A Distributed Depth First Search based Algorithm for Edge Connectivity Estimation

Onur Ugurlu
Department of Fundamental Sciences
Izmir Bakircay University
Izmir, Turkey
onur.ugurlu@bakircay.edu.tr

Vahid Khalilpour Akram
International Computer Institute
Ege University
Izmir, Turkey
vahid.akram@ege.edu.tr

Deniz Türsel Eliiyi
Department of Industrial Engineering
Izmir Bakircay University
Izmir, Turkey
deniz.eliiyi@bakircay.edu.tr

***Abstract*—The edge connectivity of a network is the minimum number of edges whose removal disconnect the network. The edge connectivity determines the minimum number of edge-disjoint paths between all nodes. Hence finding the edge connectivity can reveal useful information about reliability, alternative paths and bottlenecks. In this paper, we propose a cost-effective distributed algorithm that finds a lower bound for the edge connectivity of a network via finding at most c depth-first-search trees, where c is the edge connectivity. The proposed algorithm is asynchronous and does not need any synchronization between the nodes. In the proposed algorithm, the root node starts a distributed depth-first-search algorithm, and the nodes select next node in the tree based on their available edges to maximize the total number of established trees. The simulation results show that the proposed algorithm finds the edge connectivity with an average of 48% accuracy ratio.**

***Index Terms*—Distributed Algorithms, Edge Connectivity, Depth First Search, Spanning Tree.**

I. Introduction

Distributed systems have a wide and increasing range of applications in various fields such as industrial automation, internet of things and intelligent structures [1]. The nodes in a distributed system communicate with other nodes by sending or receiving multi-hop messages. Each node forwards the incoming messages to its neighbors until the message arrive to the target node. Multi-hop communication simplifies the establishment and scalability of the distributed systems at the cost of reducing the reliability [2]. Failure of a node or lost links may affect the communication paths between other nodes. In the worst case, failure of a node or a link may cut all paths between a group of nodes and destroy the connectivity of the network.

A distributed system can be modeled as a graph $G(V, E)$ where V is the set of nodes and E is the set of links between the nodes. An edge can be added between nodes which have a communication link to each other. In graph theory, the edge connectivity of given graph is the minimum number of edges whose removal disconnect some nodes from the graph. The edge connectivity of a network may provide useful information about the connectivity robustness, bottlenecks, critical links and traffic flow of the network. Hence, efficient detection of edge connectivity of a network has been the subject of many studies. Fig.1 shows an example 2-edge connected network with 11 nodes. Removing the thick edges (4,5) and (1,6) separates the nodes into two disconnected parts. Similarly, the vertex connectivity of a graph is the minimum number of nodes that should be removed to disconnect some nodes from the graph. A higher edge or vertex connectivity value shows more connectivity robustness for the network.

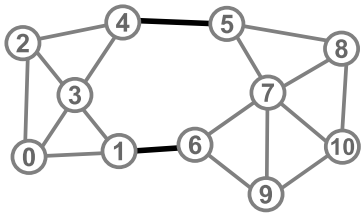

Fig. 1: A sample of 2-edge connected network.

Beside the reliability level, finding the edge connectivity of a network provides other useful information. The edge connectivity determines the minimum degree of nodes and the minimum number of edge-disjoint paths between the nodes. For example, in a c-edge connected network, each node has at least c neighbors and c edge-disjoint paths to any other node. Two paths are edge-disjoint if they have no common edge. Detecting the edge-disjoint paths between the nodes reveals the available alternative paths between the nodes, which is important for efficient routing algorithms. In this study, we propose an efficient algorithm that estimates a lower bound for the edge connectivity of a given network by finding at most c spanning trees. In each c-edge connected network, we have at most c-edge disjoint spanning trees. The spanning trees are edge disjoint if they have no common edge. Fig.2 shows a sample spanning tree rooted by node 0 in a 2-edge connected network, and the resulting disconnected network after removing the edges of the spanning tree. The established edge-disjoint spanning trees can also be used to find the edge-disjoint paths between the nodes.

The remaining parts of this paper have been organized as follows; Section II, provides a brief survey about the exiting works. Section III includes the details of the proposed approach. Section IV provides the experimental results of

978-3-903176-31-7

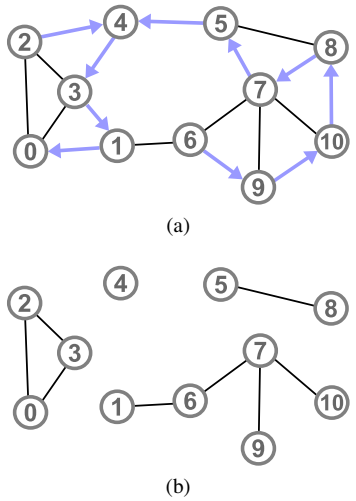

Fig. 2: A sample spanning tree and the resulting disconnected network after removing the edges of spanning tree.

implementing the proposed algorithm on sample networks. Finally, Section V contains the conclusion and future works.

II. Related Work

Finding the edge connectivity of a given graph is a well-known problem that has many central deterministic and randomized algorithms [3]–[9]. One of the first and efficient algorithms for edge connectivity detection is based on the maximum-flow problem. According to the max-flow min-cut theorem [10], the maximum flow that can pass between two arbitrary nodes $s \in V$ and $t \in V$ is equal to the minimum cut size between them. In an unweighted graph, the maximum possible flow between two nodes (assuming that the weight of each edge is 1) is equal to the number of edges in the minimum cut or edge connectivity. Therefore, to find the edge connectivity of a graph, one can find the maximum flow between all pairs of nodes and select the minimum among the detected values.

The maximum flow between two nodes can be found by the Ford-Fulkerson algorithm in $O((n+m)f)$, where f is the flow between the nodes, n is the number of nodes and m is the number of edges [11]. Therefore, finding the edge connectivity using the max-flow algorithm can be done in $O(n^2mf)$ time complexity. Usually, applying a central algorithm on a distributed environment imposes a large amount of message passing, because the entire topology of the network should be collected in single node. Goldberg and Tarjan proposed an asynchronous distributed maximum flow algorithm [12] that has $O(n^2)$ time complexity and $O(n^2m)$ message complexity.

Many other distributed algorithms have been proposed for edge and vertex connectivity detection [2], [13]–[16] that find the exact or approximate connectivity by sending many different messages between the nodes. In many applications, ensuring a certain level of reliability is enough for the correct working of the application. Hence, in these applications finding a lower bound for the connectivity can provide sufficient information on the reliability of the network. Most of these approaches find the edge-disjoint paths between all nodes and select the smallest value as the edge connectivity. This method requires at least $O(n^2)$ message complexity. Some other proposed distributed algorithms need to synchronize the nodes in time periods or rounds [17]–[20]. In these algorithms, all nodes are notified about the starting of a new round. In each round, the nodes run a few commands, exchange messages with their neighbors and wait for starting a new round. Synchronized algorithms require strict time synchronization, which is not supported in most distributed systems.

Another set of researches focus on detecting the cut edges or cut vertices in the networks [21]–[26]. A cut edge or bridge is an edge which in case of failure separates the network to the disconnected parts. Similarly a cut vertex is a node which its failure partitions the network to the disconnected parts. Generally, finding the cut edges or cut vertices is a restricted version of edge and vertex connectivity problem which has lower complexity.

In this paper we propose a cost-effective distributed algorithm that finds a lower bound for the edge connectivity by finding at most c spanning trees, where c is the edge connectivity of the graph. The proposed algorithm sends at most $O(\Delta n)$ messages with constant size where Δ is the maximum node degree in the network. The algorithm is straightforward and asynchronous and can be used in almost all types of multi-hop networks because it does not need any synchronization between the nodes. We provide a comparison between the performance of the proposed and existing algorithms by implementing them on 88 randoms graph with different node count and density, and measuring the estimated connectivity values.

III. The Proposed Approach

The proposed approach is based on the iterative establishment of the depth-first-search (DFS) trees in the network and eliminates the selected edges in each tree. The maximum number of DFS trees in each network is bounded by the edge connectivity of that network. Therefore, via finding the number of DFS trees, we can obtain a lower bound for the edge connectivity of that network. A traditional distributed DFS algorithm selects a random neighbor and continues searching until all nodes are visited [27]. In this algorithm, a root node r starts the algorithm and sends a token message to a randomly selected neighbor, say v. Node v then selects another unvisited neighbor (a neighbor that has not sent token yet) randomly and forwards the token message to that node. In this way, each node selects a random unvisited neighbor to forward the token message. Each node sets the sender of token message as its parent in the tree. If a node has no unvisited neighbor, it returns the token to its parent. If a parent node receives a token from a child node, it forwards the token to another unvisited neighbor (if any) or returns it to its parent. The algorithm terminates when the root node receives a token from a neighbor and has no unvisited neighbor. The details of the distributed DFS algorithm can be found in [27].

To have a good estimation about the edge connectivity of the network we should maximize the number of established DFS trees. To increase the number of detected DFS trees, we use a weighted DFS (WDFS) algorithm that selects the next node in the tree based on a weight value of the node. The basic idea is to establish a tree that covers all nodes with a minimum number of selected edges in each node. In other words, we want to establish a tree such that the maximum number of selected edges in each node is minimal. In this way, the number of available edges in each node for the next tree will be maximum, and the detected lower bound will be tighter. To achieve this result, we assign a weight value to each node, which initially are all 0.

After adding each node to the tree, we update the weight of its neighbors by adding i to their current weights, where i is the number of unvisited nodes in the graph. So, after adding the first node to the tree, the weight value of its neighbors is increased by $n-1$ where n is the number of nodes. After adding the i'th node to the tree, the weight of its neighbors is increased by $n-i$. To add a new node to the tree, each node selects its unvisited neighbor with the lowest weight until all nodes are visited. If a node has no unvisited neighbors, it returns the token message to its parent to let the search continue from there. The search finishes when the root node receives a token and has no other unvisited neighbors.

Figure3 illustrates the steps of the proposed algorithm on a sample network with 11 nodes. Initially, the weight of each node is 0. The algorithms starts at Node 0 as the root node, and increases the weight of its neighbors to 10 (the number of remaining unvisited nodes). Hence, the weight of nodes 1, 2 and 3 becomes 10. Since the weight of neighbors of node 0 is identical, node 0 selects a random neighbor, say node 1, to send the token message. After receiving the token message, node 1 increases the weight of its neighbors by 9, which leads to $w(3)=19$, $w(2)=19$, and $w(4)=9$. Node 1 selects node 4 as the next node, because it has the lowest weight among all neighbors of node 1. After receiving the token message, node 4 increases the weight of its neighbors by 8, which leads to $w(3)=27$ and $w(5)=8$. Since 5 has the lowest weight among the unvisited neighbors of node 4, it is selected as the next node in the tree. After receiving the token, node 5 increases the weight of its neighbors by 7, resulting in $w(7)=7$ and $w(8)=7$. As nodes 7 and 8 have identical weights, node 5 selects one randomly, say node 7, as the next node in the tree. Node 7 increases the weight of its neighbors by 6, and the algorithm continues in this fashion until all nodes are visited. Fig.3m shows the resulting WDFS tree.

After finding a tree, the edges used in the tree can be ignored. To do this, the nodes can mark their links used for sent/received token messages. By ignoring these links, the nodes remove them from the network and use other edges for the next trees. In this way, establishing each tree removes a complete edge-disjoint path between all nodes. The root node can start a new WDFS search after completing a successful search. Obviously, if the current search finishes without covering all nodes, then there is no need for a new

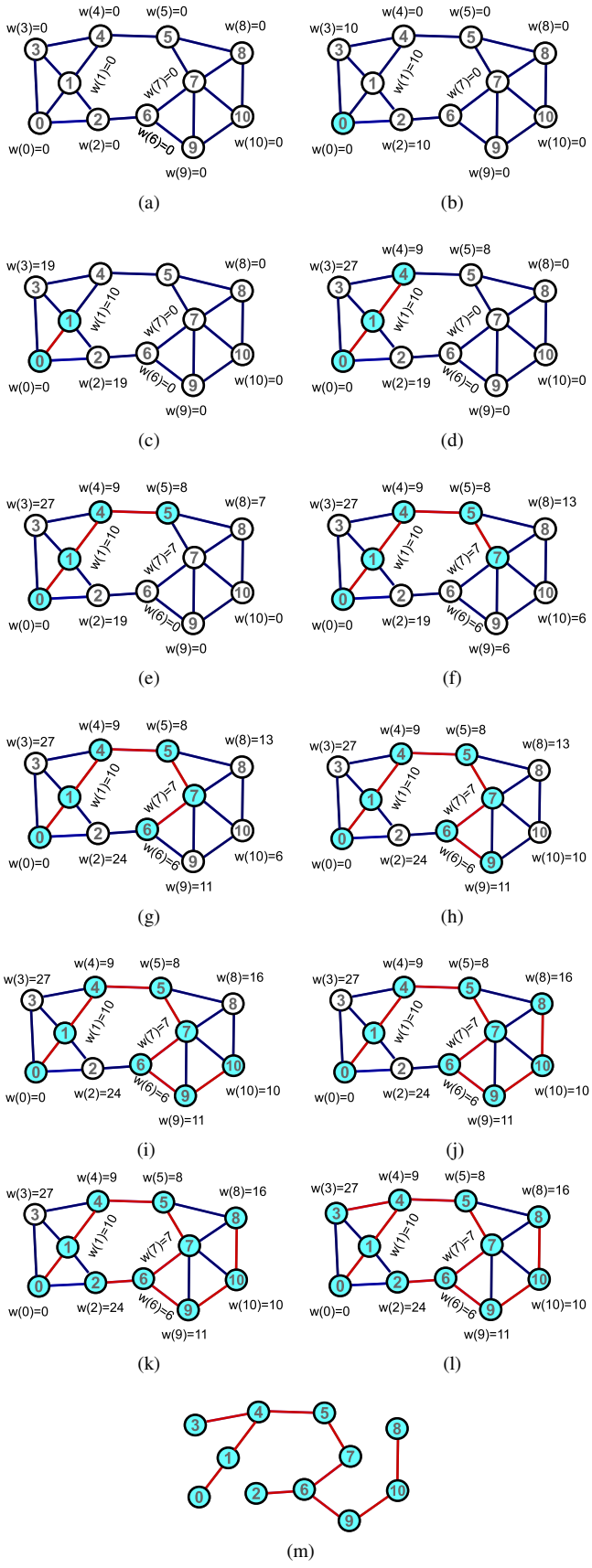

Fig. 3: The steps of the proposed algorithm.

search and the algorithm can be terminated. The number of successful searches determines the lower bound of the edge connectivity for the network. To control the number of covered nodes in each DFS tree, we may pass a visited node value in the token message. Each node that receives the token from a parent node can increase the visited node count by 1. When the token returns back to the root node, this node can compare the number of visited nodes by the total number of nodes in the network. If the number of visited nodes is equal to the total number of nodes, then root node can increase the detected edge connectivity value by 1 and start a new search. Otherwise, the root node can terminate the algorithm and reports the detected edge connectivity.

Algorithm 1, shows the steps of the proposed approach. Initially, the root node sets the $edgeConnectivity$ value to 0, and starts a distributed weighted algorithm to establish the WDFS tree T. After finding a tree, each node marks its selected local edges in the tree, to ignore them in the next iteration. Let $N(T)$ be the number of selected nodes in tree T. If $N(T) = |V|$, then T has covered all nodes in the network. In this case, the root node can increase the $edgeConnectivity$ by 1 and repeats the algorithm to find a new tree. The root node stops the algorithm when the established T has less than $|V|$ nodes, which means that some of the nodes were unreachable in the last search.

Algorithm 1: Edge Connectivity Estimation

1: *edgeConnectivity* $\leftarrow 0$.
2: **do**
3: Find a distributed weighted DFS tree T.
4: Mark the edges in T to ignore them in next iteration.
5: **if** $N(T) = |V|$ **then**
6: *edgeConnectivity* $\leftarrow$ *edgeConnectivity*+1.
7: **while** $N(T) = |V|$.

The proposed algorithm uses DFS, which has $O(2n-2)$ time complexity and $O(2n-2)$ message complexity [27]. After selecting each node, it broadcasts an index message, containing the remaining number of nodes in the tree, to update the weight of its neighbors. Each node should broadcast a notification message to its neighbors to notify them about its new weight value. So, each node broadcasts at most c index and Δ notification messages where c is the edge connectivity and Δ is the maximum nodes degree. Hence, the message complexity of the proposed algorithm is $O(2n-2+\Delta n+cn)$. Considering that Δ is an upper bound for c, the message complexity of the proposed algorithm is $O(\Delta n)$. Finding all DFS trees takes $O(\Delta n)$ time unit because each node sends at most Δ token messages to its neighbor nodes.

IV. Performance Analysis

In this section we present the simulation results. To test the performance of the WDFS, we compare the results with traditional DFS. Besides, we compute the edge connectivity of the network via maximum-flow algorithm for benchmarking purposes. For simulations, we use 88 random graphs [28] where the number of nodes varies from 100 to 200 and the density (the ratio of the number of edges and the number of all possible edges) varies from 30 to 90 for each graph size. Fig.4 shows the edge connectivity of the graphs. As the density of the graphs increases, naturally the edge connectivity increases at a certain rate.

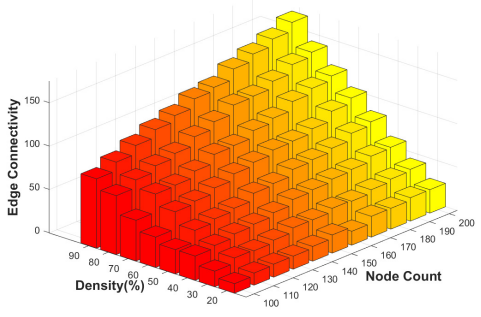

Fig. 4: Edge connectivity of the networks against the network density and number of nodes.

Fig.5 shows the numbers of edge-disjoint trees found by the traditional DFS algorithm for all instances. Note that, even if the maximum number of disjoint trees is found for a graph, it may be still smaller than the edge connectivity. For instance, the edge connectivity of a cycle graph with 3 nodes equals to 2, where the maximum number of the edge-disjoint trees equals to 1.

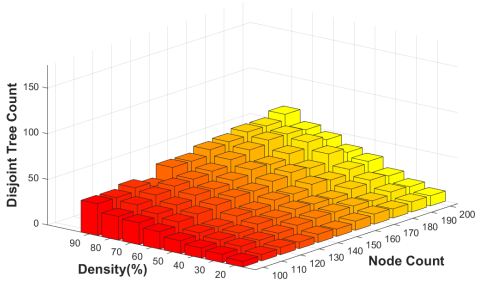

Fig. 5: Number of the disjoint trees found by the traditional DFS against the network density and number of nodes.

We present the number of the edge-disjoint trees found by the WDFS algorithm for all instances in Fig.6. It can be seen that the results of the traditional DFS are improved by this algorithm. For a more precise analysis, we average the results according to the density and size of the graphs.

Fig.7 depicts the average results for each density. The results indicate that the proposed algorithm finds more trees than the

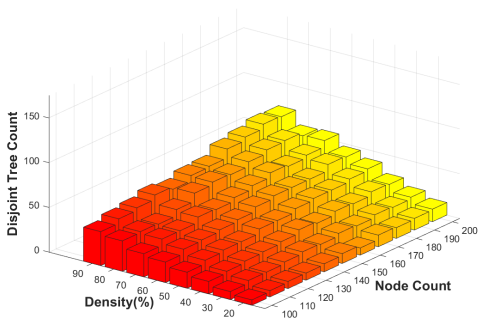

Fig. 6: Number of the disjoint tree found by the WDFS against the network density and number of nodes.

traditional DFS for each density. For densities 20% and 30%, both algorithms seem to perform similarly. However, when the density of the graph increases, the difference between the WDFS and the traditional DFS becomes clearer and WDFS dominates DFS.

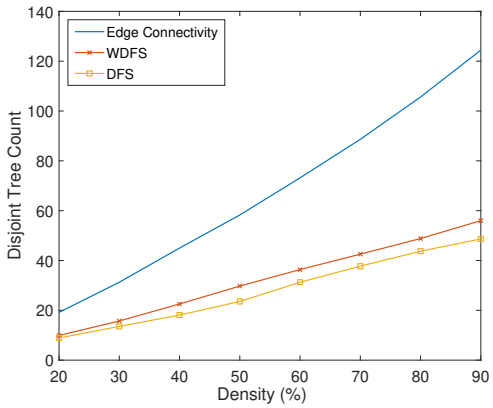

Fig. 7: Average number of detected disjoint trees against the network density.

Fig.8 shows the average results for each graph size. The proposed algorithm can find edge-disjoint trees as nearly half of the edge connectivity. For all instances, the WDFS finds 48% of the edge connectivity, whereas the traditional DFS finds 41%.

V. Conclusion

In this study, we focus on the edge connectivity problem in distributed networks. We propose a cost-effective distributed algorithm that finds a lower bound for the edge connectivity of a network by finding the edge-disjoint spanning trees. To find the edge-disjoint spanning trees, we use the depth-first-search algorithm. Moreover, we modify the traditional depth-first-search by using weighted nodes and improve its performance. The simulation results show that the proposed algorithm can find %48 of the edge connectivity. Using Golberg and Tarjan's

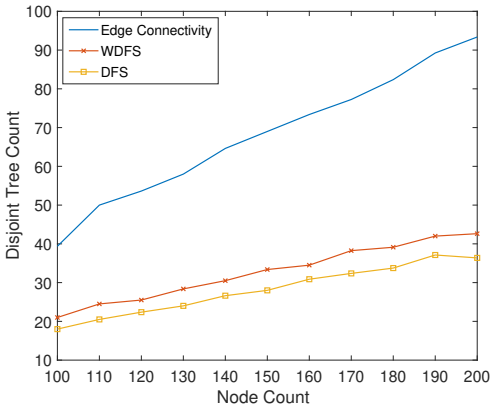

Fig. 8: Average number of detected disjoint trees against the number of nodes in the network.

distributed max-flow algorithm, edge connectivity of a network can be computed in $O(n^4)$ time complexity and $O(n^4m)$ message complexity. In contrast, our proposed algorithm needs $O(\Delta n)$ time complexity and $O(\Delta n)$ message complexity to find a lower bound for the edge connectivity of a given network.

Simple heuristics with reasonable time and message complexity can be developed to improve the lower bound for edge connectivity. For instance, before the construction of a tree an algorithm can be used for selecting the root of the tree, or using the information from previous searches may bring improvement. As a future work, we plan to combine our algorithm with fast heuristics to increase the number of the edge-disjoint spanning trees detected. Finally, investigation of the performance of the proposed algorithm on connected unit disk graphs may be worth studying.

References

[1] Y. Al Mtawa, A. Haque, and B. Bitar, "The mammoth internet: Are we ready?" *IEEE Access*, vol. 7, pp. 132 894–132 908, 2019.

[2] V. K. Akram, "An asynchronous distributed algorithm for minimum s-t cut detection in wireless multi-hop networks," *Ad Hoc Networks*, vol. 101, p. 102092, 2020.

[3] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," in *Combinatorial Optimization Eureka, You Shrink!* Springer, 2003, pp. 31–33.

[4] J. Hao and J. B. Orlin, "A faster algorithm for finding the minimum cut in a directed graph," *Journal of Algorithms*, vol. 17, no. 3, pp. 424–446, 1994.

[5] H. Nagamochi, T. Ono, and T. Ibaraki, "Implementing an efficient minimum capacity cut algorithm," *Mathematical Programming*, vol. 67, pp. 325–341, 1994.

[6] M. Stoer and F. Wagner, "A simple min-cut algorithm," *Journal of the ACM (JACM)*, vol. 44, no. 4, pp. 585–591, 1997.

[7] M. Brinkmeier, "A simple and fast min-cut algorithm," *Theory of Computing Systems*, vol. 41, no. 2, pp. 369–380, 2007.

[8] D. R. Karger and C. Stein, "A new approach to the minimum cut problem," *Journal of the ACM (JACM)*, vol. 43, no. 4, pp. 601–640, 1996.

[9] A. A. Benczúr and D. R. Karger, "Approximating st minimum cuts in õ (n 2) time," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 1996, pp. 47–55.

[10] G. Dantzig and D. R. Fulkerson, "On the max flow min cut theorem of networks," *Linear inequalities and related systems*, vol. 38, pp. 225–231, 2003.

[11] L. R. Ford Jr and D. R. Fulkerson, *Flows in networks*. Princeton university press, 2015.

[12] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *Journal of the ACM (JACM)*, vol. 35, no. 4, pp. 921–940, 1988.

[13] P. Szczytowski, A. Khelil, and N. Suri, "Dkm: Distributed k-connectivity maintenance in wireless sensor networks," in *2012 9th Annual Conference on Wireless On-Demand Network Systems and Services (WONS)*. IEEE, 2012, pp. 83–90.

[14] V. K. Akram and O. Dagdeviren, "Deck: A distributed, asynchronous and exact k-connectivity detection algorithm for wireless sensor networks," *Computer Communications*, vol. 116, pp. 9–20, 2018.

[15] M. Jorgic, N. Goel, K. Kalaichevan, A. Nayak, and I. Stojmenovic, "Localized detection of k-connectivity in wireless ad hoc, actuator and sensor networks," in *2007 16th International Conference on Computer Communications and Networks*. IEEE, 2007, pp. 33–38.

[16] O. Dagdeviren and V. K. Akram, "Pack: Path coloring based k-connectivity detection algorithm for wireless sensor networks," *Ad Hoc Networks*, vol. 64, pp. 41–52, 2017.

[17] M. Ghaffari and F. Kuhn, "Distributed minimum cut approximation," in *International Symposium on Distributed Computing*. Springer, 2013, pp. 1–15.

[18] D. Nanongkai and H.-H. Su, "Almost-tight distributed minimum cut algorithms," in *Int. Symp on Distributed Computing*. Springer, 2014, pp. 439–453.

[19] H.-H. Su, "A distributed minimum cut approximation scheme," *arXiv preprint arXiv:1401.5316*, 2014.

[20] M. Daga, M. Henzinger, D. Nanongkai, and T. Saranurak, "Distributed edge connectivity in sublinear time," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2019, pp. 343–354.

[21] V. K. Akram and O. Dagdeviren, "Breadth-first search-based single-phase algorithms for bridge detection in wireless sensor networks," *Sensors*, vol. 13, no. 7, pp. 8786–8813, 2013.

[22] X. Liu, L. Xiao, and A. Kreling, "A fully distributed method to detect and reduce cut vertices in large-scale overlay networks," *IEEE Transactions on Computers*, vol. 61, no. 7, pp. 969–985, 2011.

[23] M. Ahuja and Y. Zhu, "An efficient distributed algorithm for finding articulation points, bridges, and biconnected components in asynchronous networks," in *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 1989, pp. 99–108.

[24] P. Chaudhuri, "An optimal distributed algorithm for computing bridge-connected components," *The Computer Journal*, vol. 40, no. 4, pp. 200–207, 1997.

[25] D. Pritchard, "An optimal distributed bridge-finding algorithm," in *Proceedings of the 25th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Denver, CO, USA*, vol. 2326. Citeseer, 2006.

[26] M. H. Karaata and P. Chaudhuri, "A self-stabilizing algorithm for bridge finding," *Distributed Computing*, vol. 12, no. 1, pp. 47–53, 1999.

[27] K. Erciyes, *Distributed graph algorithms for computer networks*. Springer Science & Business Media, 2013.

[28] M. E. Berberler and Z. N. Berberler, "Measuring the vulnerability in networks: a heuristic approach," *Ars Combinatoria*, vol. 135, pp. 3–15, 2017.