# Distributed Detection of Cycles

PIERRE FRAIGNIAUD[†], CNRS and University Paris Diderot
DENNIS OLIVETTI[‡], Aalto University

Distributed property testing in networks has been introduced by Brakerski and Patt-Shamir [6], with the objective of detecting the presence of large dense sub-networks in a distributed manner. Recently, Censor-Hillel et al. [7] have revisited this notion and formalized it in a broader context. In particular, they have shown how to detect 3-cycles in a constant number of rounds by a distributed algorithm. In a follow-up work, Fraigniaud et al. [21] have shown how to detect 4-cycles in a constant number of rounds as well. However, the techniques in these latter works were shown not to generalize to larger cycles $C_k$ with $k \geq 5$. In this article, we completely settle the problem of cycle detection by establishing the following result: For every $k \geq 3$, there exists a distributed property testing algorithm for $C_k$-freeness, performing in a constant number of rounds. All these results hold in the classical CONGEST model for distributed network computing. Our algorithm is 1-sided error. Its round-complexity is $O(1/\epsilon)$ where $\epsilon \in (0, 1)$ is the property-testing parameter measuring the gap between legal and illegal instances.

CCS Concepts: • **Theory of computation → Graph algorithms analysis**; **Distributed algorithms**;

Additional Key Words and Phrases: Distributed computing, distributed decision, distributed property testing, cycle detection, CONGEST model

**ACM Reference format:**
Pierre Fraigniaud and Dennis Olivetti. 2019. Distributed Detection of Cycles. *ACM Trans. Parallel Comput.* 6, 3, Article 12 (October 2019), 20 pages.
https://doi.org/10.1145/3322811

## 1 INTRODUCTION

### 1.1 Context

The objective of (sequential) *property testing* [22] is the design of efficient algorithms for detecting whether data-structures satisfy a given property. In the context of networks, a vast literature has been dedicated to testing the presence or absence of specific patterns such as triangles, cycles, cliques, and so on (see, e.g., References [2, 3, 11, 23]). A property testing algorithm, a.k.a. *tester*, is a centralized algorithm $\mathcal{A}$ that is given the ability to probe nodes with queries of the form $\deg(i)$

A preliminary version of this article has appeared in the proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), Washington D.C., July 24–26, 2017.
[†]Additional support from ANR Project DESCARTES and Inria Project GANG.
[‡]This work was done while this author was visiting IRIF at University Paris Diderot, financially supported by the PhD School of Gran Sasso Science Institute, L'Aquila, Italy. Additional support from ANR Project DESCARTES, and the Academy of Finland, Grant 285721.
Authors' addresses: P. Fraigniaud, CNRS and University Paris Diderot, Bâtiment Sophie Germain, 8 Place Aurélie Nemours, 75013, Paris, France; email: pierre.fraigniaud@irif.fr; D. Olivetti, Department of Computer Science, Aalto University, Konemiehentie 2, 02150 Espoo, Finland; email: dennis.olivetti@aalto.fi.

2329-4949/2019/10-ART12 $15.00
https://doi.org/10.1145/3322811

returning the degree of the $i$th node, and $\mathrm{adj}(i,j)$ returning the identity of the $j$th neighbor of the $i$th node. Besides its running time, the quality of a tester is typically measured by the number of queries that it must perform before deciding whether or not the network satisfies the considered property.

Property testing aims at designing algorithms performing a sub-linear number of requests, and therefore considers promising versions of the problem. Typically, property testing is merely requiring the tester to distinguish between instances satisfying the property, and instances that are *far* from satisfying that property. In the context of networks, several notions of farness have been considered. We consider here the so-called *sparse* model: Given any $\epsilon \in (0,1)$, an $n$-node $m$-edge network $G$ is said to be $\epsilon$-far from satisfying a graph property $\mathcal{P}$ if adding and/or removing at most $\epsilon m$ edges to/from $G$ cannot result in a network satisfying $\mathcal{P}$.

A tester for a graph property $\mathcal{P}$ is a randomized algorithm $\mathcal{A}$ that is required to accept or reject any given network instance, under the following two constraints:

- $G$ satisfies $\mathcal{P} \Longrightarrow \Pr[\mathcal{A} \text{ accepts } G] \geq 2/3$;
- $G$ is $\epsilon$-far from satisfying $\mathcal{P} \Longrightarrow \Pr[\mathcal{A} \text{ rejects } G] \geq 2/3$.

The success guarantee $2/3$ is arbitrary, as one can boost any success guarantee by repetition. In the case of instances that are nearly satisfying $\mathcal{P}$ but not quite, the algorithm can output either ways. Hence, a tester for $\mathcal{P}$ is an algorithm enabling to detect degraded instances (i.e., instances that are far from satisfying a desired property $\mathcal{P}$) with arbitrarily large probability, while correct instances are accepted also with arbitrarily large probability. Also, a tester is 1-sided error if

- $G$ satisfies $\mathcal{P} \Longrightarrow \Pr[\mathcal{A} \text{ accepts } G] = 1$.

Distributed property testing has been introduced in Reference [6], and recently revisited and formalized in Reference [7]. In networks, a *distributed* tester is a distributed algorithm running at every node in parallel (every node executes the same code). After having inspected its surrounding, i.e., the nodes in its vicinity, every node outputs accept or reject. In this distributed setting, we say that a distributed tester $\mathcal{A}$ accepts a network $G$ if and only if all nodes output accept. That is, $\mathcal{A}$ rejects if at least one of the nodes outputs reject. Therefore, a distributed tester for a graph property $\mathcal{P}$ is a randomized algorithm $\mathcal{A}$ that is required to accept or reject any given network instance, under the following two constraints:

- $G$ satisfies $\mathcal{P} \Longrightarrow \Pr[\mathcal{A} \text{ accepts } G \text{ at all nodes}] \geq 2/3$;
- $G$ is $\epsilon$-far from satisfying $\mathcal{P} \Longrightarrow \Pr[\mathcal{A} \text{ rejects } G \text{ in at least one node}] \geq 2/3$.

The fact that the success guarantee $2/3$ is also arbitrary in the distributed setting is not obvious. In fact, it does not hold in general (see, e.g., Reference [19]), as one cannot systematically boost any success guarantee by repetition, because the rejecting node(s) may differ at each repetition. Nevertheless, for 1-sided error distributed testers, i.e., testers leading all nodes to accept legal instances with probability 1, boosting can be applied, and therefore the success guarantee $2/3$ becomes indeed arbitrary. In this article, all testers are 1-sided.

In this article, we are focusing on the detection of cycles, one of the most basic and central structures in graph theory, with impact on Ramsey theory and block design. Let $k \geq 3$. A $k$-node cycle, or $k$-cycle for short, is denoted by $C_k$. A network $G$ is $C_k$-free if and only if $G$ does not contain a $k$-node cycle as a subgraph. A case of particular interest is $k = 3$, and a $C_3$ is often called triangle.

It has been shown in Reference [7] that, in the classical CONGEST model[1] for distributed computing [31], there exists a distributed property testing algorithm for triangle-freeness performing

[1] *The* CONGEST *model states that all nodes perform synchronously in a sequence of rounds; at each round, messages of $O(\log n)$ bits can be exchanged along the edges of the network.*

in $O(1/\epsilon^2)$ rounds. This result has been extended in Reference [21] where it is proved that there exists a distributed property testing algorithm for $C_4$-freeness performing in $O(1/\epsilon^2)$ rounds as well.

Perhaps surprisingly, the techniques in References [7, 21] do not extend to larger cycles. Indeed, using explicit constructions of so-called Behrend graphs, it was proved in Reference [21] that these techniques fail for most values of $k \geq 5$. That is, these techniques cannot result in a tester performing in a constant number of rounds in all graphs, even if the constant is allowed to be a function of $1/\epsilon$. Prior to this work, the existence of distributed property testing algorithms for $C_k$-freeness performing in a constant number of rounds was open for $k$ larger than 4.

## 1.2 Our Results

We completely settle the problem of cycle detection for every possible length $k \geq 3$. Specifically, we prove that, for every $k \geq 3$, there exists a 1-sided error distributed property testing algorithm for $C_k$-freeness, performing in $O(1/\epsilon)$ rounds in the CONGEST model.

Essentially, we reduce the problem of detecting $k$-cycles to the problem of detecting whether a given edge $e$ belongs to some $C_k$. At first glance, the latter problem may seem to be much more simple. Indeed, it does not require to deal with the link congestion caused by the simultaneous testing of several edges. However, the problem remains actually quite challenging as, in the CONGEST model, even collecting the identities of the nodes at distance 2 from a given node $u$ might be impossible to achieve in $o(n)$ rounds in $n$-node network. Indeed, $u$ may have constant degree, with $\Omega(n)$ neighbors at distance 2. To overcome this difficulty, we proceed by pruning the set of information transmitted between nodes, namely by pruning the set of candidate cycles passing through the given edge $e$. This pruning is at the risk of discarding candidate cycles that would have turned out to be actual cycles. Nevertheless, our pruning mechanism guarantees that at least one actual cycle remains in the current set of candidate cycles throughout the execution of the algorithm.

Interestingly, the use of randomization is limited to the reduction of the general problem of testing $C_k$-freeness to the problem of detecting whether there exists a $k$-cycle *passing through a given edge $e$*. Indeed, our algorithm solving the latter problem is *deterministic*. In particular, the aforementioned pruning mechanism is deterministic. That is, the existence of an actual cycle passing through $e$ among the restricted set of candidate cycles kept at each round is not a property that holds under some statistical guarantee, but it holds systematically. Moreover, even if there is just a *single* $k$-cycle passing through $e$, that cycle will be detected by our algorithm. That is, our algorithm for detecting whether there exists a $k$-cycle passing through a given edge $e$ does not rely on the $\epsilon$-farness assumption.

*Remark.* After the acceptance of the conference version of this article, we became aware of the existence of a combinatorial lemma due to Erdős et al. [14], stating the following: Let $V$ be a set of size $n$, and let us fix two integers $p$ and $q$ with $p + q \leq n$. Then, for any set $F \subseteq \mathcal{P}(V)$ of subsets of size at most $p$ of $V$, there exists a subset $\widehat{F}$ of $F$ of cardinality at most $\binom{p+q}{p}$ such that, for every 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 $\widehat{L} \in \widehat{F}$ such that $\widehat{L} \cap C = \emptyset$. This combinatorial result has been used in different contexts, including the design of sequential parametrized algorithms for the longest path problem [27]. In this article, we show that it can also be used to the benefit of designing efficient distributed algorithms. Specifically, the lemma is used as follows: As said before, at each round, every node receives a collection of partial cycles, that is a collection $F$ of ID-sequences of length $p < k$, and it questions whether there exists a set $C$ of $q = k - p$ nodes that, concatenated with some partial cycle $L \in F$, would form a cycle of length $k$. Roughly, the lemma says there is no need to keep track of the entire collection $F$, as a small sub-collection $\widehat{F}$ of $\binom{k}{p}$ partial cycles suffices. Indeed, for every $C$, if there exists a sequence $L \in F$ of nodes susceptible to form a cycle together with $C$, then there also exists a sequence $\widehat{L} \in \widehat{F}$ of nodes susceptible to form a cycle with $C$.

## 1.3 Related Work

*1.3.1 Property Testing.* The property of $H$-freeness has been the subject of a lot of investigation in classical (i.e., sequential) property testing. In the so-called *dense* model, most solutions exploit the *graph removal lemma*, which essentially states that, for every $k$-node graph $H$, and every $\epsilon > 0$, there exists $\delta > 0$ such that every $n$-node graph containing at most $\delta n^k$ (induced) copies of $H$ can be transformed into an (induced) $H$-free graph by deleting at most $\epsilon n^2$ edges. This lemma was first proved for the case $k = 3$, and later generalized to subgraphs $H$ of arbitrary size [13], and further to induced subgraphs [1]. It is possible to exploit this lemma for testing the presence of any (induced or not) subgraph of constant size, in constant time. Notice that $\delta$ is a fast growing function of $\epsilon$ and $k$. The growth of the function was later improved in Reference [3] under some assumptions. For more details on the graph removal lemma, see Reference [10].

Cycle-detection has also been considered in the so-called *sparse* model. On bounded degree graphs, cycle-freeness can be tested with $O(\frac{1}{\epsilon^3} + \frac{d}{\epsilon^2})$ queries [23] by a 2-sided error algorithm, where $d$ is the maximum degree of the graph. However, if we restrict ourselves to 1-sided error algorithms, then the problem becomes harder. A lower bound of $\Omega(\sqrt{n})$ queries was established in Reference [11]. The same paper presents a tester requiring $\widetilde{O}(\text{poly}(1/\epsilon)\sqrt{n})$ queries in arbitrary graphs, and another tester requiring $\widetilde{O}(\text{poly}(d^k/\epsilon)\sqrt{n})$ queries in graphs with maximum degree $d$ for detecting cycles of length at least $k$. Detecting triangles requires at least $\Omega(n^{1/3})$ queries, and at most $O(n^{6/7})$ queries (see Reference [2]). The same lower bound holds for detecting any non bipartite subgraph $H$, and for 2-sided error algorithms as well. For some specific subgraphs $H$, the lower bound can even be as high as $\Omega(n^{1/2})$.

*1.3.2 Distributed Property Testing.* Distributed property testing has been introduced in Reference [6], and fully formalized in Reference [7]. The authors of the latter paper show that, in the dense model, any tester for a *non-disjointed* property can be emulated in the distributed setting with just a quadratic slowdown, i.e., if a sequential tester makes $q$ queries, then it can be converted into a distributed tester that performs in $O(q^2)$ rounds. This simulation exploits the fact that any dense tester can be converted to a tester that first chooses some nodes uniformly at random, gathers their edges, and then performs centralized analysis of the obtained data (see Reference [24]).

The authors of Reference [7] also provide distributed testers for the sparse model, showing that it is possible to test triangle-freeness in $O(1/\epsilon^2)$ rounds, cycle-freeness in $O(1/\epsilon \log n)$ rounds, and, in bounded degree graphs, bipartiteness in $O(\text{poly}(1/\epsilon \log(n/\epsilon)))$ rounds. Their work was inspired by Reference [6], where a constant-time distributed algorithm for finding a linear-size $\epsilon$-near clique is proposed, under the assumption that the graph contains a linear-size $\epsilon^3$-near clique. (An $\epsilon$-near clique is a set of nodes where all but an $\epsilon$ fraction of pairs of nodes have edges between them).

The result in Reference [7] regarding testing triangle-freeness was extended in Reference [21], where it is shown that, for every 4-node connected graph $H$, there exists a distributed tester for $H$-freeness performing in $O(1/\epsilon^2)$ rounds. Also, Reference [21] provides a proof that the approaches in References [7, 21] fail to test $C_k$-freeness in a constant number of rounds, whenever $k \geq 5$.

Very recently, three sets of authors independently generalized the results in this article, by showing that not only $C_k$-freeness can be tested in $O(1/\epsilon)$ rounds, but also $H$-freeness, for every graph pattern $H$ composed of a tree (or a forest) $F$, an edge $e$, and arbitrary connections between the extremities of $e$ and the nodes of $F$ (see the joint publication, Reference [15]).

*1.3.3 Distributed Decision.* Distributed property testing fits into the larger framework of *distributed decision.* The seminal paper [29] was perhaps the first to identify the connection between the ability to locally check the correctness of a solution in a distributed manner, and the ability to design an efficient deterministic distributed algorithm for constructing a correct solution. Since then, there have been a huge amount of contributions aiming at studying variants of distributed decision, in the deterministic setting (see, e.g., Reference [20]), the anonymous setting (see, e.g., Reference [12]), the probabilistic setting (see, e.g., References [16, 19]), the non-deterministic setting (see, e.g., References [25, 26]), and even beyond (see, e.g., References [4, 18]). We refer to Reference [17] for a survey on distributed decision.

*1.3.4 Distributed Cycle Detection.* Cycle detection has been investigated in various parallel and distributed computing frameworks, in particular for its connection to deadlock detection in routing or databases. The algorithm in Reference [32] detects cycles in directed graphs, under the bulk-synchronous model. It proceeds in phases. Every node sends its identifier (ID) to all its out-neighbors at the first phase. Then, at each phase, every node (1) receives a collection of ID-sequences from each of its in-neighbors, (2) appends its own ID to each such ID-sequence, and (3) sends the new set of ID-sequences to each of its out-neighbors. Moreover, to consume less bandwidth, some ID-sequences are discarded if either the first identifier of a sequence is the same as the one of the node (a cycle has then been detected, and the node can just stop), or the identifier of the node is contained in a sequence (the node can discard the sequence, since it contains a cycle already detected by some other node). This algorithm can be adapted to detect cycles of a given length $k$, and can be adapted to undirected graphs. Our algorithm uses a similar "append-and-forward" technique. However, even for detecting cycles of given length $k$ passing through a given edge $e$, the algorithm in Reference [32] may consume a bandwidth $\Omega(n)$ in $n$-node graphs. This is because the number of ID-sequences that a node sends depends on its in-degree, which can be $\Omega(n)$. Instead, our algorithm discards much more ID-sequences, and the number of sequences sent by a node at each round depends solely on $k$, and *not* on $n$. It follows that, for constant $k$, our algorithm is using messages of logarithmic length, and runs in constant time in the CONGEST model.

The algorithm in Reference [30], also designed for the bulk-synchronous model, uses an approach different from the "append-and-forward" technique. This approach uses the fact that, in directed graphs, a cycle cannot pass through nodes of in-degree or out-degree 0. Hence, one can remove all nodes with in-degree or out-degree 0 from the digraph, iteratively, until the digraph becomes empty, or no more nodes can be removed. It is proved in Reference [30] that the digraph becomes empty if and only if it does not contain a cycle. This approach is bandwidth-efficient. However, it appears difficult to adapt this idea for detecting cycles of some given length $k$ in constant time, in particular because of the chains of dependencies between nodes (a node might be removable only after having removed a faraway node), or for detecting cycles in undirected graphs.

Finally, the algorithm in Reference [5], as well as the algorithms in References [8, 9], designed for message passing models or for self-stabilization frameworks, are detecting cycles of arbitrary length in general graphs. However, these algorithms are designed for optimizing their message-complexity, and, as such, they do not aim at limiting their number of rounds in a model where the number of bits traversing each and every edge at each round is limited. It follows that these algorithms appear inadequate for efficiently detecting cycles of a specific length in the CONGEST model.

## 2 MODEL AND DEFINITIONS

### 2.1 The CONGEST Model

We consider the classical CONGEST model for distributed network computing [31]. The network is modeled as a connected simple graph (no self-loops and no parallel edges). The nodes of the graph are computing entities exchanging messages along the edges of the graph. Nodes are given arbitrary distinct identities (IDs) in a range polynomial in $n$, in $n$-node networks. Hence, every ID can be stored on $O(\log n)$ bits.

In the CONGEST model, all nodes start simultaneously and execute the same algorithm in parallel. Computation proceeds synchronously, in a sequence of *rounds*. At each round, every node

- performs some individual computation,
- sends messages to neighbors in the network, and
- receives messages sent by neighbors.

The main constraint imposed by the CONGEST model is a restriction of the amount of data that can be transferred between neighboring nodes during a round: Messages are bounded to be of $O(\log n)$ bits.

The $O(\log n)$-bit bound on the message size enables the transmission of a constant number of IDs between nodes at each round. The CONGEST model is well suited for analyzing the impact of limiting the throughput of a network on its capacity to solve tasks efficiently. The complexity of a distributed algorithm in the CONGEST model is expressed in number of rounds.

In this article, we are mostly interested in solving tasks locally. Hence, we are mainly focusing on the design of algorithms performing in a constant number of rounds in the CONGEST model.

## 2.2 Distributed Property Testing

Let $\mathcal{P}$ be a graph property such as, e.g., planarity, cycle-freeness, bipartiteness, $C_k$-freeness, and so on. Let $\epsilon \in (0, 1)$. Recall that a graph $G$ is said to be $\epsilon$-far from satisfying $\mathcal{P}$ if removing and/or adding at most $\epsilon m$ edges to/from $G$ cannot result in a graph satisfying $\mathcal{P}$.

A distributed property testing algorithm for $\mathcal{P}$ is a randomized algorithm that performs as follows: Initially, every node is only given its ID as input. After a certain number of rounds, every node must output a value in {accept, reject}. The algorithm is correct if and only if the following two conditions are satisfied:

- if $G$ satisfies $\mathcal{P}$, then Pr[every node outputs accept] $\geq 2/3$;
- if $G$ is $\epsilon$-far from satisfying $\mathcal{P}$, then Pr[at least one node outputs reject] $\geq 2/3$.

The algorithm is 1-sided error if, whenever $G$ satisfies $\mathcal{P}$, the probability that every node outputs accept equals 1, i.e., if $G$ satisfies $\mathcal{P}$, then

- if $G$ satisfies $\mathcal{P}$, then Pr[every node outputs accept] $= 1$.

Let $k \geq 3$. A $k$-node cycle, or simply $k$-cycle for short, consists of $k$ nodes $x_i$, and $k$ edges

$$\{x_i, x_{i+1 \bmod k}\},\ i = 0, \ldots, k-1.$$

Such a graph is denoted by $C_k$. Given a graph $G$, its set of nodes (respectively, edges) is denoted by $V(G)$ (respectively, $E(G)$). Throughout the article, $n = |V(G)|$, and $m = |E(G)|$.

Recall that a graph $H$ is a *subgraph* of a graph $G$ if and only if

$$V(H) \subseteq V(G) \text{ and } E(H) \subseteq E(G).$$

*Definition 2.1.* A network $G$ is $C_k$-free if and only if $G$ does not contain a $k$-node cycle as a subgraph.

Our objective is the design of efficient distributed property testing algorithms for $C_k$-freeness, for all $k \geq 3$.

## 3 DETECTING CYCLES

In this section, we establish our main result.

Theorem 3.1. *For every $k \geq 3$, there exists a 1-sided error distributed property testing algorithm for $C_k$-freeness performing in $O(\frac{1}{\epsilon})$ rounds in the* CONGEST *model.*

The rest of the section is dedicated to the proof of the theorem. Let us fix $k \geq 3$. We need to show that there exists a distributed tester for $C_k$-freeness performing in $O(\frac{1}{\epsilon})$ rounds, satisfying Pr[every node outputs accept] = 1 if $G$ is $C_k$-free, and Pr[at least one node outputs reject] $\geq 2/3$ otherwise.

Our tester algorithm for detecting $C_k$ proceeds in two phases:

(1) determining a candidate edge $e$ susceptible to belong to some cycle $C_k$, if any;
(2) checking the existence of a cycle $C_k$ passing through $e$.

Only the first phase is randomized; the second phase is fully deterministic.

### 3.1 Description of Phase 1

We assign weights to edges such that, with sufficient high probability, there is a unique edge $e$ with minimum weight. For this purpose, we exploit the *isolation lemma* in Reference [28]. We start by assigning to each node $u$ a weight $w(u) \in [1, 2n]$. Then, we assign to each edge $e = \{u, v\}$ the weight $r(e) = w(u) + w(v)$, called the *rank* of $e$. Note that one round suffices for assigning a rank to every edge. The isolation lemma guarantees that, with probability at least 1/2, there is a unique edge with minimum rank.

Ideally, we would like to let all nodes of the graph know the edge of minimum rank and start performing the second phase of the algorithm for that edge. This is hard to achieve in a constant number of rounds, since identifying the edge with minimum rank (e.g., by leader election) requires a number of rounds equal to the diameter. So, we shall perform Phase 2 of our algorithm for all edges in parallel. However, to avoid consuming too much bandwidth, the process carries on by stopping the search related to edges with high ranks. More specifically, we proceed as follows: Every node $u$ selects the edge $e_u$ of lowest rank among all its incident edges, where ties are broken arbitrarily (e.g., based on the ID of extremities) and starts performing the second phase, which consists in checking whether there exists a cycle $C_k$ passing through $e_u$.

To avoid congestion, every node performs only instructions of Phase 2 related to the edge with smallest rank it becomes aware of during the execution of the algorithm (again, ties are broken arbitrarily), in a way similar to the prioritized search in Reference [7]. Specifically, if a node $u$ currently involved in checking the existence of a cycle $C_k$ passing through $e$ receives a message related to checking the existence of a cycle $C_k$ passing through $e' \neq e$, then $u$ discards this message if

$$r(e') > r(e),$$

and otherwise switches to checking the existence of cycles passing through $e'$. This guarantees that no two messages corresponding to checking the existence of a cycle $C_k$ passing through two different edges ever traverse an edge in the same direction at the same round. Moreover, if there is a unique edge $e$ with minimum rank, then no nodes discard messages related to checking the existence of a cycle $C_k$ passing through $e$, and thus the checking phase for $e$ will not be interrupted.

Note that it might be the case that a node becomes inactive at a given round. This is for instance the case if it performed the instructions of Phase 2 for detecting a cycle passing through an edge $e$ at some round and subsequently received messages related to edges $e'$ of higher ranks, which leads that node to discard all received messages. As a result, a node may receive no messages at some rounds, in case all its neighbors discarded all their messages at these rounds. Yet, no nodes stop performing Phase 2, as it may happen that, later, messages related to an edge $e'$ of rank smaller than the one of $e$ will be received. Observe that no nodes need to know the edge of minimum rank for performing Phase 2. In fact, some nodes will not even eventually identify the edge with minimum rank. Indeed, consider for instance the two edges $e$ and $e'$ with smallest ranks, and assume $r(e') > r(e)$. If $e$ and $e'$ are at distance $> 2k$, then the two parallel search for a cycle of length $k$ passing through $e$, and for a cycle of length $k$ passing through $e'$ do not interfere, and so the search will complete for the edge $e'$, even though $e'$ is not the edge with minimum rank.

Before analyzing Phase 1, we now describe Phase 2, which is the core of the property testing algorithm for $C_k$-freeness. For simplifying the presentation, let us fix some edge

$$e = \{u, v\},$$

and let us describe Phase 2 for edge $e$ only, assuming that no other checks for other edges are running concurrently. Likewise, the reader can assume that $e$ is the unique edge with minimum rank in $G$, which guarantees that the Phase 2 for $e$ will not be slowed down by messages corresponding to Phase 2 applied to other edges.

## 3.2 Description of Phase 2

We describe the algorithm used to check whether there exists a cycle $C_k$ passing through a given edge $e$. The algorithm proceeds in $\lfloor \frac{k}{2} \rfloor$ rounds. At each round $t = 1, \ldots, \lfloor \frac{k}{2} \rfloor$ of the algorithm, sequences of $t$ IDs are exchanged between nodes participating to the search for $C_k$. Every node that receives some sequences at round $t$ concatenates its own ID to each received sequence and sends the resulting collection of sequences to all its neighbors.

Before describing our general algorithm, let us first show how to detect a $C_5$ passing through $e = \{u, v\}$, as a warm-up (recall that even testing $C_5$-freeness was open prior to this work). As it will be shown, the trivial "append-and-forward" technique is enough to perform $C_5$-detection in $\lfloor \frac{k}{2} \rfloor = 2$ rounds, but it cannot be used to detect $C_7$ in $\lfloor \frac{k}{2} \rfloor = 3$ rounds, because it requires too much bandwidth for large $k$.

*3.2.1 Detecting $C_5$.* The distributed algorithm is described in Algorithm 1. In this algorithm, Round 1 consists in nodes $u$ and $v$ sending their IDs to their neighbors (all other nodes are doing essentially nothing, just sending empty messages). A node may thus receive zero, one, or two IDs depending on whether it is adjacent to none, one of, or both nodes $u$ and $v$. Each node $x$ that received at least one of these IDs forms a set $\mathcal{R}$ of 1-element sequences of the form $(\mathrm{ID}(w))$, where $w \in \{u, v\}$. Such a node appends its own ID to each sequence and sends the resulting set $\mathcal{S}$ of sequences to all its neighbors at Round 2. (If $\mathcal{R} = \emptyset$, then $\mathcal{S} = \emptyset$ as well.) A node $z$ that, at Round 2, receives a sequence $(\mathrm{ID}(u), \mathrm{ID}(x))$, and a sequence $(\mathrm{ID}(v), \mathrm{ID}(y))$ from distinct neighbors $x$ and $y$, respectively, both distinct from $u$ and $v$, has detected the presence of the cycle $(u, x, z, y, v)$.

Figure 1 shows an example of $C_5$-detection. At Round 1, $x_3$ receives messages from both $u$ and $v$, and thus sends the sequences $\{(\mathrm{ID}(u), \mathrm{ID}(x_3)), (\mathrm{ID}(v), \mathrm{ID}(x_3))\}$ to its neighbors at Round 2. At Round 1, $x_6$ receives a message from $v$, and thus sends $\{(\mathrm{ID}(v), \mathrm{ID}(x_6))\}$ to its neighbors at Round 2. At Round 2, $x_{10}$ has received the sequences in $\{(\mathrm{ID}(u), \mathrm{ID}(x_3)), (\mathrm{ID}(v), \mathrm{ID}(x_3)), (\mathrm{ID}(v), \mathrm{ID}(x_6))\}$, and thus detects the cycle $(u, x_3, x_{10}, x_6, v)$.

This trivial "append-and-forward" technique can obviously be extended to detecting $C_k$, for arbitrary $k \geq 5$. However, a node of high degree may have to forward very many sequences during a round (this is typically the case of a node connected to $u$ and/or $v$ via many vertex-disjoint paths of same length), violating the bandwidth restriction of the CONGEST model. In fact, even for

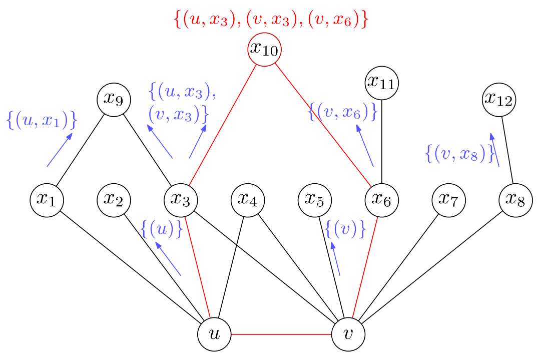

Fig. 1. Detecting $C_5$ passing through $\{u, v\}$.

**ALGORITHM 1:** $C_5$ detection for edge $e = \{u, v\}$ executed by node with ID *myid*.

```
1:  function DETECT_$C_5(u, v)$
2:      if $myid = u$ or $myid = v$ then
3:          $\mathcal{S} \leftarrow \{(myid)\}$
4:      else
5:          $\mathcal{S} \leftarrow \emptyset$
6:      end if
7:      send $\mathcal{S}$ to all neighbors
8:      receive messages from all neighbors
9:      $\mathcal{R} \leftarrow$ received sequences
10:     $\mathcal{S} \leftarrow$ append $myid$ at the tail of each $L \in \mathcal{R}$
11:     send $\mathcal{S}$ to all neighbors
12:     receive messages from all neighbors
13:     if received two disjoint sequences then
14:         output reject
15:     else
16:         output accept
17:     end if
18: end function
```

detecting $C_7$ this basic technique fails, because a node at distance 2 from $u$ (respectively, from $v$) may have to forward a number of sequences that is linear in the degree of $u$ (respectively, $v$), which could depend on $n$.

The main concern of our algorithm is therefore to limit the maximum number of different sequences of IDs to be sent by each node during the execution. Yet, it is crucial that nodes forward sufficiently many sequences of IDs to guarantee detection. (For instance, in the graph depicted on Figure 1, node $x_3$ receives both ID($u$) and ID($v$) at the first round. If $x_3$ forwards only the sequence (ID($v$), ID($x_3$)) to its neighbors, then the 5-cycle will not be detected by $x_{10}$). In other words, discarding too many sequences may prevent the algorithm from detecting the cycle, while forwarding too many sequences overloads the communication links. We show that sending a constant number of sequences is sufficient to guarantee cycle detection whenever these sequences are carefully chosen. We first show how to circumvent the bandwidth limitation while testing $C_7$-freeness, and we then show how to generalize the procedure for any $C_k, k \geq 3$.

### 3.2.2 *Detecting $C_7$.*

Algorithm 2 shows how to test $C_7$ freeness in three rounds. The procedure is very similar to Algorithm 1, only Lines 12–18 have been added. This algorithm essentially uses

**ALGORITHM 2:** $C_7$ detection for edge $e = \{u, v\}$ executed by node with ID *myid*.

```
1: function DETECT_$C_7(u, v)$
2:     if $myid = u$ or $myid = v$ then
3:         $\mathcal{S} \leftarrow \{(myid)\}$
4:     else
5:         $\mathcal{S} \leftarrow \emptyset$
6:     end if
7:     send $\mathcal{S}$ to all neighbors
8:     receive messages from all neighbors
9:     $\mathcal{R} \leftarrow$ received sequences
10:    $\mathcal{S} \leftarrow$ append $myid$ at the tail of each $L \in \mathcal{R}$
11:    send $\mathcal{S}$ to all neighbors
12:    receive messages from all neighbors
13:    $\mathcal{R} \leftarrow$ received sequences
14:    $c_1 \leftarrow$ number of sequences in $\mathcal{R}$ starting with $u$
15:    $c_2 \leftarrow$ number of sequences in $\mathcal{R}$ starting with $v$
16:    $\mathcal{S}_1 \leftarrow min\{c_1, 3\}$ sequences starting with $u$ in $\mathcal{R}$, chosen arbitrarily
17:    $\mathcal{S}_2 \leftarrow min\{c_2, 3\}$ sequences starting with $v$ in $\mathcal{R}$, chosen arbitrarily
18:    $\mathcal{S} \leftarrow \mathcal{S}_1 \cup \mathcal{S}_2$
19:    $\mathcal{S} \leftarrow$ append $myid$ at the tail of each $L \in \mathcal{S}$
20:    send $\mathcal{S}$ to all neighbors
21:    receive messages from all neighbors
22:    if received two disjoint sequences then
23:        output reject
24:    else
25:        output accept
26:    end if
27: end function
```

the append-and-forward technique, but limiting to six the number of sequences of length 3 that can be sent, three sequences for each initial node $u$ or $v$. Notice that a node may receive $\Omega(n)$ sequences of length 2. In Algorithm 2, just three (per initial node) out of this up to $\Omega(n)$ sequences are forwarded at the third round.

To see why there is no need to send more than three sequences of length 3 at Round 3, let us consider a node $z$ that received three triples of the form

$$(\mathrm{ID}(u), \mathrm{ID}(x_1), \mathrm{ID}(a)),\ (\mathrm{ID}(u), \mathrm{ID}(x_2), \mathrm{ID}(a)),\ \text{and } (\mathrm{ID}(u), \mathrm{ID}(x_3), \mathrm{ID}(a))$$

from some node $a$, and three triples

$$(\mathrm{ID}(v), \mathrm{ID}(y_1), \mathrm{ID}(b)),\ (\mathrm{ID}(v), \mathrm{ID}(y_2), \mathrm{ID}(b)),\ \text{and } (\mathrm{ID}(v), \mathrm{ID}(y_3), \mathrm{ID}(b))$$

from some node $b$. Then, one can easily check that there must exist $x_i$ and $y_j$, $1 \leq i, j \leq 3$, satisfying

$$\begin{cases} x_i & \neq & y_j \\ x_i & \neq & b \\ y_j & \neq & a \end{cases},$$

which is a witness of the 7-cycle

$$(u, x_i, a, z, b, y_j, v).$$

Letting $a$ and $b$ send more triples would have been of no help, as three are sufficient. Note that, assuming the existence of the 7-cycle $(u, x_i, a, z, b, y_j, v)$, it may have been the case that $a$ received less than three pairs at the second round, leading that node to send less than three triples at the

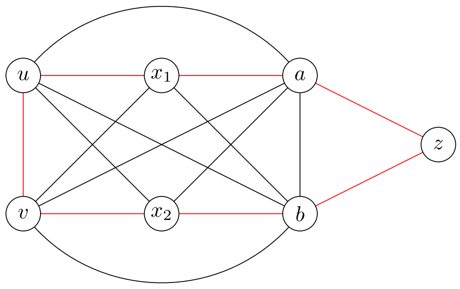

Fig. 2. Detecting $C_7$ passing through $\{u, v\}$.

third round. In this case, even if $b$ received more than three pairs at the second round, a similar reasoning as above shows that having $b$ send more than three triples at the third round would be useless.

Note that, with the technique of Algorithm 2, not less than three triples containing $u$, and three containing $v$ may be sent at the third round. Indeed, let us consider the example depicted in Figure 2. In this example, if Node $a$ sends $\{(u, x_1, a), (u, b, a), (v, x_1, a), (v, b, a)\}$ to $z$ at the third round, while Node $b$ sends $\{(u, x_1, b), (u, a, b), (v, x_1, b), (v, a, b)\}$ to $z$ at the same round, then $z$ cannot detect the 7-cycle.

We now move on to the general case, where we show that, for every $k \geq 3$, we do not need to send more than a constant number of sequences at each round.

*3.2.3 General Case: Detecting $C_k$ for Every $k \geq 3$.* The pseudocode of our algorithm for detecting a $C_k$ including a given edge $\{u, v\}$ is depicted as Algorithm 3. Algorithm 3 is essentially of the form "append-and-forward" (cf. Instruction 25), but selects only a few lists to be sent at each round. The "seed" lists are just formed by ID$(u)$ and ID$(v)$ (cf. Instruction 3). The algorithm proceeds in $\lfloor \frac{k}{2} \rfloor$ rounds (cf. the for-loop of Instruction 9). At each round, every node that received non-empty messages collects all IDs that were contained in these messages, distinct from its own ID, in a set $\mathcal{I}$ (cf. Instructions 12–14). Then, at round $t$, a set of $k - t$ "fake" IDs are added in $\mathcal{I}$ (cf. Instruction 15). Intuitively, these fake IDs represent the yet-unknown IDs of nodes that could potentially form a $C_k$ together with the nodes of some list received by the current node at this round. They are used for the purpose of selecting the sequences that will be forwarded at the next round. Specifically, to form the collection $\mathcal{S}$ of lists that will be sent to neighbors at the next round $t$ (cf. Instruction 29), the collection $\mathcal{X}$ of all possible sets $X$ of $k - t$ IDs are constructed, including fake IDs (cf. Instruction 16).

The core of the algorithm is the construction of $\mathcal{S}$ by the Instructions from 17 to 25. Before describing this crucial part of Algorithm 3 in detail, let us complete the description of the final part of the algorithm.

At Round $\lfloor \frac{k}{2} \rfloor$, all lists of IDs sent and received at this round, or received during the previous round, are considered, and stored in a set of lists $\mathcal{R}$ (cf. Instructions 33–38). If a node $w$ has two lists $L_1$ and $L_2$ in $\mathcal{R}$ such that

$$|L_1 \cup L_2 \cup \{\text{ID}(w)\}| = k,$$

then Node $w$ outputs reject, as a cycle $C_k$ has been discovered.

Before showing that a cycle $C_k$ formed by all nodes with IDs in $L_1 \cup L_2 \cup \{\text{ID}(w)\}$ exists if and only if $|L_1 \cup L_2 \cup \{\text{ID}(w)\}| = k$, we first return to the core of Algorithm 3, which is the setup of the set of lists $\mathcal{R}$.

**ALGORITHM 3:** $C_k$ detection for edge $e = \{u, v\}$ executed by node with ID *myid*.

```
function DETECTCK(u,v)
    if $myid = u$ or $myid = v$ then                              ▷ initial computation at round 1
        $\mathcal{S} \leftarrow \{(myid)\}$                        ▷ $\mathcal{S}$ is a set of sequences of IDs
    else
        $\mathcal{S} \leftarrow \emptyset$
    end if
    send $\mathcal{S}$ to all neighbors                              ▷ send operation at round 1
    receive messages from all neighbors                              ▷ receive operation at round 1
    for $t = 2$ to $\lfloor \frac{k}{2} \rfloor$ do                 ▷ rounds 2 to $\lfloor \frac{k}{2} \rfloor$
        if non-empty messages have been received at round $t-1$ then
            $\mathcal{R} \leftarrow$ set of all ordered sequences of IDs received at round $t-1$
                                                                     ▷ $\mathcal{R}$ contains sequences of $t-1$ IDs
            remove from $\mathcal{R}$ all sequences containing $myid$
            $\mathcal{I} \leftarrow$ set of IDs included in at least one sequence in $\mathcal{R}$
            $\mathcal{I} \leftarrow \mathcal{I} \cup \{-1, \dots, -k+t\}$   ▷ add $k-t$ distinct “fake” IDs to $\mathcal{I}$
            $\mathcal{X} \leftarrow$ collection of all sets $X$ of $k-t$ IDs in $\mathcal{I}$
            $\mathcal{S} \leftarrow \emptyset$                        ▷ initializes the set of sequences to be sent
            for all $L \in \mathcal{R}$ do
                $C \leftarrow \{X \in \mathcal{X} : X \cap L = \emptyset\}$   ▷ $C$ is a sub-collection of sets $X$ of $k-t$ IDs
                if $C \neq \emptyset$ then
                    $\mathcal{S} \leftarrow \mathcal{S} \cup \{L\}$      ▷ $\mathcal{S}$ contains ordered sequences of existing IDs
                    $\mathcal{X} \leftarrow \mathcal{X} \setminus C$
                end if
            end for
            append $myid$ at the tail of each $L \in \mathcal{S}$      ▷ $\mathcal{S}$ contains sequences of $t$ IDs
        else
            $\mathcal{S} \leftarrow \emptyset$
        end if
        send $\mathcal{S}$ to all neighbors                          ▷ send operation at round $t$
        receive messages from all neighbors                          ▷ receive operation at round $t$
    end for
    if non-empty messages have been received at any round $1, \dots, \lfloor \frac{k}{2} \rfloor$ then
        if $k$ is odd then
            $\mathcal{R} \leftarrow$ {sequences received at round $\lfloor \frac{k}{2} \rfloor$}   ▷ $\mathcal{R}$ contains sequences of equal length
        else
            $\mathcal{R} \leftarrow \mathcal{S} \cup$ {sequences received at round $\lfloor \frac{k}{2} \rfloor - 1$}
                                          ▷ $\mathcal{R}$ contains sequences of lengths differing by at most 1
        end if
        if $\exists L_1, L_2 \in \mathcal{R} : |L_1 \cup L_2 \cup \{myid\}| = k$ then
            output reject                                            ▷ a $C_k$ has been detected
        else output accept
        end if
    else output accept
    end if
end function
```

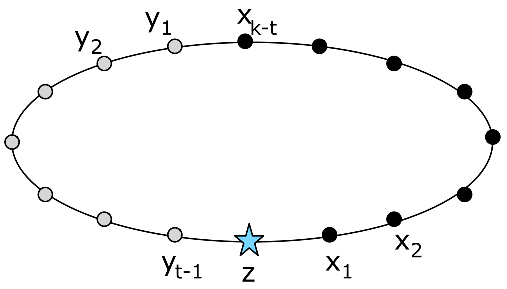

Fig. 3. Construction of $\mathcal{R}$.

*Construction of the set of lists to be sent at each round.* For comfort and ease of reading, we repeat below the instructions performed by Algorithm 3 for computing the set $\mathcal{S}$ of ordered sequences to be sent to all neighboring nodes.

```
$\mathcal{S} \leftarrow \emptyset$
**for all** $L \in \mathcal{R}$ **do**
  $C \leftarrow \{X \in \mathcal{X} : X \cap L = \emptyset\}$
  **if** $C \neq \emptyset$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{L\}$
    $\mathcal{X} \leftarrow \mathcal{X} \setminus C$
  **end if**
**end for**
append *myid* at the tail of each $L \in \mathcal{S}$
```

Recall that $\mathcal{R}$ denotes the set of all ordered sequences $L$ of IDs received at this round, and $\mathcal{X}$ denotes the collection of all sets of $k - t$ elements in $\mathcal{I}$, where $\mathcal{I}$ is the set of all collected IDs at this round, including the fake IDs in $\{-1, -2, \ldots, -k + t\}$.

For each sequence $L \in \mathcal{R}$ the algorithm decides whether to include $L$ in $\mathcal{S}$ or not. For this purpose, the algorithm checks whether there is a set $X \subseteq \mathcal{I}$ with $k - t$ elements that does not intersect $L$. If this is the case, then such a list $L$ is added to $\mathcal{S}$. The intuition is that $L$ (of length $t - 1$ at round $t$) may potentially be extended by adding the current node, plus $k - t$ other nodes, to form a cycle $C_k$. These nodes can be nodes whose IDs were collected before (but not in $L$), or nodes whose existence is only postulated (hence, the use of fake IDs).

For instance, Figure 3 displays the case where

$$L = (y_1, y_2, \ldots, y_{t-1})$$

and

$$X = \{x_1, x_2, \ldots, x_{k-t}\}$$

are considered by some node $z$, depicted as a star $\star$ on the figure. The nodes in $L$ are depicted in light grey, while the nodes in $X$ are depicted in black. Note that $X$ is a set, and the ordering of the $x_i$'s on the figure is arbitrary. The list $L$ is placed in $\mathcal{S}$, because there are $k - t$ nodes, i.e., those in $X$, that can potentially form a $k$-cycle with $z$ and all the nodes in $L$.

Importantly, all sets $X$ in $C = \{X \in \mathcal{X} : X \cap L = \emptyset\}$ are then removed from $\mathcal{X}$ so they are not considered again for another list $L'$. The intuition is that if there is a $k$-cycle formed by the nodes in $L' \cup \{z\} \cup X$ for some list $L' \in \mathcal{R}$ where $z$ is the actual node, then the nodes in $L \cup \{z\} \cup X$ also form a $k$-cycle, and therefore there is no need to keep track of both $L$ and $L'$. Therefore, as soon as $L$ has been identified, all "witness sets" $X \in C$ that qualify $L$ for being forwarded at the next round can safely be removed from $\mathcal{X}$.

For instance, considering again the example of Figure 3, as long as $L$ has been placed in $\mathcal{S}$, the set $X$ can be removed from $\mathcal{X}$, since it could only be used to identify another sequence

$$L' = (y'_1, y'_2, \ldots, y'_{t-1})$$

potentially forming a $k$-cycle with $X$, while we are not interested in enumerating all cycles $C_k$ but just in determining whether there is one.

Note here the role of the fake IDs that were added to $\mathcal{I}$:

- First, observe that the first sequence $L \in \mathcal{R}$ that is considered in the for-loop is necessarily placed in $\mathcal{S}$ (the order in which these sequences are enumerated is arbitrary). Indeed,
$$X = \{-1, -2, \ldots, -k + t\}$$
is originally in $\mathcal{X}$, and for sure does not intersect $L$.
- Second, notice that the fact that all sets $X \in \mathcal{C}$ can be safely removed from $\mathcal{X}$ is not necessarily obvious if $X$ contains fake IDs, as such an $X$ does not fully specify the cycle. Nevertheless, we shall show that those sets can still be removed without preventing the algorithm to detect a cycle, if there is one.

To give a more precise intuition of the use of fake IDs in our algorithm, let us consider a cycle of length 9, where node IDs are from 1 to 9, consecutively around the cycle (hence, the edges are $\{1, 2\}, \ldots, \{8, 9\}$ and $\{1, 9\}$. Let us assume that one wants to detect $C_9$, starting from the edge $\{1,9\}$. Then, in particular, when node 3 receives the sequence $(1, 2)$ from node 2, we want that node to send the sequence $(1, 2, 3)$ to node 4. This is the role of Lines 17–25 in Algorithm 3, where $\mathcal{R}$ contains just the sequence $(1, 2)$. In Algorithm 3, if one would not add fake IDs to $\mathcal{I}$, then $\mathcal{I} = 1, 2$, and $\mathcal{X}$ would become empty, as one cannot construct sequences of length $k - t = 9 - 3 = 6$ using IDs from $\mathcal{I}$. As a consequence, $\mathcal{C}$ would also be empty, as it results from an intersection with the empty set, and we would not add $(1, 2)$ to $\mathcal{S}$. It would follow that node 3 does not send any sequence. Instead, if we add the fake IDs $-1, \ldots, -6$ to $\mathcal{I}$, then the sequence $(-1, \ldots, -6)$ is in $\mathcal{X}$, and since $(1, 2)$ is disjoint with $\{-1, \ldots, -6\}$, the sequence $(1, 2)$ is added to $\mathcal{S}$, and the sequence $(1, 2, 3)$ will be sent, as desired. The same reasoning applies all around the cycle.

## 3.3 Analysis of Our Algorithm

We start by proving the correctness of the algorithm, before analyzing its performances.

Lemma 3.2. *For every $t = 1, \ldots, \lfloor \frac{k}{2} \rfloor$, every sequence $L$ contained in a non-empty set $\mathcal{S}$ sent at round $t$ is composed of $t$ distinct IDs and forms a simple path in the graph with one extremity equal to the sender, and the other equal to $u$ or $v$.*

Proof. By induction on $t$. The lemma trivially holds for $t = 1$ (cf. Instruction 3). All messages set to be sent at round $t + 1$ are constructed by appending the ID of the current node to sequences $L$ received at round $t$ (cf. Instruction 25), and these sequences $L$ do not contain the ID of the current node (cf. Instruction 13). Therefore, every sequence sent at round $t + 1$ is composed of $t + 1$ distinct IDs. Moreover, by induction, a sequence $L$ received at round $t$ by a node $x$ from a neighboring node $y$ forms a simple path in the graph with one extremity equal to $y$. Therefore, as long as $\mathrm{ID}(x) \notin L$ (which is guaranteed by Instruction 13), the sequence $L \cup \{\mathrm{ID}(x)\}$ forms a simple path in the graph with one extremity equal to $x$. The other extremity remains unchanged, and thus equal to $u$ or $v$. □

Lemma 3.3. *For any graph $G$, and every edge $e = \{u, v\}$ of $G$, Algorithm 3 running on $G$ satisfies that all nodes output accept if and only if there are no $C_k$ passing through the edge $e$.*

Proof. Let us assume that some node $w$ outputs reject, and let us show that there is indeed a $k$-cycle passing through $e$. From Instruction 39, this node $w$ satisfies that there exist two sequences $L_1, L_2 \in \mathcal{R}$ such that

$$|L_1 \cup L_2 \cup \{\mathrm{ID}(w)\}| = k.$$

By Lemma 3.2, both sequences are simple paths of length at most $\lfloor \frac{k}{2} \rfloor$ from $u$ or $v$ to a neighbor of $w$. Let

$$L_1 = (x_1, x_2, \ldots, x_\ell),$$

and

$$L_2 = (y_1, y_2, \ldots, y_m),$$

where $\ell \leq \lfloor k/2 \rfloor$ and $m \leq \lfloor k/2 \rfloor$.

- If $k$ is odd, then $|L_1 \cup L_2 \cup \{\mathrm{ID}(w)\}| = k$ implies that $\ell = m = \lfloor k/2 \rfloor$, $w$ is distinct from every $x_i$ and every $y_j$, and every $x_i$ is distinct from every $y_j$, $i = 1, \ldots, \ell$, $j = 1, \ldots, m$. In particular, since $x_1 \neq y_1$, we have $\{x_1, y_1\} = \{u, v\}$. It follows that

  $$(x_1, x_2, \ldots, x_\ell, w, y_m, y_{m-1}, \ldots, y_1)$$

  is a $k$-cycle passing through $e$.
- If $k$ is even, then we claim that

  $$(L_1 \in \mathcal{S} \text{ and } L_2 \notin \mathcal{S}) \text{ or } (L_1 \notin \mathcal{S} \text{ and } L_2 \in \mathcal{S}).$$

  Indeed, $w$ tries to construct a cycle from sequences constructed in Line 36 of the algorithm. Such sequences can contain either sequences in $\mathcal{S}$ of length $k/2$ constructed by $w$ itself (such sequences contain $\mathrm{ID}(w)$), or sequences of length $k/2 - 1$ received by $w$ from other nodes. Let us consider two distinct sequences $L$ and $L'$ in $\mathcal{S}$. Since they are both of length $k/2$, and since they both contain $\mathrm{ID}(w)$, we have $|L \cup L' \cup \{\mathrm{ID}(w)\}| \leq k - 1$. Thus, at least one sequence $L_1$ or $L_2$ must not be contained in $\mathcal{S}$. Moreover, the sequences received at round $k/2 - 1$ are of length $k/2 - 1$, and thus $|L_1 \cup L_2 \cup \{\mathrm{ID}(w)\}| = k$ implies that at least one of these two sequences is necessarily in $\mathcal{S}$. Hence, the claim holds. So, let us now assume, w.l.o.g., that $L_1 \in \mathcal{S}$ and $L_2 \notin \mathcal{S}$. It follows that $L_1$ is of length $k/2$ and contains $\mathrm{ID}(w)$, and that $L_2$ is of length $k/2$ without containing $\mathrm{ID}(w)$. The equality $|L_1 \cup L_2 \cup \{\mathrm{ID}(w)\}| = k$ then implies that $w$ is distinct from every $x_i$ and every $y_j$, and every $x_i$ is distinct from every $y_j$, $i = 1, \ldots, \ell$, $j = 1, \ldots, m$. In particular, since $x_1 \neq y_1$, we have $\{x_1, y_1\} = \{u, v\}$. It follows that

  $$(x_1, x_2, \ldots, x_\ell, w, y_m, y_{m-1}, \ldots, y_1)$$

  is a $k$-cycle passing through $e$.

Therefore, for both cases, $k$ even or odd, the existence of a node that outputs reject implies the existence of a cycle passing through $e$.

Conversely, let us assume that there is a $k$-cycle passing through $e$, and let us show that at least one node detects that cycle (i.e., outputs reject). Observe that a modified version of the algorithm where the construction of $\mathcal{S}$ in the for-loop of Instruction 18 is replaced by

$$\mathcal{S} \leftarrow \mathcal{R}$$

clearly detects the cycle. Indeed, at each round $t$, all the possible paths of length $t$ from the edge to the actual node are transmitted. However, there can be too many such paths, and transmitting all of them would not fit with the constraints of the CONGEST model. Hence, some paths are discarded by Algorithm 3. Yet, we show that Algorithm 3 keeps sufficiently many options for detecting the cycle. Let us fix some round $t \in \{2, \dots, \lfloor \frac{k}{2} \rfloor\}$, and a node $w$. Let us consider a discarded sequence

$$L = (x_1, x_2, \dots, x_{t-1})$$

at $w$, and let us assume that the existing $k$-cycle precisely includes that sequence of nodes, that is, the $k$-cycle is of the form

$$x_1, x_2, \dots, x_{t-1}, w, y_1, \dots, y_{k-t}$$

where $\{x_1, y_{k-t}\} = \{u, v\}$. Since the sequence $L$ has been discarded, we have

$$\{X \in \mathcal{X} : X \cap L = \emptyset\} = \emptyset$$

where $\mathcal{X}$ is the collection of all remaining sets $X$ at the time where the sequence $L$ is considered in the for-loop of Line 18. (Recall that, initially, every $X \in \mathcal{X}$ is composed of $k - t$ IDs in $\mathcal{I}$, where $\mathcal{I}$ is the set of known IDs, including the "fake" IDs $-1, \dots, -k + t$.) This implies that all sets $X \in \mathcal{X}$ that do not intersect $L$ have been removed from $\mathcal{X}$ when considering other sequences in previous executions of the for-loop. Let $J$ be the set of indices $j$ such that $y_j$ is known by $w$, i.e., such that $\mathrm{ID}(y_j) \in \mathcal{I}$. Among others, all sets

$$X = \{y_j, j \in J\} \cup F$$

where $F \subseteq \{-1, \dots, -k + t\}$ and $|F| = k - t - |J|$ have been removed when considering some other sequence

$$L' = (z_1, z_2, \dots, z_{t-1})$$

during a previous execution of the for-loop. In particular, $L' \cap \{y_j, j \in J\} = \emptyset$. By the definition of $J$, it follows that

$$\{z_1, z_2, \dots, z_{t-1}\} \cup \{y_1, \dots, y_{k-t}\} = \emptyset,$$

since $\mathrm{ID}(z_i) \in \mathcal{I}$ for every $i = 1, \dots, t - 1$. Therefore, there exists another cycle,

$$z_1, z_2, \dots, z_{t-1}, w, y_1, \dots, y_{k-t}$$

where $\{z_1, y_{k-t}\} = \{u, v\}$. Therefore, Algorithm 3 satisfies that, at every round $t \in \{2, \dots, \lfloor \frac{k}{2} \rfloor\}$, if $w$ belongs to a cycle

$$x_1, x_2, \dots, x_{t-1}, w, y_1, \dots, y_{k-t}$$

passing through $e = \{x_1, y_{k-t}\}$, and $w$ receives the sequence $x_1, x_2, \dots, x_{t-1}$, then it is guaranteed that if $w$ does not send the sequence $(x_1, x_2, \dots, x_{t-1}, w)$ to $y_1$, then $w$ necessarily sends another sequence $(z_1, z_2, \dots, z_{t-1}, w)$ to $y_1$ where

$$z_1, z_2, \dots, z_{t-1}, w, y_1, \dots, y_{k-t}$$

is a cycle passing through $e = \{z_1, y_{k-t}\}$. Therefore, the nodes antipodal to $e$ (that is, the nodes at distance $\lceil \frac{k}{2} \rceil - 1$ from $e$ in the cycle) will detect a cycle at round $\lfloor \frac{k}{2} \rfloor$ and will output reject, as desired. □

In the next lemma, we show that, for a fixed $k$, the messages exchanged during the execution of Algorithm 3 are of constant size.

Lemma 3.4. *For every $t = 1, \dots, \lfloor \frac{k}{2} \rfloor$, every message sent by nodes at round $t$ is composed of at most $(k - t + 1)^{t-1}$ ordered sequences of $t$ IDs.*

Proof. For the ease of notation, we rephrase the statement of the lemma as: For every $t = 0, \dots, \lfloor \frac{k}{2} \rfloor - 1$, every message sent by nodes at round $t + 1$ is composed of at most $(k - t)^t$ ordered sequences of $t + 1$ IDs. Let us fix $t \in \{0, \dots, \lfloor \frac{k}{2} \rfloor - 1\}$ and a node $w$, and let us focus on round $t + 1$. For $i = 0, \dots, t$, let us then define Property $P_i$ as:

for every set of $t-i$ IDs, $w$ sends at most $(k-t)^i$ sequences that contain that set.

Note that Property $P_t$ establishes the lemma.

Property $P_0$ stating that, for every set of $t$ IDs, $w$ sends at most one sequence that contains that set, follows from the fact that, in the construction of $\mathcal{S}$ in the for-loop of Instruction 18, this set will be sent only once, in one of all its possible orderings.

Let us assume that $P_{i-1}$ holds, and let us establish $P_i$. Consider the case where, during the execution of the for-loop of Instruction 18, we already added $(k-t)^i$ sequences to $\mathcal{S}$ containing the same $t-i$ elements $x_1, x_2, \dots, x_{t-i}$. That is, $\mathcal{S}$ contains

$$\begin{gathered}\{x_1, x_2, \dots, x_{t-i}, y_{1,1}, y_{1,2}, \dots, y_{1,i}\}\\ \{x_1, x_2, \dots, x_{t-i}, y_{2,1}, y_{2,2}, \dots, y_{2,i}\}\\ \vdots\\ \{x_1, x_2, \dots, x_{t-i}, y_{(k-t)^i,1}, y_{(k-t)^i,2}, \dots, y_{(k-t)^i,i}\}.\end{gathered}$$

After these sequences have been added to $\mathcal{S}$, the remaining sequences in $\mathcal{X}$ must contain at least one element of each such sequences. That is, for every $X \in \mathcal{X}$, we have

$$(x_1 \in X) \vee (x_2 \in X) \vee \dots \vee (x_{t-i} \in X)$$

or

$$\bigwedge_{j=1}^{(k-t)^i} \Big((y_{j,1} \in X) \vee (y_{j,2} \in X) \vee \dots \vee (y_{j,i} \in X)\Big). \tag{1}$$

Indeed, if a sequence does not contain $x_1, x_2, \dots, x_{t-i}$, then it should contain an element $y_{a,b}$ for each sequence. We can now apply the induction hypothesis to show that the same element $y_{a,b}$ cannot appear more than $(k-t)^{i-1}$ times. Indeed, the sequence $x_1, x_2, \dots, x_{t-i}, y_{a,b}$ is of length $t-(i-1)$, and therefore, by induction, it cannot appear more than $(k-t)^{i-1}$ times.

Therefore, since there are $(k-t)^i = (k-t)^{i-1} \cdot (k-t)$ sequences in $\mathcal{S}$ containing the same $t-i$ elements $x_1, x_2, \dots, x_{t-i}$, Equation (1) implies that a sequence $X \in \mathcal{X}$ must contain $k-t$ different elements. However, sequences in $X$ are of size $k-t-1$. Therefore, the formula in Equation (1) cannot be satisfied. It follows that, for every $X \in \mathcal{X}$, we have

$$(x_1 \in X) \vee (x_2 \in X) \vee \dots \vee (x_{t-i} \in X).$$

Let us now consider another sequence

$$L = (x_1, x_2, \dots, x_{t-i}, z_1, \dots, z_i)$$

taken from $\mathcal{R}$. This sequence will not be added to $\mathcal{S}$, because every sequence $X \in \mathcal{X}$ contains at least one element from $\{x_1, x_2, \dots, x_{t-i}\}$, which implies that $L \cap X \neq \emptyset$. □

*Remark.* By Lemma 3.4, we get that, for detecting $k$-cycles, the number of IDs sent by a node at each round is at most $(\frac{k}{2})^{k/2+O(1)}$. Computing these IDs is therefore time-consuming, but observe that, since computing the length of the longest cycle in a graph is hard, it is unlikely (unless P = NP) that one can detect $k$-cycles in time poly($k$). Moreover, the sequential algorithm in Reference [27] for detecting $C_k$, based on a "method of representative" similar to the one used in Algorithm 3, also uses roughly $k!$ "representatives," i.e., in our setting, at least roughly $k!$ sequences of $O(k)$ IDs. In fact, we do not know whether it could be possible to send significantly less sequences at each round. Note that there is a general exponential lower bound on the size of the sets for the "method of representative" (see References [14, 27]). However, it is not clear whether this lower bound applies to the specific setting of this article, i.e., finding $k$-cycles in graphs. However, there are graphs in which our algorithm, i.e., Algorithm 3, does produce exponentially many sequences of $k$

IDs to be sent by a node at some round—that is, the bound given by Lemma 3.4 is essentially tight. For instance, let us consider a graph in which the edge $e = \{u, v\}$ is part of a multi-layered graph with $\lceil k/2 \rceil + 1$ layers $\mathcal{L}_0, \mathcal{L}_1, \dots, \mathcal{L}_{\lceil k/2 \rceil}$ where, $\mathcal{L}_0 = \{u, v\}$, and, for $i \geq 1$, $\mathcal{L}_i$ is a 2-node stable, and $\mathcal{L}_i$ and $\mathcal{L}_{i-1}$ form a complete bipartite graph. Let us assume that nodes in this multi-layered graph are labeled with IDs from 1 to $2(1 + \lceil k/2 \rceil)$ in increasing order, layer-by-layer, starting from $\mathcal{L}_0$, then $\mathcal{L}_1$, and so on until $\mathcal{L}_{\lceil k/2 \rceil}$. In this graph, nodes at level $\mathcal{L}_1$ receive two sequences at round 1. Then, by induction on the number of rounds, let us assume that, at the end of round $t$, nodes with IDs $2t + 1$ and $2t + 2$ in layer $\mathcal{L}_t$ receive $2^t$ different sequences of IDs, of the form $(x_1, \dots, x_t)$ where $x_i \in \{2i - 1, 2i\}$ for $i = 1, \dots, t$. By line 16 of Algorithm 3, it follows from the induction hypothesis that, for every received sequence $L = (x_1, \dots, x_t)$, the set $\mathcal{X}$ contains the set $Y = \{y_1, \dots, y_t, -1, \dots, -k + 2t\}$ as subset, where $y_i = 4i - x_i - 1$ for $i = 1, \dots, t$. Since $L$ is the only sequence in $\mathcal{R}$ that is disjoint with $Y$, it follows that $L$ will be added to $\mathcal{S}$ by Algorithm 3. Thus, each of the two nodes $2t + 1$ and $2t + 2$ at layer $\mathcal{L}_t$ will append its ID to $L$, and will send it to the next layer $\mathcal{L}_{t+1}$, in which the nodes will receive $2^{t+1}$ sequences of the form $(x_1, \dots, x_{t+1})$ such that $x_i \in \{2i - 1, 2i\}$ for $i = 1, \dots, t + 1$. Therefore, in particular, $2^{\lceil k/2 \rceil - 1}$ sequences of IDs are sent between layer $\mathcal{L}_{\lceil k/2 \rceil - 1}$ and layer $\mathcal{L}_{\lceil k/2 \rceil}$.

## 3.4 Proof of Theorem 3.1

Let us first compute the probability of detecting a cycle in a network that is $\epsilon$-far from being $C_k$-free. We exploit the fact that, in such a network, there must be many edge-disjoint copies of $C_k$, as stated below:

Lemma 3.5 [21]. *Let $H$ be any graph. Let $G$ be an $m$-edge graph that is $\epsilon$-far from being $H$-free. Then $G$ contains at least $\epsilon m/|E(H)|$ edge-disjoint copies of $H$.*

Hence, a graph $G$ that is $\epsilon$-far from being $C_k$-free contains at least $\epsilon m/k$ edge-disjoint copies of $C_k$, i.e., $\epsilon m$ edges belong to edge-disjoint cycles.

To compute the probability of having a unique edge of minimum rank, we exploit the isolation lemma.

Lemma 3.6 (Isolation Lemma [28]). *Let $n$ and $N$ be positive integers, and let $\mathcal{F}$ be an arbitrary family of subsets of the universe $\{1, \dots, n\}$. Suppose each element $x \in \{1, \dots, n\}$ in the universe receives an integer weight $w(x)$, each of which is chosen independently and uniformly at random from $\{1, \dots, N\}$. The weight of a set $S \in \mathcal{F}$ is defined as $w(S) = \sum_{x \in S} w(x)$. Then, with probability at least $1 - n/N$, there is a unique set in $\mathcal{F}$ that has minimum weight among all sets of $\mathcal{F}$.*

The next lemma is a direct consequence of the Isolation Lemma.

Lemma 3.7. *The probability that there is a unique edge with minimum rank after the execution of Phase 1 is at least $1/2$.*

Proof. Set $n = |V|$, $\mathcal{F} = E$, and $N = 2n$, and then apply Lemma 3.6. □

Let $G$ be a graph that is $\epsilon$-far from being $C_k$-free, and let $\mathcal{E}$ be the event "there is a unique edge with minimum rank after the execution of Phase 1, and this edge belongs to a $k$-cycle." Combining the previous two lemmas, we get that

$$\Pr[\mathcal{E}] \geq \epsilon/2.$$

Now, if event $\mathcal{E}$ holds, then, by Lemma 3.3, at least one node will output reject, as desired. To boost the probability of detecting a cycle in a graph that is $\epsilon$-far from being $C_k$-free, we repeat the whole process $\frac{2 \ln 3}{\epsilon}$ times. In this way, the probability that $\mathcal{E}$ holds in at least one of these repetitions is at least $2/3$ as desired.

By Lemma 3.4, each repetition of the whole process of executing Phases 1 and 2 requires a constant number of rounds. This completes the proof of Theorem 3.1. □

## 4 CONCLUSION

In this article, we have proved that, for every $k \geq 3$, there exists a 1-sided error distributed property testing algorithm for $C_k$-freeness, performing in $O(1/\epsilon)$ rounds. We mention hereafter some possible directions for further work.

It was proved in Reference [21] that, for every graph pattern $H$ with at most 4 nodes, there exists a distributed property testing algorithm for $H$-freeness, performing in constant number of rounds. The question of whether a distributed property testing algorithm for $H$-freeness exists for every arbitrarily large pattern $H$ was left open in [21]. The techniques in this article do not seem to extend to arbitrary patterns. To see why, consider $H$ as a $k$-cycle with a chord between two nodes. The pruning technique in Algorithm 3 of discarding some sequences of nodes is oblivious to the neighborhood of the nodes in these sequences. Hence, while Algorithm 3 makes sure to keep at least one sequence corresponding to a cycle, if such a cycle exists, it may well discard the sequence corresponding to the cycle in $H$ and keep a sequence without a chord. It was also pointed out in Reference [21] that their techniques do not seem to extend to *induced* subgraphs.[2] The same apparently holds for the techniques in this article. The reasons are the same as for detecting a given graph pattern $H$. Indeed, our pruning mechanism is not adapted to detect an induced cycle. It may well discard a sequence corresponding to the induced cycle and keep a sequence with chords.

We believe that proving or disproving the existence of distributed property testing algorithms for $H$-freeness, as a subgraph or as an induced subgraph, is a potentially challenging but definitely rewarding issue whose study is susceptible to shed new light on the CONGEST model, and, more generally, to improve our understanding of local distributed computing in presence of bandwidth limitation. The most recent results in this field (see Reference [15]) extend the results in this article by establishing that $H$-freeness can be tested in $O(1/\epsilon)$ rounds for every graph pattern $H$ composed of a forest $F$, an edge $e$, and arbitrary connections between the extremities of $e$ and the nodes of $F$. $K_5$ is the smallest graph pattern that cannot be described as a "forest plus one edge." Proving or disproving that $K_5$-freeness can be tested in $O(1)$ rounds in the CONGEST model is therefore quite an intriguing problem.

## REFERENCES

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

[2] Noga Alon, Tali Kaufman, Michael Krivelevich, and Dana Ron. 2008. Testing triangle-freeness in general graphs. *SIAM J. Disc. Math.* 22, 2 (2008), 786–819.

[3] Noga Alon and Asaf Shapira. 2006. A characterization of easily testable induced subgraphs. *Combin., Prob. Comput.* 15, 6 (2006), 791–805.

[4] Alkida Balliu, Gianlorenzo D'Angelo, Pierre Fraigniaud, and Dennis Olivetti. 2017. What can be verified locally? In *Proceedings of the 34th Symposium on Theoretical Aspects of Computer Science (STACS'17).*

[5] Azzedine Boukerche and Carl Tropper. 1998. A distributed graph algorithm for the detection of local cycles and knots. *IEEE Trans. Parallel Distrib. Syst.* 9, 8 (1998), 748–757.

[6] Zvika Brakerski and Boaz Patt-Shamir. 2011. Distributed discovery of large near-cliques. *Distrib. Comput.* 24, 2 (2011), 79–89.

[7] Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. 2016. Fast distributed algorithms for testing graph properties. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC'16) (LNCS)*, Vol. 9888. Springer, 43–56.

[2] *A graph $H$ is an* induced *subgraph of a graph $G$ iff $V(H) \subseteq V(G)$ and $E(H) = E(G[V(H)])$, i.e., for every $(u, v) \in V(H) \times V(H)$, we have $\{u, v\} \in E(H) \iff \{u, v\} \in E(G)$. (In other words, $H$ is isomorphic to the subgraph of $G$ induced by the nodes in $H$.)*

[8] Pranay Chaudhuri. 1999. A self-stabilizing algorithm for detecting fundamental cycles in a graph. *J. Comput. Syst. Sci.* 59, 1 (1999), 84–93.
[9] Pranay Chaudhuri. 2002. An optimal distributed algorithm for finding a set of fundamental cycles in a graph. *Comput. Syst. Sci. Eng.* 17, 1 (2002), 41–47.
[10] David Conlon and Jacob Fox. 2012. Graph removal lemmas. Retrieved from *CoRR* abs/1211.3487 (2012).
[11] Artur Czumaj, Oded Goldreich, Dana Ron, C. Seshadhri, Asaf Shapira, and Christian Sohler. 2014. Finding cycles and trees in sublinear time. *Rand. Struct. Algor.* 45, 2 (2014), 139–184.
[12] Yuval Emek, Christoph Pfister, Jochen Seidel, and Roger Wattenhofer. 2014. Anonymous networks: Randomization = 2-hop coloring. In *Proceedings of the 33rd ACM Symposium on Principles of Distributed Computing.* 96–105.
[13] Paul Erdös, Peter Frankl, and Vojtech Rödl. 1986. The asymptotic number of graphs not containing a fixed subgraph and a problem for hypergraphs having no exponent. *Graphs. Combin.* 2, 1 (1986), 113–121.
[14] Paul Erdős, András Hajnal, and J. W. Moon. 1964. A problem in graph theory. *Amer. Math. Month.* 71, 10 (1964), 1107–1110.
[15] Guy Even, Orr Fischer, Pierre Fraigniaud, Tzlil Gonen, Reut Levi, Moti Medina, Pedro Montealegre, Dennis Olivetti, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. 2017. Three notes on distributed property testing. In *Proceedings of the 31st International Symposium on Distributed Computing (DISC'17).* 15:1–15:30. DOI: https://doi.org/10.4230/LIPIcs.DISC.2017.15
[16] Laurent Feuilloley and Pierre Fraigniaud. 2015. Randomized local network computing. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'15).* 340–349.
[17] Laurent Feuilloley and Pierre Fraigniaud. 2016. Survey of distributed decision. *Bull. EATCS* 119 (2016), 41–65.
[18] Laurent Feuilloley, Pierre Fraigniaud, and Juho Hirvonen. 2016. A hierarchy of local decision. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP'16).* 118:1–118:15.
[19] Pierre Fraigniaud, Mika Göös, Amos Korman, Merav Parter, and David Peleg. 2014. Randomized distributed decision. *Distrib. Comput.* 27, 6 (2014), 419–434.
[20] Pierre Fraigniaud, Amos Korman, and David Peleg. 2013. Towards a complexity theory for local distributed computing. *J. ACM* 60, 5 (2013), 35:1–35:26.
[21] Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. 2016. Distributed testing of excluded subgraphs. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC'16) (LNCS)*, Vol. 9888. Springer, 342–356.
[22] Oded Goldreich (Ed.). 2010. *Property Testing—Current Research and Surveys.* Vol. LNCS 6390. Springer.
[23] Oded Goldreich and Dana Ron. 2002. Property testing in bounded degree graphs. *Algorithmica* 32, 2 (2002), 302–343.
[24] Oded Goldreich and Luca Trevisan. 2003. Three theorems regarding testing graph properties. *Rand. Struct. Algor.* 23, 1 (2003), 23–57.
[25] Mika Göös and Jukka Suomela. 2016. Locally checkable proofs in distributed computing. *Theor. Comput.* 12, 1 (2016), 1–33.
[26] Amos Korman, Shay Kutten, and David Peleg. 2010. Proof labeling schemes. *Distrib. Comput.* 22, 4 (2010), 215–233.
[27] Burkhard Monien. 1985. How to find long paths efficiently. In *Analysis and Design of Algorithms for Combinatorial Problems.* North-Holland Math. Stud., Vol. 109. North-Holland, Amsterdam, 239–254.
[28] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. 1987. Matching is as easy as matrix inversion. In *Proceedings of the 19th ACM Symposium on Theory of Computing.* 345–354. DOI: https://doi.org/10.1145/28395.383347
[29] Moni Naor and Larry J. Stockmeyer. 1995. What can be computed locally? *SIAM J. Comput.* 24, 6 (1995), 1259–1277.
[30] Gabriele Oliva, Roberto Setola, Luigi Glielmo, and Christoforos N. Hadjicostis. 2016. Distributed cycle detection and removal. *IEEE Trans. Cont. Netw. Syst.* PP, 99 (2016).
[31] David Peleg. 2000. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, Philadelphia.
[32] Rodrigo Caetano Rocha and Bhalchandra D. Thatte. 2015. Distributed cycle detection in large-scale sparse graphs. In *Proceedings of Simpósio Brasileiro de Pesquisa Operacional (SBPO'15).* 1–11.

Received September 2017; revised September 2018; accepted March 2019