Distributed Property Testing on k-Vertex-Connectivity of Digraphs

Jun-Jie Tu* and Qiang-Sheng Hua*

*

School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China

***Abstract*—The study of distributed graph property testing is to design distributed algorithms to determine whether a given graph satisfies a property P or is ϵ-far from satisfying P ($\epsilon \in (0,1)$), which means at least an ϵ-fraction of the graph's edges must be modified (added or removed) to obtain a satisfactory graph. The vertex-connectivity of a graph is a fundamental and well-studied property in graph theory. In this paper, we present a one-sided error distributed algorithm for testing the k-vertex-connectivity of digraphs for the first time, in which if the graph is k-vertex-connected, all vertices output accept; if the graph is ϵ-far from k-vertex-connectivity, at least one vertex outputs reject with probability at least $2/3$. For the so-called *sparse* graph testing model, our distributed algorithm only requires constant rounds under the classical $\mathcal{CONGEST}$ model.**

I. Introduction

The vertex-connectivity of a graph is a fundamental property in graph theory. While in networks, it also plays an important role that shows how many failed machines the system can tolerate. Let $G(V,E)$ be a digraph, a pair of vertices $u,v \in V$ are k-vertex-connected (**k-connected for short**) if u and v are still strongly-connected after deleting any $(k-1)$ vertices. We call a digraph G k-connected if deletion of any $(k-1)$ vertices does not disconnect the digraph.

Generally speaking, it is difficult to exactly compute the properties of the graph. The time complexity of these algorithms is usually unsatisfactory in terms of massive graphs. Thus property testing can be first employed if the graph is far away from satisfying the predetermined property, which can avoid the expensive computation of the graph property [9]. Let P be a property, and let $\epsilon \in (0,1)$ be a parameter. We say a graph with m edges and n vertices is ϵ-far from satisfying P in the following situations:

1) Modifying at most ϵn^2 edges in the *dense* graph testing model does not cause the graph satisfying P.
2) Modifying at most $\epsilon \cdot max\{m,n\}$ edges in the *general* graph testing model or the *sparse* graph testing model does not cause the graph satisfying P.

The graph testing models have great influences on property testing. The algorithms designed for the *dense* graph testing model may not be suitable for the *general* graph testing model or *sparse* graph testing model, and vice versa. The classification of graphs is mainly based on the degree of vertices. For a graph G with m edges and n vertices, let d_{ave} and d_{max} denote the average degree and the maximum degree, respectively. If $d_{ave} = O(n)$, G is a dense graph. If $d_{max} = O(1)$, G is a bounded degree graph. The general graph lies between the two extremes. There is no bound on the degree of vertices in general graphs [1]. When $m = O(n)$, G is a sparse graph.

In sequential property testing, a tester is a randomized algorithm which will output accpet with a high probability (generally at least $2/3$) if the graph satisfies P and output reject with a high probability if the graph is ϵ-far from satisfying P. Graphs are usually assumed to be stored in the form of adjacency lists or adjacency matrices. The algorithm runs by a series of queries which usually ask for the degree of a vertex or the i-th neighbor of a vertex. So we gengerally use the number of queries the algorithm performs to measure the time complexity of sequential algorithms. The main goal of sequential property testing is to design algorithms that run as few queries as possible.

Distributed property testing is first introduced in [2]. Instead of being stored as adjacency lists or adjacency matrices, graphs in distributed property testing usually model networks. The $\mathcal{CONGEST}$ model is one of the most classical models in distributed computing [16]. In this model, vertices represent processors and edges model the communication relationship among processors. The processors of the system communicate in synchronous rounds, and processors can only communicate with their neighbors by sending at most $O(\log n)$-bit message in one round. We aim to design a distributed algorithm in which all vertices will output accept with probability at least $2/3$ if the graph satisfies property P and at least one vertex outputs reject with probability at least $2/3$ if the graph is ϵ-far from satisfying P.

Connectivity testing has been well studied in sequential property testing. An interesting fact is that the minimum number of edges of a k-connected undirected graph with n vertices is only $\lceil \frac{nk}{2} \rceil$. It shows that by adding at most $\lceil \frac{nk}{2} \rceil$ edges, any undirected graph can be tranformed into a k-connected graph. Even though in terms of directed graphs, nk edges will suffice. So according to the previous description, *the dense graph testing model is obviously not suitable for the property testing of k-connectivity*. Therefore, our study is based on the *sparse* graph testing model. In this paper, we present a one-sided error distributed algorithm for testing the k-connectivity of digraphs, where "one-sided error" means when the graph satisfies the predetermined property, vertices of the graph will output accept definitely rather than with a certain probability. Our contributions are briefly summarized as follows:

1) We propose a distributed property testing algorithm on k-connectivity of digraphs for the first time. The algorithm only requires $O(\frac{c^k}{\epsilon^k})$ ($c > 1$ is a constant) rounds, which does not depend on the number of edges and vertices.
2) Compared with the bounded-degree graph models used in sequential algorithms [10], [17], [18], the *sparse* graph testing model our distributed algorithm adopts applies in more general situations.
3) Although our main contribution is on k-vertex-connectivity testing in digraphs, our algorithms can be also adapted to obtain distributed algorithms for k-vertex-connectivity on undirected graphs and for k-edge-connectivity testing on both undirected and directed graphs (digraphs).

The remainder of this paper is organized as follows: We give the related work in Section II and the system model and problem definition are given in Section III. In Section IV, we present our technique to test k-connectivity of $(k-1)$-connected digraphs. The discussion of distributed implementation and the detailed distributed algorithms are presented in Section V and Section VI, respectively. The correctness and the efficiency analyses are given in Section VII. The distributed k-connectivity testing on digraphs which are not $(k-1)$-connected is discussed in Section VIII. We conclude the paper in Section IX.

II. Related Work

A. Sequential Property Testing

Goldreich and Ron [10] study the property testing in terms of the graphs which have a bound on the degree of vertices. They present several randomized algorithms for testing some important graph properties such as k-connectivity and cycle-freeness. These algorithms all work in time $O(\frac{1}{\epsilon})$. Parnas et al. [15] extend and simplify the bounded-degree graph model. They represent the graph by incidence lists, the length of each list is equal to the number of neighbors of the first vertex in the list. In this model, they present an algorithm for testing whether the diameter of a given graph is bounded by D, or is ϵ-far from being at most $\beta(D)$, where D is a predetermined parameter and $\beta(D)$ is a small linear function. Alon et al. [1] discuss about the testing of triangle-freeness and more generally, H-freeness in genaral graph model, where H is a fixed subgraph. They give a lower bound of $\Omega(n^{\frac{1}{3}})$ queries for testing H-freeness (H can not be bipartite subgraph) and a sublinear upper bound for testing triangle-freeness. In addition, [3] and [12] show that it is possible to test expansion in sublinear time.

B. Distributed Property Testing

Distributed property testing has attracted an increasing attention in recent years. Testing cycle-freeness is a basic problem in graph property testing. Let C_k denote the k-vertex cycle. Censor-Hillel et al. [2] and Even et al. [4] both provide a tester for triangle-freeness, the tester of [2] requires $O(\frac{1}{\epsilon^2})$ rounds in *general* graph testing model, while [4] improves the round complexity to $O(\frac{1}{\epsilon})$ rounds. In addition, [4] also presents a distributed testing algorithm for C_4-freeness, which requires $O(\frac{1}{\epsilon})$ rounds, too. Fraigniaud and Olivetti [7] completely solve the problem of C_k-freeness testing. They present a constant-round randomized algorithm for testing C_k-freeness, for every $k \geq 3$. However, their algorithm is not suitable for testing the induced cycle. Fischer et al. [5] design another constant-round algorithm for testing C_k-freeness, which is efficient even for induced cycle. The round complexity of their algorithm is $O(\frac{1}{\epsilon})$ (actually is $O(\frac{k^k}{\epsilon})$, k^k is treated as a constant). In the above work, the size of the cycle is predetermined, while [2] presents an algorithm for testing the property of cycle-freeness in logarithmic round complexity, in which the size of the cycle is not predetermined.

Constant-size tree testing is well studied in [4], [6], [5]. The algorithms in these papers all work in $O(1)$ rounds. The difference is that [4] and [6] both have a one-sided error while [5] tests tree-freeness exactly. [2] presents the distributed version of bipartiteness testing for bounded degree graphs based on the sequential testing of [11]. The sequential tester requires $\Omega(\sqrt{n})$ queries, while the distributed one only needs $O(poly(\log(n)))$ rounds.

C. Connectivity Testing

So far, the study of connectivity testing has mainly focused on sequential algorithms. [10] first studies the k-edge-connectivity testing of undirected graphs. Since any undirected graph could be k-connected by adding at most $\lceil\frac{nk}{2}\rceil$ edges, the *dense* graph testing model are obviously not suitable for our problem. Therefore they adopt the bounded-degree graph model. They first prove that the graph which is ϵ-far from k-edge-connectivity has lots of disjoint small "components" which are disconnected from each other by a cut of at most $(k-1)$ edges. Then by utilizing the idea of Karger's min-cut algorithm [13], they design an algorithm to find out such a component from a given vertex. The time complexity of their algorithm is $O(poly(\frac{k}{\epsilon}))$.

Inheriting the idea of [10], Yoshida and Ito [17] extend the result to the testing of k-edge-connectivity of digraphs. [10] designs a special tree structure to represent the $(k-1)$-edge-connected undirected graph, and by using this structure, they get the relationship between the number of egdes that should be modified and the number of small components. However, this structure is not suitable for digraphs. So [17] gives a more concrete method and proposes an algorithm that runs in $O(d(\frac{ck}{\epsilon d})^k \log \frac{k}{\epsilon d})$ ($c > 1$ is a constant, d is the bounded degree) time.

Yoshida and Ito argue about the testing of k-vertex-connectivity of undirected graphs in [18]. As we all know, the structure of k-vertex-connected graphs is more intricate than that of k-edge-connected graphs. Vertices can be partitioned into equivalence classes in regard to edge-connectivity.

TABLE I: Related Work on Connectivity Testing

Problem	Type of Graphs	Sequential Algorithm[I] (Number of Queries)	Distributed Algorithm (Number of Rounds)
k-edge-connectivity	undirected	$O(\frac{k^3 \log(\frac{1}{\epsilon d})}{\epsilon^{3-\frac{2}{k}} d^{2-\frac{2}{k}}})$ [10]	$O(\frac{c^k}{\epsilon^k})$[this paper][III]
	directed	$O(d(\frac{ck}{\epsilon d})^k \log \frac{k}{\epsilon d})$ [17][II]	$O(\frac{c^k}{\epsilon^k})$[this paper]
k-vertex-connectivity	undirected	$O(d(\frac{ck}{\epsilon d})^k \log \frac{k}{\epsilon d})$ [18]	$O(\frac{c^k}{\epsilon^k})$[this paper]
	directed	$O(\frac{c^k}{\epsilon^k} \log \frac{1}{\epsilon})$[this paper][IV]	$O(\frac{c^k}{\epsilon^k})$[this paper]

[I] The results of [10], [17], [18] are based on the bounded degree graph model. Utilizing the idea of corresponding references and processing methods of our paper, these results can be extended to the sparse graph model.
[II] d is the bounded degree, $c > 1$ is a constant, and c will differ in different table cells.
[III] Utilizing the processing methods of our paper, we can get the round complexity of distributed versions of [10], [17], [18] in the bounded degree graph model and the sparse graph model.
[IV] In the sparse graph testing model, utilizing the algorithm of [18] and Theorem 1 of this paper, we can get the time complexity of k-vertex-connectivity testing on digraphs.

However, since vertex-connectivity does not satisfy transitivity, there is no such equivalence calsses in regard to vertex-connectivity. Therefore, they employ some facts in the field of vertex-connectivity augmentation to solve the problem, and present an algorithm which tests k-vertex-connectivity of undirected graphs in $O(d(\frac{ck}{\epsilon d})^k \log \frac{k}{\epsilon d})$ time.

Our algorithm in this paper also extends the idea of Goldreich and Ron [10]. Utilizing an important lemma on vertex-connectivity augmentation [8], we prove that there are $\Omega(m)$ small "components" which contain constant number of vertices and present a distributed testing algorithm with constant round complexity in the sparse graph testing model. We summarize the related work on connectivity testing in Table I.

III. Preliminary

A. System Model

The $\mathcal{CONGEST}$ model is one of the most classical models in distributed computing [16]. In this model, we use a graph to model a network where the vertices represent the processors and the edges represent the communication relationship among processors. The model usually runs in synchronous rounds and enforces the $O(\log n)$-bit limitation on the maximum message size. In every synchronous round, each vertex performs the following steps:

1) Receive messages from its neighbors.
2) Perform some local computation.
3) Send messages to its neighbors.

Generally, local computation is assumed to take negligible time compared to message transmission. In this paper, we test the k-connectivity of digraphs, and **the communication network is the underlying undirected graph**, which means the model is able to communicate independent of the directions of edges [14]. To measure the performance of a distributed algorithm, we introduce *round complexity*, which is defined as the number of rounds that the algorithm performs.

B. Problem Definition

Let $G(V, E)$ denote a simple digraph without loops and multiple edges, where V and E represent the set of vertices and the set of directed edges (or edges, for short), respectively. The out-edges of v are edges $(v, w) \in E, \exists w \in V$, and we call w an out-neighbor of v. The out-degree of v is the number of the out-edges of v, which is denoted by $d^+(v)$. Similarly, the in-edges of v are edges $(u, v) \in E, \exists u \in V$, and u is an in-neighbor of v. We use $d^-(v)$ to denote the in-degree of v, which is the number of the in-edges of v. For any vertex of a k-connected digraph, there exist $d^+(v) \geq k$ and $d^-(v) \geq k$. So we suppose the given digraphs to be tested satisfy this property, which means $m = \Omega(kn)$. For any set $S \subseteq V$, $N^+(S)$ and $N^-(S)$ represent the set of out-neighbors and in-neighbors of set S, respectively. That is $N^+(S) = \{w \in V - S | \exists v \in S, (v, w) \in E\}$, and $N^-(S) = \{u \in V - S | \exists v \in S, (u, v) \in E\}$.

Now, we give the formal description of our research problem in this paper. We aim to design a randomized one-sided error distributed algorithm to test whether a given digraph is k-connected or ϵ-far from k-connectivity, where being ϵ-far from k-connectivity means that we have to add at least ϵm edges to transform the digraph into a k-connected one. Table II summarizes the frequently used notations in this paper.

TABLE II: Notations and Their Definitions

Notation	Definition
m	the number of edges in a digraph
n	the number of vertices in a digraph
(u, v)	the directed edge from u to v
$d^+(v)$	the out-degree of v
$d^-(v)$	the in-degree of v
$N^+(S)$	the set of out-neighbors of set S
$N^-(S)$	the set of in-neighbors of set S
$t_{out}(G)$	the maximum number of disjoint minimal out-tight sets in G
$t_{in}(G)$	the maximum number of disjoint minimal in-tight sets in G
$m(G)$	the minimum number of edges added to make G k-connected
G_S	the induced subgraph of vertex set S

IV. Insights and The Sequential Algorithm

A. Main Insights

As mentioned in Section II, we empoly some facts in the field of vertex-connectivity augmentation to design our algorithm. Given a digraph $G(V, E)$, the problem of digraph vertex-connectivity augmentation is to find the smallest edge set B so that $G(V, E \cup B)$ is k-connected.

Based on the study of vertex-connectivity augmentation, we first argue about an easier situation: how to test whether a given $(k-1)$-connected digraph is k-connected or ϵ-far from being k-connected. Given a $(k-1)$-connected digraph, for any set $S \subseteq V$, S is called out-tight if $N^+(S) = k-1$. Similarly, S is an in-tight set if $N^-(S) = k-1$. If a tight set S does not contain any other tight set, we call S a minimal tight set. Let $t_{in}(G)$ and $t_{out}(G)$ denote the maximum number of disjoint minimal in-tight sets and minimal out-tight sets in G, respectively. Let $m(G)$ denote the minimum number of edges to be added to make G k-connected. The following Lemma 1 shows the relationship between $m(G)$ and $t_{in}(G)$ or $t_{out}(G)$.

Lemma 1 ([8]). *Let G be a $(k-1)$-connected digraph, then $m(G) \leq max\{t_{in}(G), t_{out}(G)\} + (k-1)$.*

Lemma 1 gives an upper bound on $m(G)$. On the contrary, if $m(G)$ is given, we can get a lower bound on $t_{out}(G)$ or $t_{in}(G)$. So we can get the following corollary.

Corollary 1. *Let G be a $(k-1)$-connected digraph which is ϵ-far from being k-connected, then $max\{t_{in}(G), t_{out}(G)\} \geq \frac{\epsilon m}{2}$.*

Proof. Since G is ϵ-far from being k-connected, $\epsilon m \leq m(G) \leq max\{t_{in}(G), t_{out}(G)\} + (k-1)$. And it is obvious that $m \geq \frac{2k-2}{\epsilon}$ for k and ϵ are all constants, so $max\{t_{in}(G), t_{out}(G)\} \geq \epsilon m - k + 1 \geq \frac{\epsilon m}{2}$. □

For the sake of convenience, we suppose $t_{out}(G)$ is larger than $t_{in}(G)$ in the following. Then, we have the following Theorem 1.

Theorem 1. *Let G be a $(k-1)$-connected digraph which is ϵ-far from being k-connected, then there exist at least $\frac{\epsilon m}{4}$ disjoint minimal out-tight sets, and each set contains at most $\frac{4n}{\epsilon m}$ vertices.*

Proof. By contradiction, we suppose that there are less than $\frac{\epsilon m}{4}$ disjoint out-tight sets with at most $\frac{4n}{\epsilon m}$ vertices in each one. So by Corollary 1 we know that there are more than $\frac{\epsilon m}{4}$ disjoint minimal out-tight sets, each containing at least $\frac{4n}{\epsilon m}$ vertices. Then the number of vertices in G is larger than n, which is a contradiction. □

This theorem shows that there are quite a lot of disjoint minimal tight sets that contain only constant number of vertices in $(k-1)$-connected digraphs which are ϵ-far from being k-connected. In regard to a k-connected digraph, for any vertex set X, we have $N^+(X) \geq k$ and $N^-(X) \geq k$. So we can distinguish k-connected digraphs from $(k-1)$-connected digraphs which are ϵ-far from being k-connected by finding out such a minimal out-tight set. Now, considering if a vertex r in such a minimal out-tight set S is given, how can we find out S?

B. The Sequential Search Tight Set Algorithm

[18] solves this problem with the BFS (Breadth-First-Search) algorithm in undirected graphs. Suppose the specified vertex r and the upper bound t (t is a constant) of the size of the tight set S are given. Let $N(S)$ denote the neighbors of S. Firstly, we perform a BFS from source r. Let T be the set of vertices reached in the BFS, the BFS stops when it finds $(t+1)$ vertices. Since $|S| \leq t$, when $|T| > t$, there must be at least one vertex that belongs to $N(S)$. We remove one vertex from T. Suppose that we just remove one vertex in $N(S)$, then $|N(S)|$ decreases by one. To find other vertices in $N(S)$, we need to start a BFS again from r until $(t+1)$ vertices are reached, then we delete one vertex in T again. This process repeats $(k-1)$ times. If all the $(k-1)$ vertices in $N(S)$ are removed by coincidence, BFS from r will not reach more than t vertices any more, which means we find out such a small tight set. We will present the Exhaustive-Search Algorithm designed by Yoshida and Ito in Algorithm 1.

Since in the centralized system, the system knows the global knowledge, so the BFS can stop as soon as $(t+1)$ vertices have been reached (line 1). Since $|T|$ is fixed, we can try to delete each vertex $v \in T$ and use the iterative approach to run the Exhaustive-Search Algorithm in which the tested graph is $G - v$, and the size of $N(S)$ is $(k-2)$ (lines 8-12). If $v \notin N(S)$, after another $(k-2)$ times BFS, we can still reach more than t vertices from r and the algorithm outputs FALSE (lines 5-7). If $v \in N(S)$, $|N(S)|$ decreases by one. After $(k-1)$ times of such deletion, BFS will not reach more than t vertices and return TRUE to show that the algorithm has found out the tight set (lines 2-4).

Algorithm 1 Exhaustive-Search($r, k-1, t, G$) [18]

```
Input: graph $G$, the source $r$, the bound $t$
Output: TRUE or FALSE
1: Perform BFS from $r$ until $(t+1)$ vertices have been reached, and let $X$ be the set of reached vertices
2: if $|X| < u + 1$ then
3:     return TRUE
4: end if
5: if $k - 1 = 0$ then
6:     return FALSE
7: end if
8: for each $v \in X$ do
9:     if Exhaustive-Search($r, k-2, t, G-v$) then
10:        return TRUE
11:    end if
12: end for
13: return FALSE
```

V. Distributed Implementation Discussions

A. Distributed Implementation Issues

We know that in distributed $\mathcal{CONGEST}$ model, vertices can only directly communicate with neighbors in a round using $O(\log n)$ size messages, so it's hard to gain the global knowledge of the graph. Therefore, there exist several issues to implement the distributed testing on k-connectivity of digraphs.

1) The structure of digraphs is more complicated than that of undirected graphs. We do not know the structure of the minimal tight sets in digraphs: are these vertices in the same minimal tight set reachable to each other without crossing vertices outside the tight set?
2) As mentioned in the sequential algorithm, when $|T| > t$, there must be at least one vertex that belongs to $N^+(S)$. Then the BFS will stop and we remove a vertex from the spanning tree. In the sequential BFS algorithm, we can know the size of the spanning tree in real time, the algorithm can stop as soon as the BFS reach $(t+1)$ vertices. However, it is an issue for the distributed BFS algorithm to know the specific number of vertices of the spanning tree in real time. It must take steps to give real-time feedback on the size of the spanning tree and control the upper bound of the size of the spanning tree so that we can analyze the correctness and the round complexity of our algorithm.
3) Compared with the bounded degree graph testing model widely used in the sequential setting, our distributed algorithm employs the sparse graph testing model. There might be constant number of vertices whose degrees are more than $\Omega(1)$ in the sparse graph model. When our distributed BFS algorithm reaches such vertices, the size of the spanning tree will be out of control in the next stage where the BFS extends outword. So how to deal with such vertices is an issue to be considered in our testing model.

B. Solutions for The Issues

We first study the structure of the minimal out-tight set, then we talk about how to control the size of the spanning tree in distributed BFS. In the end, we argue how to deal with the vertices with non-constant ($\omega(1)$) number of out-neighbors.

Lemma 2. *Let $G(V,E)$ be a $(k-1)$-connected digraph and $S \subseteq V$ be a minimal out-tight set. For any set $S_1 \subseteq S$, there exists at least one edge between S_1 and $S - S_1$, which means there exist $u \in S_1, v \in S - S_1$, such that $(u,v) \in E$ or $(v,u) \in E$.*

Proof. By contradiction, we suppose that there exists a vertex set S_1, such that there is no edge between S_1 and $S - S_1$, so $N^+(S_1) \subseteq N^+(S)$. Since S is a minimal out-tight set, for any set $S_1 \subseteq S$, we have $|N^+(S_1)| > k-1$. Therefore, $|N^+(S)| \geq |N^+(S_1)| > k-1$, which is a contradiction to the condition that S is an out-tight set. □

Lemma 3. *Let $G(V,E)$ be a $(k-1)$-connected digraph and $S \subseteq V$ be a minimal out-tight set. Then the induced subgraph of S is strongly connected.*

Proof. Let G_S denote the induced subgraph of S. By contradiction, supposing that G_S is not strongly connected, there exists a partition of S denoted by $S_1, S_2, \ldots, S_l$, such that $G_{S_1}, G_{S_2}, \ldots, G_{S_l}$ are strongly connected components of G_S. By Lemma 2, we know for each $i \in \{1,2,\ldots,l\}$, there exists at least one $j \neq i$ and $j \in \{1,2,\ldots,l\}$, such that there exists at least one edge between S_i and S_j. And these edges are all from S_i to S_j or from S_j to S_i, otherwise the induced subgraph of $S_i \cup S_j$ is strongly connected.

Now we contract G_S as follows: for each strongly connected component G_{S_i}, we contract it into a new vertex s_i. The effect is that the edges whose two endpoints are both in S_i are deleted, and all the vertices in S_i are removed and replaced by a new vertex s_i. The edges between S_i and S_j are replaced by a weighted edge (s_i, s_j) or (s_j, s_i), whose weights equal the number of edges between S_i and S_j and the direction is the same as that of these edges. Then G_S is contracted into a directed acyclic graph (or DAG, for short). Since if there is a cycle in the digraph, the strongly connected components represented by the vertices in the cycle together constitute a strongly connected component, which is a contradiction. Fig. 1 shows how to contract a digraph.

Then considering the DAG G'_S, which is the result of contracting G_S. According to the property of DAG, there must exist at least one vertex which only has in-neighbors. Without loss of generality, suppose s_l is such a vertex. This means S_l has no out-neighbors in G_S, so $N^+(S_l) \subseteq N^+(S)$. Since S is a minimal out-tight set, $|N^+(S_l)| > k-1$. Therefore, $|N^+(S)| \geq |N^+(S_l)| > k-1$, which is a contradiction to the condition that S is an out-tight set. So G_S is strongly connected. □

By Lemma 3, we know that from any vertex $u \in S$, the BFS can reach any other vertex in S without going through any vertex $v \notin S$. Now given the specified vertex r and the upper bound t (t is a constant) of the size of S, we first perform a distributed BFS from source r. Here we introduce *stage* to describe the process of our distributed bounded BFS extending outword. One stage consists of two parts, one part is the BFS

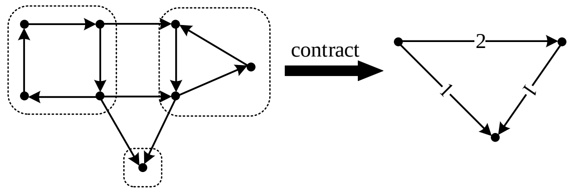


Fig. 1: This is an example of contraction operation on a digraph. The digraph in this example contains three strongly connected components, which are cycled by dotted lines. It is contracted into a triangle, and the numbers on the edges represent the weights of the edges.

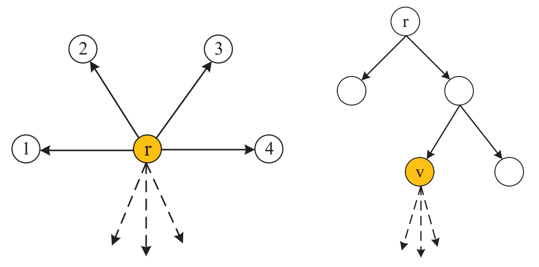
tree extending outward by one layer; the other part is the process of counting the size of the spanning tree at this stage. We use echo algorithm to compute the size of spanning tree in each stage. When the BFS finds more than t vertices, the source will remove several vertices from the leaves according to the rule introduced in the subsequent Section VI and stop the BFS. Then we delete one vertex from the final spanning tree. This process repeats $(k-1)$ times. If all the $(k-1)$ vertices in $N^+(S)$ are removed by coincidence, BFS from r will not reach more than t vertices any more, which means we find out such a small out-tight set. It seems that the probability of just deleting all vertices in $N^+(S)$ is very low. Nevertheless, considering that S contains only constant number of vertices, we can repeat the above process many times, then we can prove that the situation where we happen to delete the $(k-1)$ vertices in $N^+(S)$ will happen with a high probability.

When we perform distributed BFS algorithm in the sparse digraph, we may meet several vertices which have non-constant number of out-neighbors. We summarize this situation into two cases:

1) If the source r has non-constant number of out-neighbors, the distributed BFS should terminate immediately.
2) If a leaf v has non-constant number of out-neighbors, v will not participate in the subsequent stages until the distributed BFS terminates.

Suppose that the BFS does not terminate immediately even if the source r has non-constant number of out-neighbors, then it will terminate after the first stage for $|T| > t$. Now we will show that the source r cannot be a member of a minimal out-tight set containing at most t vertices. Take Fig. 2a as an example, if all of r's out-neighbors are the members of an out-tight set, the size of the out-tight set is obviously out of the bound t. If r selects several out-neighbors such as the vertices with labels in Fig. 2a as the members of the out-tight set S, $N^+(S) > k-1$ for r still have non-constant number of out-neighbors, which violates the definition of tight set. So in Case 1, it is impossible to find out a minimal out-tight set with a size at most t from source r.

Similar to Case 1, take Fig. 2b as an example of Case 2. Suppose that the leaf v which has non-constant number of out-neighbors continues to participate in the next stage of BFS. Then the BFS will terminate after the next stage for $|T| > t$. If all of v's out-neighbors are the members of a minimal out-tight set, the size of the out-tight set is obviously out of the bound t. If v selects several out-neighbors as the members of the out-tight set S, $N^+(S) > k-1$ for $N^+(v) = \omega(1)$, which violates the definition of tight set. So if the source r is contained in a minimal out-tight set with a size at most t, v must be out of the set. Therefore, when BFS reaches such a vertex, this vertex should stop participating in the algorithm.



(a) Case 1 (b) Case 2

Fig. 2: These are examples of Case 1 and Case 2. In Case 1, r is the source vertex, vertices with labels are its several out-neighbors. Three edges indicated by dashed lines denote $\omega(1)$ out-neighbors. In Case 2, r is the source vertex, v is the leaf which has $\omega(1)$ out-neighbors. Three edges indicated by dashed lines denote all v's out-neighbors.

VI. Details of Distributed Algorithms

In this section, detailed description of algorithms for k-connectivity testing of $(k-1)$-connected digraph is given. Different from the corresponding sequential algorithm, the distributed BFS algorithm does not know the number of vertices it has reached, so we first give Algorithm 2 to count the number of vertices of the tree. Then we give the distributed bounded BFS algorithm with the help of Algorithm 2. At last, we present the testing algorithm.

A. Count Vertices of Tree

Algorithm 2 is an application of the echo algorithm. Given the tree T and the root r, every vertex in the tree has a unique ID and knows who is its parent or who are its children. Let $count_v$ denote the size of the subtree rooted at v. We initialize $count_v \leftarrow 1$ for every vertex v (line 2). The algorithm begins with the leaves (lines 3-5), and if an inner vertex has received messages from all children, it computes the size of the subtree rooted at itself and sends the result to its parent (lines 6-11). When the algorithm terminates, the root will compute the size of the tree T (lines 12-15).

From Section IV, we know the BFS will stop when it reachs more than t vertices. Then we will randomly delete a vertex in the spanning tree. In our algorithm, we assign an ID for each vertex from $[n^3]$ uniformly and randomly. Suppose that all of these IDs are unique (we will evaluate the effect of this assumption in Observation 1), the effect of deleting the vertex with minimum ID is equal to that of deleting a vertex uniformly and randomly. So we need to know which vertex has the minimum ID in the tree. Let id_v denote the ID of vertex v, and let $path_v$ denote the queue from v to the vertex with minimum ID of the subtree rooted at v. We initialize $path_v \leftarrow \{id_v\}$ for each vertex v (line 2). Firstly, all leaves send their queues to parents (lines 3-5). For an inner vertex w, if it has received all queues from its children, it will save the queue with the minimum ID and add id_w into the queue. Then w sends the queue to its parent (lines 8-11). When the algorithm terminates, this process produces a path from the root to the vertex with minimum ID (lines 12-15). If the root of the tree has the minimum ID, we should select the second minimum ID (line 13).

Algorithm 2 Count Vertices of Tree

```
Input: Tree $T$, the root $r$
Output: $count_r$, $path_r$
1: for vertex $v$ do
2:     $v$ initializes $count_v \leftarrow 1$ and $path_v \leftarrow \{id_v\}$
3:     if $child_v = \varnothing$ then
4:         $v$ sends $count_v$ and $path_v$ to its parent
5:     end if
6:     if $v$ receives $count_w$ and $path_w$ from its child $w$ then
7:         $count_v \leftarrow count_v + count_w$
8:         if $v \neq r$ receives all its children's messages then
9:             $path_v \leftarrow$ insert $id_v$ into the head of the queue
               which has the minimum tail
10:            $v$ sends $count_v$ and $path_v$ to its parent
11:        end if
12:        if $v = r$ receives all its children's messages then
13:            $path_r \leftarrow$ insert $id_r$ into the head of the queue
               which has the minimum tail except $id_r$
14:            output $count_r$ and $path_r$
15:        end if
16:    end if
17: end for
```

B. Distributed Bounded BFS

The distributed bounded BFS means performing distributed BFS process by the number of vertices constraint t ($t > 0$ is a constant). Let L_v denote the distance from the source r to vertex v. Initially, we set $L_r \leftarrow 0, L_v \leftarrow \infty$ (line 2 and line 14). Let $parent_v$ denote the parent of v and let $child_v$ denote the child set of v. Each stage, every leaf w sends a message L_w to all out-neighbors. If a vertex v receives a message L_w from an in-neighbor w, vertex v will compare $L_w + 1$ with L_v, if $L_w + 1 < L_v$, vertex v will set $parent_v \leftarrow w$ and $L_v \leftarrow L_w + 1$. Then, v sends a message $v \in child_w$ to its parent w (lines 15-24). Vertex w will incorporate v into $child_w$ after receiving this message. Before the start of the next stage, all vertices in the current spanning tree perform Algorithm 2 to check whether the size of the current spanning tree is out of bound (lines 25-31). If not, the algorithm goes into the next stage, otherwise the algorithm terminates.

Since the number of new vertices reached in each stage in distributed BFS algorithm is not fixed, Algorithm 3 provides a lower bound t but not an upper bound. To control the size of the spanning tree so that we can analyze our algorithms, when $|T| > t$, the source r should compute two difference values in Algorithm 3. Let $count_r^i$ denote the size of the spanning tree at stage i. Suppose that Algorithm 3 runs h stages totally, then the one value is $\Delta count_1 \leftarrow (t+1) - count_r^{h-1}$. The other value is $\Delta count_2 \leftarrow count_r^h - count_r^{h-1}$. Then r sends $ratio \leftarrow \Delta count_1 / \Delta count_2$ to all vertices whose children are leaves (lines 6-8). For such a vertex v whose children are leaves, it uniformly and randomly selects $\lceil |child_v| \cdot ratio \rceil$ vertices from child set as leaves of the final spanning tree and remove other vertices from child set (lines 32-34). The upper bound of the size of the final spanning tree will be analyzed in Section VII. As for how these vertices know that their children are leaves, when a leaf receives the $ratio$, it can send a message to its parent to show that it is a leaf. This process is easy, so we omit the description in Algorithm 3.

Since there is no bound on the degree of vertex, there may exist some vertices whose out-degrees are $\omega(1)$. Algorithm 3 needs to pay attention to two special cases. Case 1 is that if the source r has $\omega(1)$ out-neighbors, Algorithm 3 terminates immediately (line 1). Case 2 is that if a leaf v has $\omega(1)$ out-neighbors, v will not participate in the subsequent stages until Algorithm 3 terminates (line 15).

Algorithm 3 Distributed Bounded BFS

```
Input: Digraph $G$, the source $r$, the bound $t$
Output: the spanning tree $T$
1: for source $r$ satisfying $d^+(r) = O(1)$ do
2:     Initialize $L_r \leftarrow 0$, $child_r \leftarrow \varnothing$, $count_r \leftarrow 1$
3:     if $count_r$ does not change in two stages then
4:         Broadcast STOP to the spanning tree
5:     end if
6:     if $count_r > t$ then
7:         $ratio \leftarrow \Delta count_1 / \Delta count_2$
8:         Broadcast $ratio$ to the spanning tree
9:     else
10:        Broadcast CONTINUE to the spanning tree
11:    end if
12: end for
13: for vertex $v$ do
14:    Initialize $L_v \leftarrow \infty$, $child_v \leftarrow \varnothing$, $count_v \leftarrow 1$
15:    if leaf $v$ satisfying $d^+(v) = O(1)$ receives CONTINUE
       then
16:        Send $L_v$ to all out-neighbors
17:    end if
18:    if vertex $v$ receives a message $L_w$ from $w$ then
19:        if $L_w + 1 < L_v$ then
20:            $parent_v \leftarrow w$
21:            $L_v \leftarrow L_w + 1$
22:            Send $v \in child_w$ to $w$
23:        end if
24:    end if
25:    if $v$ receives a message $w \in child_v$ from $w$ then
26:        $child_v \leftarrow child_v \cup \{w\}$
27:        Send Acknowledge to $w$
28:    end if
29:    if $v$ receives Acknowledge from parent then
30:        Perform Algorithm 2
31:    end if
32:    if $v$ receives $ratio$ and its children are leaves then
33:        Uniformly and randomly delete $|child_v| - \lceil |child_v| \cdot$
           $ratio \rceil$ children from $child_v$
34:    end if
35: end for
```

C. Search Tight Set

Suppose that we are given a starting vertex r which lies in a minimal out-tight set S satisfying $|S| \leq t$, what we want to do is to find out all the $(k-1)$ vertices in $N^+(S)$. If we can successfully delete these vertices, BFS from source r will never reach more than t vertices. Algorithm 4 solves this problem. There are double loops in Algorithm 4. In the inner loop (lines 2-6), we first assign an ID $\in [n^3]$ to each vertex, and then perform Algorithm 3 from source r. Each time Algorithm 3 terminates, source r will select the vertex with the smallest ID to sleep through the path provided by Algorithm 2 (sleeping means no longer participating in the BFS until the BFS terminates). The vertex selected will not participate in the algorithm until the source awakens it. After Algorithm 3 runs k times, source r will check whether it satisfies that $count_r \leq t$, if so, it means that the BFS from source r can no longer reach more than t vertices. In other words we have found out the minimal tight set S. Then source r outputs TRUE and the algorithm terminates. Otherwise, source r awakens all sleeping vertices and runs the inner loop again (lines 7-10). After the outer loop runs $2^{k+2}t^k$ times, if source r still does not output TRUE, source r outputs FALSE to show that Algorithm 4 does not find out the tight set.

D. Connectivity Testing

Given a $(k-1)$-connected digraph G, we don't know which vertex is in a minimal out-tight set containing at most t vertices. So we need to select several vertices uniformly and randomly as sources to perform Algorithm 4. By Theorem 1, we set the bound t as $\lceil \frac{4n}{\epsilon m} \rceil$. If Algorithm 4 outputs TRUE, which means there exists such a minimal out-tight set, the source will output REJECT to show that G is ϵ-far from being k-connected. Otherwise all vertices in G output ACCEPT.

Algorithm 4 Search Tight Set

```
Input: Digraph $G$, the source $r$, the bound $t$
Output: TRUE or FLASE
 1: for $i = 1$ to $2^{k+2}t^k$ do
 2:     for $j = 1$ to $k$ do
 3:         Each vertex uniformly and randomly select an ID from $[n^3]$
 4:         Perform Algorithm 3 from source $r$
 5:         Choose the vertex $v$ with minimum ID in the spanning tree to Sleep
 6:     end for
 7:     if $count_r < t + 1$ then
 8:         $r$ outputs TRUE
 9:     end if
10:     Sourse $r$ broadcasts Wake Up to all vertices
11: end for
12: $r$ outputs FLASE
```

Algorithm 5 Connectivity Testing

```
Input: Digraph $G$
Output: REJECT or ACCEPT
 1: for $i = 1$ to $\lceil \frac{8n}{\epsilon m} \rceil$ do
 2:     Select a vertex as the source uniformly and randomly
 3:     Perform Algorithm 4 with bound $\lceil \frac{4n}{\epsilon m} \rceil$
 4:     if Algorithm 4 outputs TRUE then
 5:         the source outputs REJECT
 6:     else
 7:         the source outputs ACCEPT
 8:     end if
 9: end for
10: all other vertices output ACCEPT
```

VII. Algorithms Analyses

A. Correctness Analysis

Lemma 4. *When Algorithm 3 terminates, the size of the spanning tree is at most* $2t$*.*

Proof. To control the size of the spanning tree, for each vertex v whose children are leaves, it uniformly and randomly selects $\lceil |child_v| \cdot ratio \rceil$ vertices from its children set as leaves of the final spanning tree and removes other vertices from child set. Since the value is rounded up, the size of final spanning tree may be larger than $t+1$. while the number of vertices whose children are leaves is at most $t-1$, so the size of the final spanning tree

$$\begin{aligned} |T| &\leq t + 1 + \sum_{i=1}^{t-1} [\lceil |child_{v_i}| \cdot ratio \rceil - (|child_{v_i}| \cdot ratio)] \\ &\leq 2t. \end{aligned}$$

□

In Algorithm 4, after each time we run Algorithm 3, we have to choose the vertex with the minimum ID in the spanning tree to go to sleep. But it's difficult to ensure every vertex has an unique ID. So we evaluates the effect in the following observation.

Observation 1. *Every vertex in digraph G uniformly and randomly selects an ID from* $[n^3]$*, then with probability at least* $1 - \frac{2}{n}$*, every vertex has an unique ID.*

Proof. We first select n different IDs from $[n^3]$, and then assign these IDs to n vertices. These are all cases where all vertices' IDs are different from each other. We use ξ to denote the event where all vertices have different IDs. So the probability of ξ is

$$\begin{aligned} P(\xi) &= \frac{n^3(n^3-1)\cdots(n^3-n+1)}{n^{3n}} \\ &\geq (\frac{n^3-n+1}{n^3})^n \geq (1 - \frac{1}{n^2})^n \\ &= (1 + \frac{1}{n})^n (1 - \frac{1}{n})^n \\ &\geq e(1 - \frac{1}{n})\frac{1}{e}(1 - \frac{1}{n}) \\ &\geq 1 - \frac{2}{n}, \end{aligned}$$

where the third inequality holds whenever $n > 1$. □

Lemma 5. *Algorithm 4 will output TRUE with probability at least $9/10$ if the source r is contained in a minimal out-tight set with a size at most t, otherwise output FALSE.*

Proof. We use η to denote the event where we just remove all vertices in $N^+(S)$. By Lemma 4, we know the size of the spanning tree is at most $2t$, so $P(\eta) \geq \frac{1}{(2t)^k}$. We use ζ to denote the event where η happens in Algorithm 4. So the probability of ζ is

$$\begin{aligned} P(\zeta) &= 1 - (1 - P(\eta))^{2^{k+2}t^k} \\ &\geq 1 - [(1 + \frac{-1}{(2t)^k})^{(2t)^k}]^4 \\ &\geq 1 - e^{-4}. \end{aligned}$$

In Algorithm 4, we implement ζ by removing the vertex with the minimum ID each time. So we should make sure all IDs in G are unique. So the probability that ξ and ζ occur simultaneously is

$$\begin{aligned} P(\xi \cap \zeta) &= P(\xi)P(\zeta) \\ &\geq (1 - \frac{2}{n})(1 - e^{-4}) \\ &\geq \frac{9}{10}, \end{aligned}$$

where the second inequality holds for large n.

If the source r is not contained in a minimal out-tight set, which means $N^+(S) > k-1$ for any set S containing r, BFS from r can always reach more than t vertices after deleting any $k-1$ vertices. So the source r will output FALSE. □

Theorem 2. *Let G be a $(k-1)$-connected digraph. In Algorithm 5, all vertices will always output ACCEPT if digraph G is k-connected, and at least one vertex outputs REJECT with probability at least $2/3$ if G is ϵ-far from being k-connected.*

Proof. If G is k-connected, for any set S, we have $N^+(S) > k-1$, which means there is no out-tight set in G, Algorithm 4 will never output TRUE. So all vertices in Algorithm 5 will always output ACCEPT. If G is ϵ-far from being k-connected, we know that there exist at least $\frac{\varepsilon m}{4}$ disjoint minimal out-tight sets in G, and each set contains at most $\frac{4n}{\varepsilon m}$ vertices by Theorem 1. The number of vertices in these sets are at least $\frac{\varepsilon m}{4}$. We use μ to denote the event where we select at least one vertex in these sets in Algorithm 5, then the probability of event μ is

$$\begin{aligned} P(\mu) &\geq 1 - (1 - \frac{\epsilon m}{4n})^{\frac{8n}{\epsilon m}} \\ &\geq 1 - e^{-2}. \end{aligned}$$

When combined with Lemma 3, the probability that μ, ξ and ζ occur simultaneously is

$$\begin{aligned} P(\mu \cap \xi \cap \zeta) &= P(\mu)P(\xi \cap \zeta) \\ &\geq \frac{9}{10} \times (1 - e^{-2}) \\ &\geq \frac{2}{3}. \end{aligned}$$

□

B. Round Complexity

Lemma 6. *Algorithm 3 terminates in $O(\frac{t^2}{k^2})$ rounds.*

Proof. The round complexity of Algorithm 3 consists of three parts: the rounds BFS takes, the rounds broadcast takes and the rounds Algorithm 2 takes. We first compute how many rounds the i-th stage takes. Since the depth of the BFS tree is i in the i-th stage, the broadcast operation of the source r and Algorithm 2 both take i rounds; the extending of the BFS tree takes three rounds, so the i-th stage takes $2i+3$ rounds totally. Since we suppose in the digraph G the out-degree of all vertices is at least k, which means the BFS tree will increase at least k vertices each stage. Therefore the number of stages will not exceed $\frac{t}{k}$. In the last stage, source r has an extra broadcast to control the size of the final spanning tree. So the rounds Algorithm 3 takes are

$$\begin{aligned} \sum_{i=1}^{\frac{t}{k}} (2i+3) + \frac{t}{k} &= \frac{4t}{k} + \sum_{i=1}^{\frac{t}{k}} 2i \\ &= \frac{4t}{k} + (2 + \frac{2t}{k}) \times \frac{t}{2k} \\ &= O(\frac{t^2}{k^2}). \end{aligned}$$

□

Lemma 7. *Algorithm 4 terminates in $O(c_1^k t^k)$ ($c_1 > 1$ is a constant) rounds.*

Proof. Actually, Algorithm 3 performs $2^{k+2}t^k$ times in Algorithm 4, so the round complexity is

$$2^{k+2}t^k \cdot O(\frac{t^2}{k^2}) = O(c_1^k t^k).$$

□

Theorem 3. *Let G be a $(k-1)$-connected digraph, Algorithm 5 tests k-connectivity of G in $O((\frac{c}{\epsilon k})^k)$ ($c > 1$ is a constant) rounds.*

Proof. In Algorithm 5, we perform Algorithm 4 $\frac{8n}{\epsilon m}$ times with bound $\frac{4n}{\epsilon m}$, so the round complexity of Algorithm 5 is

$$\begin{aligned} O(c_1^k t^k) \cdot \frac{8n}{\epsilon m} &= O(c_1^k (\frac{4n}{\epsilon m})^k) \cdot \frac{8n}{\epsilon m} \\ &= O((\frac{c}{\epsilon k})^k), \end{aligned}$$

where the second equality holds for $m = \Omega(kn)$. □

VIII. Testing k-Connectivity of Digraph That Are Not $(k-1)$ Connected

In this section, we consider how to test k-connectivity of digraphs which are not $(k-1)$-connected. Let G be a digraph that is ϵ-far from being k-connected. Considering such a process, we first add some edges to make G connected (If G is already connected, add zero edges), then add some edges to make G 2-connected, 3-connected, an so on until G is k-connected. We set $G_0 = G$, $m_0 = 0$. Let G_i be the i-connected digraph by adding fewest edges to G_{i-1}, and m_i

be the minimum number of edges which should be added to G_{i-1} to make it i-connected (If G_{i-1} is already i-connected, $m_i = 0$). Obviously, there exists some steps such that the number of added edges is more than $\frac{\epsilon m}{k}$. Without loss of generality, let i be the first step where $m_i \geq \frac{\epsilon m}{k}$. We can get the following lemma.

Lemma 8. *There exists some i such that G_{i-1} is $\frac{\epsilon}{k+(i-1)\epsilon}$-far from being i-connected.*

Proof. Let i be the first step where $m_i \geq \frac{\epsilon m}{k}$. Let $G_{i-1}(V, E_{i-1})$ be the digraph which is $(i-1)$-connected by adding edges to $G(V, E)$, and $m' = |E_{i-1}| = m + m_0 + \ldots + m_{i-1}$, at least m_i edges should be added to make G_{i-1} i-connected. We can get

$$\begin{aligned}\frac{m_i}{m'} &= \frac{m_i}{m + m_0 + \ldots + m_{i-1}} \\ &\geq \frac{\frac{\epsilon m}{k}}{m + \frac{(i-1)\epsilon m}{k}} \\ &= \frac{\epsilon}{k + (i-1)\epsilon}.\end{aligned}$$

So the first i satisfying $m_i \geq \frac{\epsilon m}{k}$ makes that G_{i-1} is $\frac{\epsilon}{k+(i-1)\epsilon}$-far from i-connectivity. □

Lemma 7 shows that G_{i-1} is ϵ'-far from being i-connected, where $\epsilon' = \frac{\epsilon}{k+(i-1)\epsilon}$. By Lemma 1, we know that $t_{out}(G_{i-1}) \geq \frac{\epsilon m}{2k}$; by Theorem 1 we know that there are at least $\frac{\epsilon m}{4k}$ out-tight sets which contain at most $\frac{4kn}{\epsilon m}$ vertices. So if we set the appropriate parameters, we could test the connectivity of G_{i-1} with our algorithms in Section VI.

However, the first i satisfying $m_i \geq \frac{\epsilon m}{k}$ and $m_0, m_1, ..., m_i$ are all unknown parameters, we do not know how many edges exactly should be added to G to transform it into G_{i-1}, and we also do not know the structure of G_{i-1}. As a consequence, we can only test G directly. Recap the technique we used in Section IV, we detect G is ϵ-far from being k-connected by searching the tight sets of G, or more gengrally, the sets S satisfying $N^+(S) < k$, which we call deficient fragments. And the more deficient fragments the digraph G has, the higher probability that our algorithms output REJECT. Since G is a subgraph of G_{i-1}, the number of deficient fragments of G is no less than that of G_{i-1}, which means that directly testing G with our algorithms can also achieve good results. We only need to reset the parameters in Algorithm 5. Firstly we change the loop times into $\frac{8kn}{\epsilon m}$ (cf. Line 1 in Algorithm 5), and set the bound t into $\frac{4kn}{\epsilon m}$ (cf. Line 3 in Algorithm 5). Then we get Theorem 4.

Theorem 4. *Let G be a digraph. All vertices in our algorithm will always output ACCEPT if G is k-connected, and at least one vertex outputs REJECT with probability at least $2/3$ if G is ϵ-far from being k-connected. The algorithm requires $O(\frac{c^k}{\epsilon^k})$ ($c > 1$ is a constant) rounds.*

IX. Conclusion

Based on the assumption that $t_{out}(G) \geq t_{in}(G)$, we first present a distributed algorithm to test k-connectivity of $(k-1)$-connected digraphs. Then we show that our algorithm is also suitable for digraphs which are not $(k-1)$-connected by adjusting some parameters. The design of the algorithm is exactly the same when $t_{in}(G) \geq t_{out}(G)$. The only difference is that we should consider the in-neighbors of vertices instead of out-neighbors in the above algorithms. Actually, we do not know which one is larger for a given digraph, so we need to run our algorithms twice for two situations. As far as we know, our algorithm is the first non-trivial distributed algorithm to test k-connectivity of digraphs with constant round complexity. And the methods of handling the issues of distributed k-vertex-connectivity testing of digraphs in this paper can be also adapted to the other three situations mentioned in Table I, and the corresponding results are listed in the Table.

References

[1] Noga Alon, Tali Kaufman, Michael Krivelevich, and Dana Ron. Testing triangle-freeness in general graphs. *SIAM Journal on Discrete Mathematics*, 22(2):786–819, 2008.

[2] Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. In *International Symposium on Distributed Computing In Proc.*, pages 43–56. Springer, 2016.

[3] Artur Czumaj and Christian Sohler. Testing expansion in bounded-degree graphs. *Combinatorics, Probability and Computing*, 19(5-6):693–709, 2010.

[4] Guy Even, Reut Levi, and Moti Medina. Faster and simpler distributed algorithms for testing and correcting graph properties in the congest-model. *arXiv preprint arXiv:1705.04898*, 2017.

[5] Orr Fischer, Tzlil Gonen, and Rotem Oshman. Distributed property testing for subgraph-freeness revisited. *arXiv preprint arXiv:1705.04033*, 2017.

[6] Pierre Fraigniaud, Pedro Montealegre, Dennis Olivetti, Ivan Rapaport, and Ioan Todinca. Distributed subgraph detection. *arXiv preprint arXiv:1706.03996*, 2017.

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

[8] András Frank and Tibor Jordán. Directed vertex-connectivity augmentation. *Mathematical programming*, 84(3):537–553, 1999.

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

[10] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing In Proc.*, pages 406–415. ACM, 1997.

[11] Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[12] Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 68–75. Springer, 2011.

[13] David R Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *SODA In Proc.*, volume 93, pages 21–30, 1993.

[14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing In Proc.*, pages 565–573. ACM, 2014.

[15] Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Structures & Algorithms*, 20(2):165–183, 2002.

[16] David Peleg. Distributed computing a locality sensitive approach. *SIAM Monographs on discrete mathematics and applications*, 5, 2000.

[17] Yuichi Yoshida and Hiro Ito. Testing k-edge-connectivity of digraphs. *Journal of Systems Science and Complexity*, 23(1):91–101, 2010.

[18] Yuichi Yoshida and Hiro Ito. Property testing on k-vertex-connectivity of graphs. *Algorithmica*, 62(3-4):701–712, 2012.