort the process (we did not find a copy of $T$). If the process terminates, then this implies that the algorithm found a copy of $T$. In order to obtain probability of success of 2/3 the above process is repeated $t = f(\varepsilon, k)$ times. The listing of this algorithm appears in Algorithm 6. The algorithm can be simulated in the CONGEST model in a straight-forward way. The proofs of Theorems 36 and 37 appear in the full version of this paper.

**Algorithm 6:** Global-tree-free-test$(T, v)$

```
for t ≜ Θ(k^{k^2}/ε^k) times do
    Pick an edge u.a.r. and an endpoint, v, of the selected edge u.a.r.
    Initialize all the vertices in G to be un-labeled.
    Call Recursive-tree-exclusion(T, 0, v) and return REJECT if it returned 1.
return ACCEPT.
```

**Procedure** Recursive-tree-exclusion$(T, i, v)$

1 If $v$ was already labeled then return 0, otherwise, label $v$ by $i$. `// The recursion returns` 0 `if the revealed labeled subgraph is not` $T$.
2 Define $\ell = d_T(v_i) - 1$ if $i > 0$ and $\ell = d_T(v_i)$ otherwise.
3 Let $v_{i_1}, \ldots, v_{i_\ell}$ denote the labels of the children of $v_i$ in $T$ (in which $v_0$ is the root).
4 Pick u.a.r. $\ell$ vertices $u_1, \ldots, u_\ell$ from $N_G(v)$ and recursively call Recursive-tree-exclusion$(T, i_j, u_j)$ for each $j \in [\ell]$
5 If one of the calls returned 0, then return 0, otherwise return 1.

## References


1 Amir Abboud, Keren Censor-Hillel, and Seri Khoury. Near-linear lower bounds for distributed distance computations, even in sparse networks. In *30th International Symposium in Distributed Computing (DISC)*, pages 29–42, 2016.

2 Noga Alon, Sonny Ben-Shimon, and Michael Krivelevich. A note on regular ramsey graphs. *Journal of Graph Theory*, 64(3):244–249, 2010.

3 Noga Alon, Eldar Fischer, Michael Krivelevich, and Mario Szegedy. Efficient testing of large graphs. *Combinatorica*, 20(4):451–476, 2000.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

5 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

6 Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Low-diameter graph decomposition is in nc. In *Scandinavian Workshop on Algorithm Theory*, pages 83–93. Springer, 1992.

7 Guy E Blelloch, Anupam Gupta, Ioannis Koutis, Gary L Miller, Richard Peng, and Kanat Tangwongsan. Nearly-linear work parallel sdd solvers, low-diameter decomposition, and low-stretch subgraphs. *Theory of Computing Systems*, 55(3):521–554, 2014.

8 Luciana Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *25th ACM Symposium on Principles of Database Systems (PODS)*, pages 253–262, 2006.

9 Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. In *30th Int. Symposium on Distributed Computing (DISC)*, volume 9888 of *LNCS*, pages 43–56. Springer, 2016.

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

11 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. In *43rd ACM Symposium on Theory of Computing (STOC)*, pages 363–372, 2011.

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

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

14 Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 652–669. SIAM, 2017.

**15** Paul Erdős, András Hajnal, and J. W. Moon. A problem in graph theory. *The American Mathematical Monthly*, 71(10):1107–1110, 1964.

**16** Guy Even, Reut Levi, and Moti Medina. Faster and simpler distributed algorithms for testing and correcting graph properties in the congest-model. *CoRR*, abs/1705.04898, 2017. URL: `http://arxiv.org/abs/1705.04898`.

**17** Orr Fischer, Tzlil Gonen, and Rotem Oshman. Distributed property testing for subgraph-freeness revisited. *CoRR*, abs/1705.04033, 2017. URL: `http://arxiv.org/abs/1705.04033`.

**18** Pierre Fraigniaud, Pedro Montealegre, Dennis Olivetti, Ivan Rapaport, and Ioan Todinca. Distributed subgraph detection. *CoRR*, abs/1706.03996, 2017. URL: `http://arxiv.org/abs/1706.03996`.

**19** Pierre Fraigniaud and Dennis Olivetti. Distributed detection of cycles. In *29th ACM on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2017.

**20** Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed testing of excluded subgraphs. In *30th Int. Symposium on Distributed Computing (DISC)*, volume 9888 of *LNCS*, pages 342–356. Springer, 2016.

**21** Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1150–1162, 2012.

**22** Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

**23** Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

**24** Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 355–364, 2012.

**25** Kazuo Iwama and Yuichi Yoshida. Parameterized testability. In *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 507–516, 2014. URL: `http://doi.acm.org/10.1145/2554797.2554843`, `doi:10.1145/2554797.2554843`.

**26** Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In *ACM Symposium on Principles of Distributed Computing (PODC)*, 2017.

**27** Stasys Jukna and Georg Schnitger. Triangle-freeness is hard to detect. *Combinatorics, Probability, & Computing*, 11(6):549–569, 2002.

**28** Shay Kutten and David Peleg. Fast distributed construction of small $k$-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998.

**29** Tom Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufmann, 1992.

**30** Christoph Lenzen and Boaz Patt-Shamir. Fast partial distance estimation and applications. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 153–162, 2015.

**31** Gary L Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pages 196–203. ACM, 2013.

**32** Burkhard Monien. How to find long paths efficiently. In *Analysis and design of algorithms for combinatorial problems*, volume 109 of *North-Holland Math. Stud.*, pages 239–254. North-Holland, Amsterdam, 1985.

**33** Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *ACM Symposium on Theory of Computing (STOC)*, pages 565–573, 2014.

34 Hiroaki Ookawa and Taisuke Izumi. Filling logarithmic gaps in distributed complexity for global problems. In *41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 377–388, 2015.

35 Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artif. Intell.*, 29(3):241–288, 1986.

36 David Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000.