Distributed Detection of Cycles

Pierre Fraigniaud*
Institut de Recherche en Informatique Fondamentale
CNRS and University Paris Diderot
France
pierre.fraigniaud@irif.fr

Dennis Olivetti†
Gran Sasso Science Institute
L'Aquila, Italy
dennis.olivetti@gssi.infn.it

ABSTRACT

Distributed property testing in networks has been introduced by Brakerski and Patt-Shamir (2011), with the objective of detecting the presence of large dense sub-networks in a distributed manner. Recently, Censor-Hillel et al. (2016) 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. (2016) 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 paper, 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.

1 INTRODUCTION

1.1 Context

1.1.1 Property Testing. The objective of (sequential) *property testing* [21] is the design of efficient mechanisms 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 like triangles, cycles, cliques, etc. (see, e.g., [2, 3, 11, 22]). A property testing mechanism, a.k.a. *tester*, is a centralized algorithm $\mathcal{A}$ which is given the ability to probe nodes with queries of the form $\deg(i)$ returning the degree of the ith node, and $\mathrm{adj}(i,j)$ returning the identity of the jth neighbor of the ith node. Beside 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 finds its main interest when the problem is relaxed by simply 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 ϵ-far from satisfying a graph property $\mathcal{P}$ if adding and/or removing at most ϵm 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 ϵ-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 which are nearly satisfying $\mathcal{P}$ but not quite, the algorithm can output either ways. Hence, a tester for $\mathcal{P}$ is a mechanism 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.

A tester is 1-sided error if

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

1.1.2 Distributed Property Testing. Distributed property testing has been introduced in [6], and recently revisited in [7, 20]. 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. One says that $\mathcal{A}$ 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.

In this paper, we are focussing 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.

*Additional support from ANR Project DESCARTES and Inria Project GANG.
†Additional support from ANR Project DESCARTES.

SPAA '17, July 24-26, 2017, Washington DC, USA

ACM ISBN 978-1-4503-4593-4/17/07…$15.00
https://doi.org/10.1145/3087556.3087571

It has been shown in [7] that, in the classical CONGEST model[1] for distributed computing [29], there exists a distributed property testing algorithm for triangle-freeness performing in $O(1/\epsilon^2)$ rounds. This result has been extended in [20] 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 [7, 20] do not extend to larger cycles. Indeed, using explicit constructions of so-called Behrend graphs, it was proved in [20] 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 $1/\epsilon$. The existence of distributed property testing algorithms for C_k-freeness performing in a constant number of rounds was left 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, our algorithm for testing the existence of a k-cycle passing through a given edge e does not rely on the ϵ-farness assumption. That is, even if there is just a single k-cycle passing through e, that cycle will be detected by our algorithm.

After the acceptance of the paper, 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 [26]. Our technique for detecting cycles can also be viewed as a distributed implementation of this combinatorial lemma.

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 δn^k (induced) copies of H can be transformed into an (induced) H-free graph by deleting at most ϵn^2 edges. This lemma was first proved for the case $k = 3$, and later generalized to subgraphs H of any 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 δ is a fast growing function of ϵ and k. The growth of the function was later improved in [3] under some assumptions. For more details on the graph removal lemma, see [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 [22] 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 [11]. The same paper presents a tester requiring $\widetilde{O}(poly(1/\epsilon)\sqrt{n})$ queries in arbitrary graphs, and another tester requiring $\widetilde{O}(poly(d^k/\epsilon)\sqrt{n})$ 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 [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 [6], and fully formalized in [7].

The authors of that 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

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

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 [23]).

The authors of [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

$$O(poly(1/\epsilon \log(n/\epsilon)))$$

rounds. Their work was inspired by [6], where a constant-time distributed algorithm for finding a linear-size ϵ-near clique is proposed, under the assumption that the graph contains a linear-size ϵ^3-near clique. (An ϵ-near clique is a set of nodes where all but an ϵ fraction of pairs of nodes have edges between them).

The result in [7] regarding testing triangle-freeness was extended in [20], 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, [20] provides a proof that the approach in [7, 20] fails to test C_k-freeness in a constant number of rounds, whenever $k \geq 5$.

1.3.3 Distributed Decision. Distributed property testing fits into the larger framework of *distributed decision*. The seminal paper [27] 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 variant of distributed decision, in the deterministic setting (see, e.g., [19]), the anonymous setting (see, e.g., [12]), the probabilistic setting (see, e.g., [15, 18]), the non-deterministic setting (see, e.g., [24, 25]), and even beyond (see, e.g., [4, 17]). We refer to [16] 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. We refer to, e.g., [5, 8, 9, 28] for cycle detection in message passing, bulk-synchronization, self-stabilizing, and other models of parallel and distributed computing.

2 MODEL AND DEFINITIONS

2.1 The CONGEST Model

We are considering the classical CONGEST model for distributed network computing [29]. 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 paper, we are mostly interested in solving tasks locally. Hence, we are mainly focussing on the design of algorithms performing in a constant number of rounds in the CONGEST model.

2.2 Distributed Property Testing

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

A distributed property testing algorithm for $\mathcal{P}$ is a randomized algorithm which 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 $\{\text{accept}, \text{reject}\}$. The algorithm is correct if and only if the following two conditions are satisfied.

- if G satisfies $\mathcal{P}$, then

$$\Pr[\text{every node outputs accept}] \geq 2/3;$$

- if G is ϵ-far from satisfying $\mathcal{P}$, then

$$\Pr[\text{at least one node outputs reject}] \geq 2/3.$$

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

$$\Pr[\text{every node outputs accept}] = 1.$$

2.2.2 C_k-Freeness. 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 (resp., edges) is denoted by $V(G)$ (resp., $E(G)$). Throughout the paper, $n = |V(G)|$, 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

- if G is C_k-free, then

$$\Pr[\text{every node outputs accept}] = 1.$$

- if G contains a cycle C_k then

$$\Pr[\text{at least one node outputs reject}] \geq 2/3.$$

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

Every edge is assigned to its extremity with smallest identity. Every node picks a random integer $r(e) \in [1, m^2]$ for each edge e that is assigned to it, called the rank of e. (By construction, $O(\log n)$ random bits per edge are sufficient). For each edge e, the extremity of e which computed its rank sends $r(e)$ to the other extremity. Then, 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 ever become aware of during the execution of the algorithm (again, ties are broken arbitrarily), in a way similar to the prioritized search in [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.

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 which 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.

For instance, to detect a C_5 passing through $e = \{u, v\}$, nodes u and v send their IDs to their neighbors at Round 1 (see Fig. 1). A node may thus receive 0, 1, or 2 IDs depending on whether it is adjacent to none, one, or both nodes u and v. Each node x which received at least one of these IDs has now a set $\mathcal{R}$ of sequences of the form $(\mathrm{ID}(w))$ where $w \in \{u, v\}$. Such node appends its own ID to each sequence, and sends the resulting set of sequences to all its neighbors at Round 2. 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, detects the presence of the cycle (u, x, z, y, v).

This "append-and-forward" technique can be trivially extended to detect 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.

The main concern of our algorithm is 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 Fig. 1, nodes x and y both receive $\mathrm{ID}(u)$ and $\mathrm{ID}(v)$ at the first round. If x forwards only the sequence $(\mathrm{ID}(u), \mathrm{ID}(x))$, and y forwards only the sequence $(\mathrm{ID}(u), \mathrm{ID}(y))$, the 5-cycle will not be detected by z.

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.

The pseudocode of our algorithm is depicted as Algorithm 1.

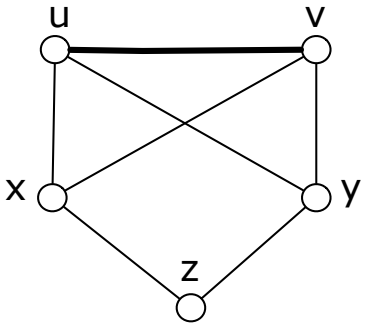

Figure 1: *Detecting C_5 passing through $\{u, v\}$*

3.3 Description of Algorithm 1

Algorithm 1 is essentially of the form "append-and-forward" (cf. Instruction 24), 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 11-13). Then, at round t, a set of $k-t$ "fake" IDs are added in $\mathcal{I}$ (cf. Instruction 14). Intuitively, these fake IDs represent the yet unknown IDs of nodes which could potentially form a C_k together with the nodes of some list received by the current node at this round. In order to form the collection $\mathcal{S}$ of lists that will be sent to neighbors at the next round (cf. Instruction 28), the collection $\mathcal{X}$ of all possible sets X of $k-t$ IDs are constructed, including fake IDs (cf. Instruction 15).

The core of the algorithm is the construction of $\mathcal{S}$ by the Instructions from 16 to 24. Before describing this crucial part of Algorithm 1 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 32-36). If a node w has two lists L_1 and L_2 in $\mathcal{R}$ such that

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

then Node w outputs "yes". Before showing that a cycle C_k formed by all nodes with IDs in $L_1 \cup L_2 \cup \{\mathrm{ID}(w)\}$ exists if and only if $|L_1 \cup L_2 \cup \{\mathrm{ID}(w)\}| = k$, we first return to the core of Algorithm 1, that is the set up of the set of lists $\mathcal{R}$.

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 1 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**
  $\mathcal{C} \leftarrow \{X \in \mathcal{X} : X \cap L = \emptyset\}$
  if $\mathcal{C} \neq \emptyset$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{L\}$
    $\mathcal{X} \leftarrow \mathcal{X} \setminus \mathcal{C}$
  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 takes the decision 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 which does not intersect L. If this is the case, 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, so that to form a cycle C_k.

For instance, Fig. 2 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, which can potentially form a k-cycle with z and all the nodes in L.

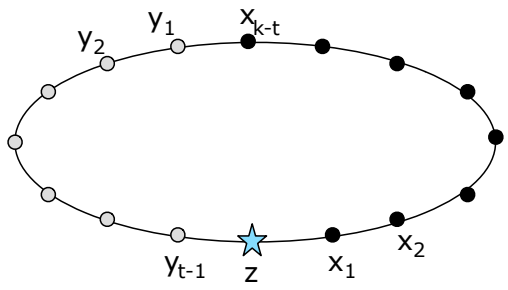

Figure 2: *Construction of $\mathcal{R}$*

Importantly, the sets $X \in \mathcal{C}$ are then removed from $\mathcal{X}$. 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 both L and L'. Therefore, as soon as L has been identified, all witnesses sets $X \in \mathcal{C}$ can safely be removed from $\mathcal{X}$.

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

```
1: function DETECTCK(u,v)
2:     if myid = u or myid = v then                          ▷ initial computation at round 1
3:         S ← {(myid)}                                      ▷ S is a set of sequences of IDs
4:     else
5:         S ← ∅
6:     end if
7:     send S to all neighbors                               ▷ send operation at round 1
8:     receive messages from all neighbors                   ▷ receive operation at round 1
9:     for t = 2 to ⌊k/2⌋ do                                 ▷ rounds 2 to ⌊k/2⌋
10:        if non-empty messages have been received at round t − 1 then
11:            R ← set of all ordered sequences of IDs received at round t − 1   ▷ R contains sequences of t − 1 IDs
12:            remove from R all sequences containing myid
13:            I ← set of IDs included in at least one sequence in R
14:            I ← I ∪ {−1, ..., −k + t}                     ▷ add k − t distinct "fake" IDs to I
15:            X ← collection of all sets X of k − t IDs in I
16:            S ← ∅                                         ▷ initializes the set of sequences to be sent
17:            for all L ∈ R do
18:                C ← {X ∈ X : X ∩ L = ∅}                   ▷ C is a sub-collection of sets X of k − t IDs
19:                if C ≠ ∅ then
20:                    S ← S ∪ {L}                           ▷ S contains ordered sequences of existing IDs
21:                    X ← X \ C
22:                end if
23:            end for
24:            append myid at the tail of each L ∈ S         ▷ S contains sequences of t IDs
25:        else
26:            S ← ∅
27:        end if
28:        send S to all neighbors                           ▷ send operation at round t
29:        receive messages from all neighbors               ▷ receive operation at round t
30:    end for
31:    if non-empty messages have been received at any round 1, ..., ⌊k/2⌋ then
32:        if k is odd then
33:            R ← {sequences received at round ⌊k/2⌋}       ▷ R contains sequences of equal length
34:        else
35:            R ← S ∪ {sequences received at round ⌊k/2⌋ − 1}   ▷ R contains sequences of lengths differing by at most 1
36:        end if
37:        if ∃L1, L2 ∈ R : |L1 ∪ L2 ∪ {myid}| = k then
38:            output reject                                 ▷ a C_k has been detected
39:        else output accept
40:        end if
41:    else output accept
42:    end if
43: end function
```

For instance, considering again the example of Fig. 2, 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 (the order in which these sequence are taken is arbitrary) is necessarily placed in $\mathcal{S}$. Indeed,

$$X = \{-1, -2, \ldots, -k + t\}$$

is 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 obvious if X contains fake IDs because X then 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 16-24 in Algorithm 1, where $\mathcal{R}$ contains just the sequence $(1,2)$. In Algorithm 1, 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 $-1,,-6$ 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.

3.4 Analysis of Algorithm 1

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 24), and these sequences L do not contain the ID of the current node (cf. Instruction 12). Therefore, every sequence sent at round $t+1$ are 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 guarantied by Instruction 12), 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 1 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 37, 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, $|L_1 \cup L_2 \cup \{\mathrm{ID}(w)\}| = k$ 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
$$L_1 \in \mathcal{S} \text{ and } L_2 \notin \mathcal{S}$$
or
$$L_1 \notin \mathcal{S} \text{ and } L_2 \in \mathcal{S}.$$
Indeed, 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 a sequence must not be contained in $\mathcal{S}$. Moreover, if $|L \cup L' \cup \{myid\}| = k$, then at least one of the two sequences must belong to $\mathcal{S}$ because the sequences received at round $k/2-1$ are of length $k/2-1$. 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 which 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 17 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 1. Yet, we show that Algorithm 1 keeps sufficiently many options for detecting the cycle. Let us fix some round $t \in \{2,\ldots,\lfloor \frac{k}{2} \rfloor\}$, and a node w. Let us consider a discarded sequence

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

at w. Let us assume that the cycle includes that sequence of nodes, that is the cycle is of the form

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

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

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

where $\mathcal{X}$ is the collection of all sets X of $k-t$ IDs in $\mathcal{I}$, and $\mathcal{I}$ is the collection of all IDs included in at least one sequence in $\mathcal{R}$, complemented with the $k-t$ "fake" IDs

$$\{-1, \ldots, -k+t\}.$$

This implies that all sets $X \in \mathcal{X}$ that intersect L have been removed from $\mathcal{X}$ when considering other sequences in the for-loop. In particular the set

$$X = \{y_1, \ldots, y_{k-t}\}$$

has been removed when considering another sequence

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

Since $L' \cap \{y_1, \ldots, y_{k-t}\} = \emptyset$, we get that there is actually another cycle,

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

where $\{z_1, y_{k-t}\} = \{u, v\}$. Therefore, Algorithm 1 satisfies that, at every round $t \in \{2, \ldots, \lfloor \frac{k}{2} \rfloor\}$, if w belongs to a cycle

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

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

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

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 1 are of constant size.

LEMMA 3.4. *For every $t = 1, \ldots, \lfloor \frac{k}{2} \rfloor$, 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 $t = 0, \ldots, \lfloor \frac{k}{2} \rfloor - 1$, 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 $t \in \{0, \ldots, \lfloor \frac{k}{2} \rfloor - 1\}$, and a node w, and let us focus on round $t+1$. For $i = 0, \ldots, t$, let us then define property P_i stating:

> 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 17, 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 17, we already added $(k-t)^i$ sequences to $\mathcal{S}$ containing the same $t-i$ elements $x_1, x_2, \ldots, x_{t-i}$. That is, $\mathcal{S}$ contains

$$\begin{aligned}
&\{x_1, x_2, \ldots, x_{t-i}, y_{1,1}, y_{1,2}, \ldots, y_{1,i}\}\\
&\{x_1, x_2, \ldots, x_{t-i}, y_{2,1}, y_{2,2}, \ldots, y_{2,i}\}\\
&\qquad\vdots\\
&\{x_1, x_2, \ldots, x_{t-i}, y_{(k-t)^i,1}, y_{(k-t)^i,2}, \ldots, y_{(k-t)^i,i}\}
\end{aligned}$$

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 $X \in \mathcal{X}$,

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

or

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

Indeed, if a sequence does not contain $x_1, x_2, \ldots, x_{t-i}$, 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 $x_1, x_2, \ldots, x_{t-i}, y_{a,b}$ 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 $x_1, x_2, \ldots, x_{t-i}$, Eq. (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 Eq. (1) cannot be satisfied. It follows that, for every $X \in \mathcal{X}$,

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

Let us now consider another sequence

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

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 $\{x_1, x_2, \ldots, x_{t-i}\}$, which implies that $L \cap X \neq \emptyset$. □

3.5 Proof of Theorem 3.1

Let us first compute the probability of detecting a cycle in a network which is ϵ-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 ([20]). *Let H be any graph. Let G be an m-edge graph that is ϵ-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 ϵ-far from being C_k-free contains at least $\epsilon m/k$ edge-disjoint copies of C_k, i.e., ϵm edges belong to edge-disjoint cycles.

LEMMA 3.6. *The probability that there is a unique edge with minimum rank after the execution of Phase 1 is at least* $1/e^2$.

PROOF. The probability that there are no collisions while choosing for each edge a random number from $[1, m^2]$ is

$$\begin{aligned}\frac{m^2-1}{m^2} \times \ldots \times \frac{m^2-m}{m^2} &\geq \left(\frac{m^2-m}{m^2}\right)^m \\ &= \left(1-\frac{1}{m}\right)^m \\ &\geq \left(e^{\frac{-2}{m}}\right)^m \\ &= \frac{1}{e^2}\end{aligned}$$

where the last inequality holds whenever $m \geq 2$. □

Let G be a graph that is ϵ-far from being C_k-free. 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

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

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 ϵ-far from being C_k-free, we repeat the whole process $\frac{e^2}{\epsilon} \ln 3$ 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 paper, 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 [20] 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, and the question of whether a distributed property testing algorithm for H-freeness exists for every arbitrarily large pattern H was left open in [20]. The techniques in this paper does 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 1 discarding some sequences of nodes is oblivious to the neighborhood of the nodes in these sequences. Hence, while Algorithm 1 makes sure to keep at least one sequence corresponding to a cycle, if such 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 [20] that their techniques do not seem to extend to *induced* subgraphs[2]. The same apparently holds for the techniques in this paper. 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, are potentially challenging but definitely rewarding issues 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.

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. Discrete Math.* 22, 2 (2008), 786–819.

[3] Noga Alon and Asaf Shapira. 2006. A Characterization of Easily Testable Induced Subgraphs. *Combinatorics, Probability & Computing* 15, 6 (2006), 791–805.

[4] Alkida Balliu, Gianlorenzo D'Angelo, Pierre Fraigniaud, and Dennis Olivetti. 2017. What Can Be Verified Locally?. In *34th Symposium on Theoretical Aspects of Computer Science (STACS)*.

[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. *Distributed Computing* 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 *30th Int. Symposium on Distributed Computing (DISC) (LNCS)*, Vol. 9888. Springer, 43–56.

[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. *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. *Random Struct. Algorithms* 45, 2 (2014), 139–184.

[12] Yuval Emek, Christoph Pfister, Jochen Seidel, and Roger Wattenhofer. 2014. Anonymous networks: randomization = 2-hop coloring. In *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 and Combinatorics* 2, 1 (1986), 113–121.

[14] Paul Erdős, András Hajnal, and J. W. Moon. 1964. A Problem in Graph Theory. *The American Mathematical Monthly* 71, 10 (1964), 1107–1110.

[15] Laurent Feuilloley and Pierre Fraigniaud. 2015. Randomized Local Network Computing. In *27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 340–349.

[16] Laurent Feuilloley and Pierre Fraigniaud. 2016. Survey of Distributed Decision. *Bulletin of the EATCS* 119 (2016), 41–65.

[17] Laurent Feuilloley, Pierre Fraigniaud, and Juho Hirvonen. 2016. A Hierarchy of Local Decision. In *43rd Int. Colloquium on Automata, Languages, and Programming (ICALP)*. 118:1–118:15.

[18] Pierre Fraigniaud, Mika Göös, Amos Korman, Merav Parter, and David Peleg. 2014. Randomized distributed decision. *Distributed Computing* 27, 6 (2014), 419–434.

[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).*

[19] 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.
[20] Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. 2016. Distributed Testing of Excluded Subgraphs. In *30th Int. Symposium on Distributed Computing (DISC) (LNCS)*, Vol. 9888. Springer, 342–356.
[21] Oded Goldreich (Ed.). 2010. *Property Testing — Current Research and Surveys*. Vol. LNCS 6390. Springer.
[22] Oded Goldreich and Dana Ron. 2002. Property Testing in Bounded Degree Graphs. *Algorithmica* 32, 2 (2002), 302–343.
[23] Oded Goldreich and Luca Trevisan. 2003. Three theorems regarding testing graph properties. *Random Struct. Algorithms* 23, 1 (2003), 23–57.
[24] Mika Göös and Jukka Suomela. 2016. Locally Checkable Proofs in Distributed Computing. *Theory of Computing* 12, 1 (2016), 1–33.
[25] Amos Korman, Shay Kutten, and David Peleg. 2010. Proof labeling schemes. *Distributed Computing* 22, 4 (2010), 215–233.
[26] 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.
[27] Moni Naor and Larry J. Stockmeyer. 1995. What Can be Computed Locally? *SIAM J. Comput.* 24, 6 (1995), 1259–1277.
[28] Gabriele Oliva, Roberto Setola, Luigi Glielmo, and Christoforos N. Hadjicostis. 2016. Distributed Cycle Detection and Removal. *IEEE Transactions on Control of Network Systems* PP, 99 (2016).
[29] David Peleg. 2000. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, Philadelphia.