Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 985 – 997

doi:10.3906/elk-1801-49

TAPU: Test and pick up-based *k*-connectivity restoration algorithm for wireless sensor networks

Vahid KHALILPOUR AKRAM, Orhan DAĞDEVİREN*
International Computer Institute, Ege University, İzmir, Turkey

Received: 05.01.2018 • **Accepted/Published Online:** 10.12.2018 • **Final Version:** 22.03.2019

Abstract: A k-connected wireless sensor network remains connected if any k-1 arbitrary nodes stop working. The aim of movement-assisted k-connectivity restoration is to preserve the k-connectivity of a network by moving the nodes to the necessary positions after possible failures in nodes. This paper proposes an algorithm named TAPU for k-connectivity restoration that guarantees the optimal movement cost. Our algorithm improves the time and space complexities of the previous approach (MCCR) in both best and worst cases. In the proposed algorithm, the nodes are classified into safe and unsafe groups. Failures of safe nodes do not change the k value of the network while failures of unsafe nodes reduce the k value. After an unsafe node's failure, the shortest path tree of the failed node is generated. Each node moves to its parent location in the tree starting from a safe node with the minimum moving cost to the root. TAPU has been implemented on simulation and testbed environments including Kobuki robots and Iris nodes. The measurements show that TAPU finds the optimum movement up to 79.5% faster with 50% lower memory usage than MCCR and with up to 59% lower cost than the greedy algorithms.

Key words: Wireless sensor networks, network connectivity, movement-assisted k-connectivity restoration, fault tolerance, reliability

1. Introduction

Wireless sensor networks (WSNs) are widely used in military, industry, health care, intelligent structures, and many other applications. Connectivity maintenance in WSNs is an important task because the nodes usually communicate with each other using some intermediate nodes and a failure in a node may cut off the connection between other nodes. Most of the times, there is a specific sink node in WSNs that collects information or sends commands to other nodes. Losing one or more nodes may create unreachable partitions from the sink and waste many active resources in the network.

Many approaches have been proposed to increase the fault tolerance of unreliable WSNs [24]. A network is 1-connected if there is a node whose failure divides the network into disconnected parts. In a k-connected network at least k node failures can partition the network into disconnected parts. Besides reliability, k-connected networks have many interesting properties. In a k-connected network, there are at least k disjoint paths between each pair of nodes; hence, fundamental tasks such as routing, load balancing, backbone construction, and topology control can be implemented in a more efficient manner. The minimum node degree in k-connected networks is k and usually higher k values lead to denser networks. The k values are useful data in medium access control (MAC) and clustering protocols.

*Correspondence: orhan.dagdeviren@ege.edu.tr

This paper focuses on the movement-assisted k-connectivity restoration problem for general k values. The movement-assisted k-connectivity restoration is the process of moving nodes in order to preserve the current k value of the network after failures in nodes. To the best of our knowledge, only one algorithm named minimum cost k-connectivity restoration (MCCR) has been proposed for the generalized movement-assisted k-connectivity restoration problem [30]. This paper investigates the operation of the MCCR algorithm and proposes a more efficient algorithm (TAPU) for the same problem in terms of time and space consumption. The proposed algorithm uses a k-connectivity testing algorithm to check the current k value of the network after a failure in a node. If a node failure reduces the k value, the proposed algorithm selects the best possible movement strategy and restores the k value. By removing the matching operation from the k-connectivity restoration process, our proposed approach runs up to 79.5% faster and uses about 50% less memory than MCCR. In summary, the main contributions of this paper are as follows:

1. We propose a k-connectivity restoration algorithm, TAPU, that uses less time and space then the optimal algorithm.

2. We provide time and space complexities of TAPU and show that TAPU asymptotically improves these complexities of the existing optimal algorithm.

3. We evaluate the performances of TAPU and other algorithms through simulations on various random topologies and testbed experiments using IRIS motes and Kobuki robots. Measurements clearly show the superior performance of TAPU compared to the other algorithms.

The remaining parts of the paper are organized as follows: Section 2 provides a brief survey on related works. Section 3 includes the problem definition and theoretical background. Descriptions of the proposed algorithm are presented in Section 4. Section 5 includes the theoretical analysis and Section 6 presents the experimental testbed and simulation analysis. Finally, the conclusions are drawn in Section 7.

2. Related work

Various aspects of k-connectivity problems have been the subject of many studies. For example, given the network area, number of nodes, radio range of each node, and the distribution model, various relations have been proposed to estimate the probability of k-connectedness in WSNs [15, 22, 36–38]. Determining the minimum radio power of nodes in a given network to achieve k-connectivity is another interesting problem that has been studied in different works [10, 16, 19, 23, 26, 29, 34]. Some other studies focused on node deployment methods to establish k-connected networks with maximum covered area [4–6, 31, 33]. Another well-known problem about k-connectivity is finding the current k value of an existing network with minimum energy consumption.

Efficient central algorithms have been proposed for finding the k value of a graph in polynomial time [11, 12, 14, 17]. The proposed algorithms for k-connectivity detection of WSNs estimate the k value from local neighborhood information with reasonable energy consumption [8, 20, 28], or consume more energy to find more accurate k values [2, 7, 9].

The k-connectivity restoration is another interesting problem. Failure of some nodes can reduce the k value of a network and can decrease the network reliability. In the k-connectivity restoration problem, the aim is to restore the k value after failures in one or more nodes. Relay nodes' placement [3, 25, 35] and mobile nodes' movement [30] are approaches that potentially can solve this problem.

This paper focuses on the movement-assisted k-connectivity restoration problem for general k values. There are many studies on the movement-assisted k-connectivity restoration problem for $k = 1$ [18, 27] or for other specific k values [1, 32]. However, the generalized movement-assisted k-connectivity restoration is still an open problem and, to the best of our knowledge, only the MCCR algorithm has been proposed for this problem [30].

The MCCR algorithm finds an optimal movement among the nodes for k-connectivity restoration by calling a k-connectivity testing and a maximum weighted bipartite matching algorithm. After a failure in a node, the algorithm is called n times as $MCCR(G, V, P)$, where G is the graph without the crashed node, V is the set of remaining $|V| - 1$ active nodes, and P is the set of $|V| - 1$ possible positions for remaining nodes. There are $|V|$ possible P sets where MCCR tests all of them. In each run, MCCR checks whether the positions in P form a k-connected graph. This is done by calling a k-connectivity testing algorithm on G and P. If the current positions do not lead to a k-connected graph, the execution is aborted and MCCR checks the next position set. Otherwise, the nodes in V are matched with the positions in P using a maximum weighted bipartite matching algorithm where for any $v \in V$ and $p \in P$ the weight of each edge (v, p) is the reverse cost of moving node v to position p. Finally, a match with the highest value is selected and the nodes are moved to their matched positions. In this paper, it is shown that our proposed algorithm outperforms MCCR both theoretically and experimentally in terms of time and space. Moreover, MCCR performs better than two implemented greedy k-connectivity restoration approaches, which will be explained in Section 6.2.

3. Problem description

A WSN can be modeled as an undirected weighted graph $G = (V, E, w)$, where V is the set of nodes (vertices), E is the set of edges, and $w : E \to Z^+$ is the weight function. The weight of each edge is the cost of movement between two endpoints of an edge, which depends on various parameters such as distance, ground, obstacles, slope, etc. Figure 1 shows a sample 2-connected WSN where there is a an edge with a moving cost between the nodes that are located in the radio ranges of each other. To keep the simplicity, only the radio ranges of 3 nodes are shown in Figure 1. It is generally assumed that a node must move to the positions previously occupied by other nodes [30]. This assumption preserves the coverage area of the network and also keeps the problem tractable. A node failure may affect the k value of the network. For example, in Figure 1, failure of any gray node reduces k to 1 but a failure in any white node has no effect on k. Therefore, the vertices of any graph can be divided into *safe* and *unsafe* groups as in Definition 1.

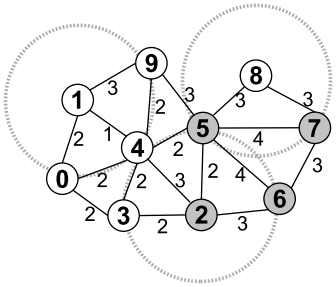


Figure 1. A 2-connected WSN.

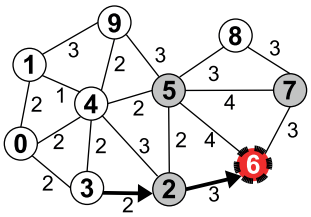


Figure 2. Nodes 3 and 2 move toward node 6.

Definition 1 *Node $v \in V$ is a safe node if G/v has the same k value as G. Otherwise, it is an unsafe node.*

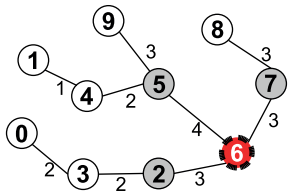


Figure 3. Shortest path tree of node 6.

No restoration is required when a safe node stops working because the k value remains unchanged. In the rest of this paper, the set of safe nodes is referred to by S. Any node $v \notin S$ is an unsafe node whose failure reduces the k value by 1. Figure 2 shows the optimum movements to restore k-connectivity after a failure in node 6 where node 3 moves to node 2's position and node 2 moves to node 6's position. As mentioned, the MCCR provides optimal movement cost, so our objective is to design an algorithm that finds optimal movements the same as MCCR at the same time consuming less time and space than MCCR.

4. Proposed algorithm

After a node failure in a k-connected network, the first question in k-connectivity restoration is whether the restoration process finishes immediately if the graph (excluding the failed node) is k-connected. If the graph is not k-connected, one of the other nodes should move to the position of the crashed node. Clearly, only one of the safe nodes can leave its position without reducing k. The optimal movement can be achieved by creating a shortest path tree (SPT) rooted from a failed node. Figure 3 shows the SPT of failed node 6. After finding the SPT of failed node v_f, the restoration procedure can select the nearest safe node $s \in S$ to v_f and move each node to its parent location from s to v_f in the tree as given in Observation 1.

Observation 1 *Moving nodes to their parent location, starting from the nearest safe node up to the root of the SPT of the failed node, is an optimal movement for k-connectivity restoration.*

Let v_f be the failed node, $T(v_f)$ be the SPT of v_f, and v_n be the nearest safe node to the root in $T(v_f)$. Suppose that moving the nodes among the path from v_n to v_f in $T(v_f)$ is not optimal. In this case, either there exists another safe node v where moving the nodes between v and v_f in $T(v_f)$ is optimal or there is another path between v_n and v_f in G where moving nodes among it provides the optimal result. v_n is the nearest safe node to v_f and v is farther than v_n from the root; hence, v cannot provide a shorter movement to v_f. Also, in $T(v_f)$, the path between v_f and any other node is the shortest possible path; hence, it is impossible to find a shorter path between v_n and v_f in G.

In Figure 2, if node 6 stops working, the optimal restoration is moving nodes 2 and 3 to their parent locations because these nodes are the nearest safe nodes to node 6 in its SPT. This paper presents a k-connectivity restoration algorithm named test and pick up (TAPU) that accepts a graph G, the connectivity value k, and the failed node v_f. If the k-connectivity restoration is impossible, the algorithm returns $false$; otherwise, it restores k with optimal movements and returns $true$.

"Algorithm TAPU" above shows the steps of proposed algorithm. The algorithm returns *true* in successful restoration and *false* in failed cases. In the first step, TAPU removes the failed node v_f from G and tests whether the new graph $\bar{G}$ is k-connected (lines 2 and 3). If $\bar{G}$ is k-connected, the algorithm immediately returns $true$, indicating that no movement is required and G/v_f is already k-connected. Otherwise, we create the set $\bar{V}$,

Algorithm TAPU (G, V, E, v_f)

1: **Begin**
2: $\bar{G} \leftarrow G/v_f$.
3: **if** $test_k(\bar{G}, k) =$ true **then return** $true$.
4: $\bar{V} \leftarrow \{v_f\}$, $V \leftarrow V/v_f$, $Parent[\,1 \,..\, |V|\,] \leftarrow$.
5: **while** $|V| > 0$ **do**
6: **select** an edge $(i, j) \in E$ with **minimum** $w(i, j)$ **such that** $i \in V$ and $j \in \bar{V}$.
7: $Parent[i] \leftarrow j$, $G' \leftarrow \bar{G}/i$.
8: **if** $test\_k(G', k) = true$ **then**
9: $v \leftarrow i$.
10: **while** $v \neq v_f$ **do**
11: **move** v **to** the position of $Parent[v]$.
12: $v \leftarrow Parent[v]$.
13: $G \leftarrow G'$.
14: **return** $true$.
15: **else** $V \leftarrow V/i$, $\bar{V} \leftarrow \bar{V} \cup i$.
16: **return** $false$.
17: **End.**

which initially includes v_f, and then remove v_f from the node set V. $\bar{V}$ is the set of added nodes to SPT. Also, we create an array to keep the parent of each node in the SPT. The initial value of all elements in the *Parent* array is set to empty (line 4). In the *while* loop we select the edge with the minimum cost that connects node $i \in V$ to node $j \in \bar{V}$. In this way, we extend the SPT to cover one more node from G. Obviously, in the first iteration i will be one of the one-hop neighbors of v_f with the minimum moving cost (lines 5 and 6). Then we set node j as the parent of node i and remove node i from $\bar{G}$ (line 7). If the remaining graph G' is k-connected then i is a safe node. In this case the algorithm moves the nodes to their parent locations starting from i up to the failed node and returns *true* (lines 8–14). Otherwise, i is an unsafe node and we should continue to expand the SPT. Thus, i is removed from V and is added to $\bar{V}$ and the loop repeats to select another edge (lines 15 and 16). TAPU continues to select the edges and expands the SPT until a safe node is detected or all nodes from V are added to the SPT. The algorithm returns *false* if V becomes empty without finding a safe node (line 17). As we mentioned in Observation 1, TAPU guarantees the optimal movements for k-connectivity restoration.

5. Theoretical analysis

The time complexity of the MCCR algorithm is $\theta(n(test + match)) = \theta(n(mnk + n^2 log_2 n + mn)) \in \theta(n^2 mk + n^3 log_2 n)$, where $test$ is the time complexity of the k-connectivity testing algorithm [17], $match$ is the time complexity of the maximum weighted bipartite graph matching algorithm [21][13], n is the node count, and m is the edge count.

At the best case, if $\bar{G}$ is k-connected, only the testing algorithm is called once in TAPU, and therefore the best-case time complexity of TAPU is $O(test) = O(mnk)$. If $\bar{G}$ is not k-connected, TAPU creates a SPT and calls the k-connectivity testing algorithm after adding each node to the SPT. The maximum iteration number of the *while* loop is n and the maximum cost of the *select* operation is m. Therefore, the time complexity of TAPU in the worst case is $O(n(m + mnk)) = O(n^2 mk)$.

If we assume that the memory is freed after an execution of an algorithm and ignore the needed space for constant number of variables used in the algorithms, then the required memory for the implementation of the MCCR and TAPU algorithms is $max(test_{sp}, match_{sp}) = (2m + 3n)$ and $max(test_{sp}, tree_{sp}) = (m + n)$, respectively ($test_{sp}$, $match_{sp}$, and $tree_{sp}$ are the spaces required to execute test, matching, and SPT algorithms, respectively). For dense connected graphs with $m \in O(n^2)$ edges, the required space of our algorithm is about half that of MCCR. For sparse connected graphs with $m \in O(n)$ edges, the required space of MCCR reaches up to approximately 2.5 times of those of our algorithm. The time complexity in the best case has a significant asymptotic improvement from $\theta(n^2mk+n^3log_2n)$ to $O(mnk)$ and in the worst case it is asymptotically improved from $\theta(n^2mk + n^3log_2n)$ to $O(n^2mk)$.

6. Experimental evaluation

To evaluate the proposed algorithm, we implemented TAPU and MCCR on testbed networks. The Hungarian algorithm was used as the bipartite matching algorithm [21] and Henzinger et al.'s algorithm was used as the k-connectivity testing algorithm [17].

6.1. Testbed experiments

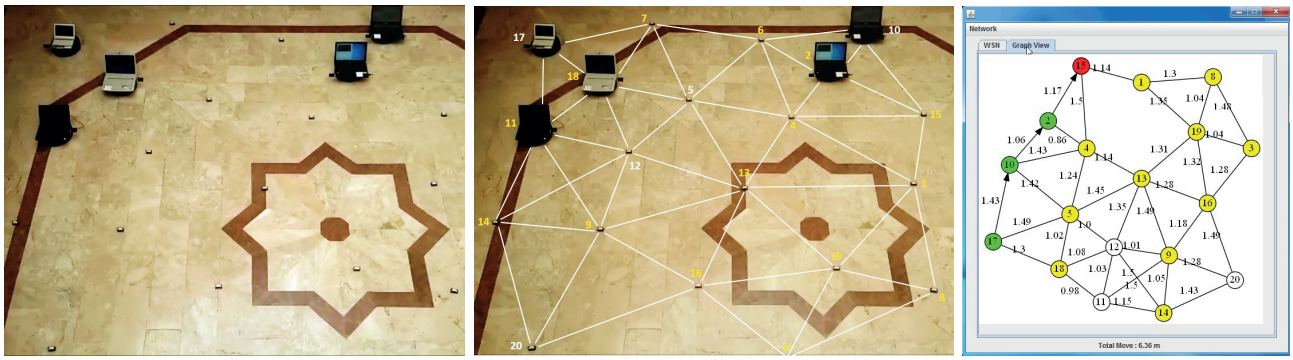
We created a testbed for our experiments with 20 Crossbow IRIS motes and 5 iClebo Kobuki robots. IRIS motes (Figure 4a) have 128 kB of programmable flash memory, 8 kB of RAM, and a 2.4-GHz IEEE 802.15.4 compliant transceiver with 250-kbps nominal transmit data rate and about 50-m indoor transmission range. IRIS motes support the TinyOS operating system and can connect to USB ports over the MIB520 gateway.

Kobuki is a mobile robot designed for research and educational purposes. It has an accurate rotation and navigation system that allows to specify the target position as polar coordinates. Kobuki robots support 70 cm/s maximum translational velocity and 180 degrees/s maximum rotational velocity and can carry up to 5-kg payloads. They provide USB and parallel ports for communications and operate up to 4 h after full charging. The developed firmware for Kobuki robots provides a C++ driver library that allows to send commands and receive the current status of the robot (speed, velocity, degree, etc.) from the Linux and Windows operating systems. Figure 4b shows a Kobuki robots.

(a) (b) (c)

Figure 4. a) An IRIS mote. b) A Kobuki robot. c) A mobile node in the network.

In our testbed, we connected an IRIS mote to a Kobuki robot using a small laptop (Figure 4c). The laptop acts as a communication bridge between the IRIS mote and Kobuki robot. We developed a Java program to receive the incoming packets from IRIS nodes and convert them to appropriate moving commands for the Kobuki robot. Also, we implemented a C++ controlling program to accept the moving command from a TCP port and deliver it to the Kobuki robot. According to the incoming messages from the IRIS mote, the Java program sends a moving command with the target position to the opened TCP socket by the robot control program. Upon receiving a move command, the C++ program calculates the distance and degree between current and target positions and sends the required rotation and translation command to Kobuki. In this way, the sink node can move any mobile node to desired position (x, y) by sending a *move(x,y)* message to it. Also, using the broadcast beacon and move messages, the Java program monitors the topology of the network and provides a real-time report for the failed nodes, movements, and sent/received bytes.



(a) (b) (c)

Figure 5. a) A sample WSN deployment. b) The established links between nodes. c) Monitoring program.

Figure 5a shows one of our established WSNs and Figure 5b shows the existing links between the nodes in this network. In the established WSNs, the sink node sends the position of each node before starting the algorithm and each node periodically (every 2 s) broadcasts a beacon packet to inform its neighbors that it is still alive. To create the desired topologies in the department building, we set the maximum communication ranges of all nodes as 1.5 m. The nodes that receive beacon packets from their neighbors calculate their distance to the sender node from incoming coordinates and discard the message if this distance is larger than 1.5 m.

If the sink does not receive a beacon from a node for 10 s (5 periods), it assumes that the node has stopped working and calls the restoration procedure. The sink node sends the network topology and the failed node id as parameters to the restoration procedure. According to the result of the restoration algorithm and the selected movement model, the sink may ignore the node failure or send a move message to a subset of active nodes. Node failures and movements are displayed by the connected Java programs to the sink node. Figure 5c shows how the program monitors the failed (red), safe (white), unsafe (yellow), and moving (green) nodes. The cost of edges between the nodes has been considered as the distance between them.

We created 10 random topologies with various k values from 1 to 5 (two topologies for each k) and used the carrier sense multiple access collision avoidance (CSMA/CA) MAC protocol with up to 1000 ms of random time to avoid the packet collision. We determined 4 random nodes for failure before the experiment and put the mobile nodes into the positions of nodes that are selected by the algorithm for moving.

The average movements of TAPU and MCCR were equal (about 8.3 m) in all topologies, as expected.

We measured the wall-clock time of the algorithms between failure detection and moving nodes' selection. We did not add the real departure time of nodes to wall-clock time because the departure time depends on various parameters including the hardware capabilities of nodes, initial acceleration, maximum speed, terrain conditions, and middleware (firmware) performances. The average wall-clock time of MCCR was about 0.39 s, which is at least 1.4 times higher than that of TAPU. The wall-clock time of TAPU was about 0.27 s.

6.2. Simulations

To compare their performances on larger networks, we implemented TAPU, MCCR, and two greedy algorithms in the simulation environment. After the failure of an unsafe node, in the greedy algorithms, a neighbor of the failed node is selected for moving to the failed node's position. The restoration process terminates if the network becomes k-connected. Otherwise, a neighbor of the moved node is selected for moving and this process continues until the network becomes k-connected. In the select minimum cost (SMC) algorithm, the neighbor with the minimum moving cost to the target position is selected for moving. In the select minimum degree (SMD) algorithm, the neighbor of the target node position that has the minimum degree is selected. Figure 6a shows a possible movement in the SMC algorithm and Figure 6b shows a possible movement in the SMD algorithm after failure of node 6.

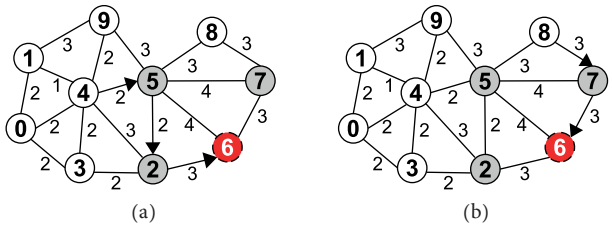


Figure 6. a) Possible movements in SMC algorithm. b) Possible movements in SMD algorithm.

We generated various random topologies for WSNs with different node counts and k values. The node counts in generated random topologies are 50, 100, 150, and 200. Generated topologies have different k values, which vary from 1 to 5. We generated 10 random instances of each topology type. Each algorithm has been tested on all topologies with different failed nodes. We run each algorithm $n/2$ times on each topology and each time a new node is selected as the failed node. In this way, the failure of half of the nodes in each topology is simulated.

Figure 7 shows the average moved distance against the node count in the TAPU and MCCR algorithms. Generally, the unsafe node counts in the higher k values are more than those of networks having smaller k values. Hence, for higher k values, both algorithms have moved more nodes to restore the k-connectivity. By increasing the node count, the number of safe nodes in the network increases and the moved distance becomes smaller.

Figure 8 shows the average moved distance in the MCCR, TAPU, SMD, and SMC algorithms against the node count. In networks with 50 nodes, the average moved distance of the TAPU and MCCR algorithms is about 23 m. This value for the SMD and SMC algorithms is higher than 27 m. In networks with 200 nodes, the average moved distance of TAPU and MCCR is less than 10 m while greedy algorithms produce movements of more than 22 m. Increasing the node count increases the probability of finding a safe node near the failed node and reduces the average moved distance. Figure 8 shows that the proposed algorithm finds the optimum movement, similar to MCCR, and produces up to 59% lower cost than the greedy algorithms.

Figure 9 shows the average moved distance in the MCCR, TAPU, SMD, and SMC algorithms against the k value. For $k = 1$, the average moved distance in TAPU and MCCR is about 9 m, while this value for the greedy algorithms is higher than 18 m. In the 5-connected networks, the average moved distance of the TAPU and MCCR algorithms is about 30 m, while greedy algorithms produce movements of more than 46 m. Figure 9 shows that for all k values, the cost of generated movements by the proposed and MCCR algorithms is at least 34.7% lower than the generated movements by the greedy algorithms.

Figure 10 shows the number of moved nodes in the MCCR and TAPU algorithms for each k value. This figure shows that the moved node counts for smaller k values are lower than those for higher k values. Generally, with small k values, finding a safe node near the failed node is easier because with larger k values, most of the nodes are used to create alternative disjoint paths between other nodes and this reduces the chance of finding a safe node near the failed node. In networks with 50 nodes and $k = 1$, the algorithms have moved about 2.6 nodes on average, while this value is 13.2 for $k = 5$. By increasing the number of nodes in the network, the probability of finding a safe node is increased; hence, the moved node count for restoration is decreased. The average number of moved nodes in the networks with 200 nodes and $k = 5$ is 1.3, while this value for $k = 5$ is 9.1. Generally, Figure 10 shows that the k value has a considerable effect on the number of moved nodes in the k-connectivity restoration process.

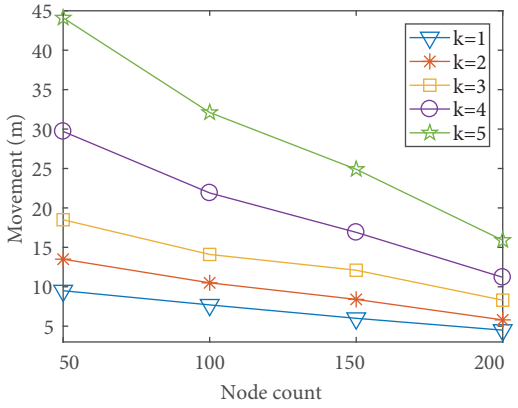


Figure 7. Average movements against the node counts.

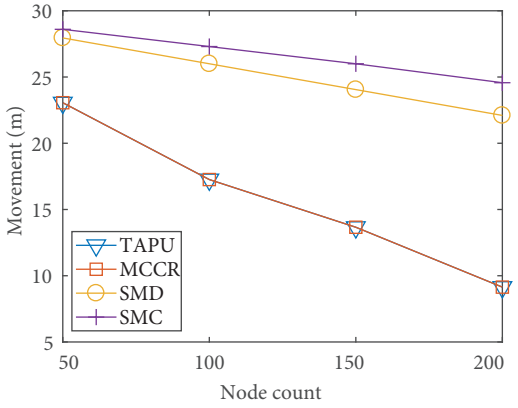


Figure 8. Number of moved nodes against the node counts.

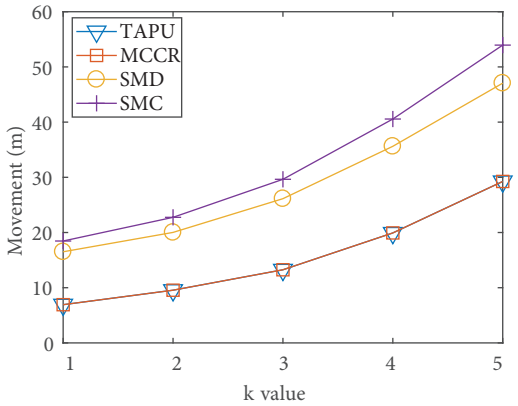


Figure 9. Average movements against the node counts.

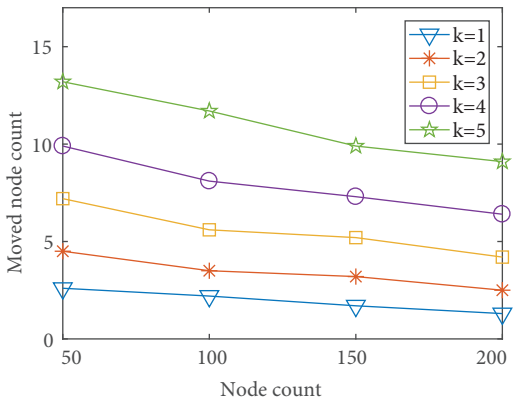


Figure 10. Number of moved nodes against the node counts.

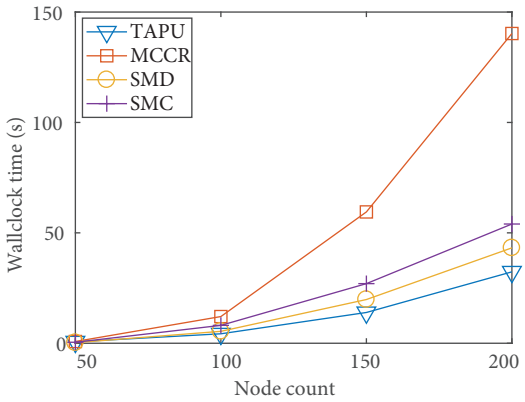


Figure 11. Wall-clock times against node counts.

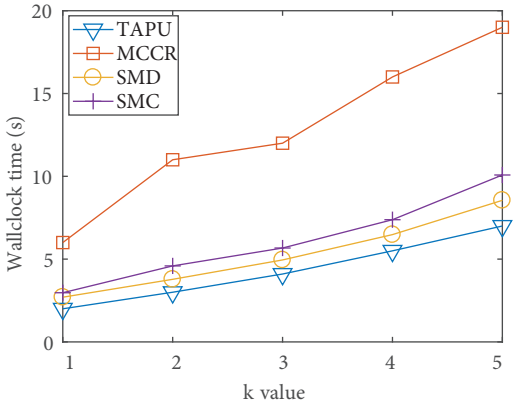


Figure 12. Wall-clock times against k values.

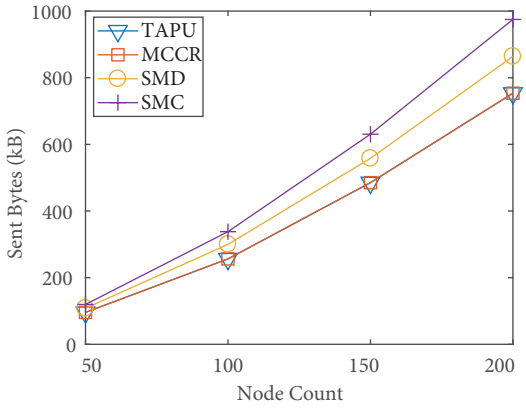


Figure 13. Wall-clock times against node counts.

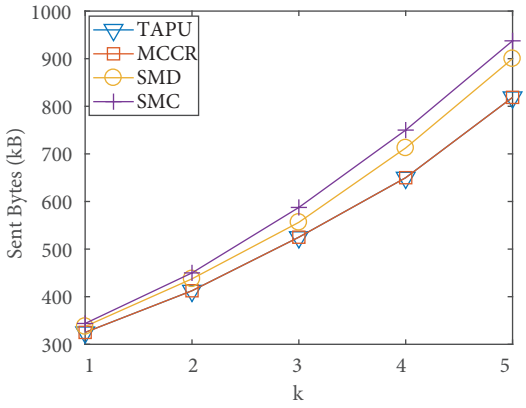


Figure 14. Wall-clock times against k value.

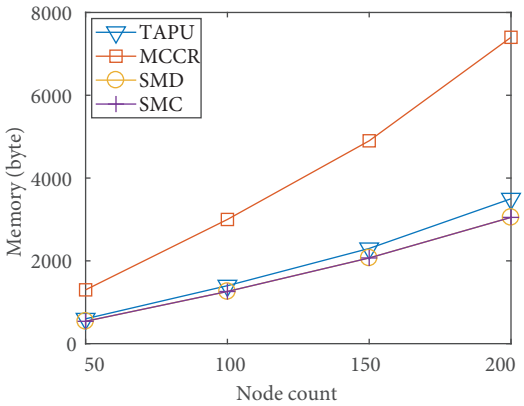


Figure 15. Memory usage against the node counts.

Figure 11 shows the wall-clock time of algorithms against the node count. These results show that TAPU is faster than the greedy algorithms and significantly better than MCCR in terms of wall-clock time. The reason for this improvement is that the proposed algorithm stops searching upon finding the nearest safe node. This causes TAPU to finish the restoration after a few calls to the testing algorithm. In the best case, if a safe node crashes, the proposed algorithm finishes after one call to the testing algorithm. Also, if the failed unsafe node has

a safe neighbor, the proposed algorithm only calls the testing algorithm twice. Most of the time, TAPU finishes the restoration process after a few calls to the k-connectivity testing algorithm. On the other hand, the MCCR algorithm calls the k-connectivity testing and matching algorithms n times after each failure. Consequently, when the network has many nodes, the wall-clock time of MCCR grows faster than that of TAPU. The greedy algorithms stop the restoration process upon finding a possible movement that leads to a k-connected topology, but since the greedy algorithms usually move more nodes than TAPU, they call the testing and node selection procedures more times, which leads to longer wall-clock time. According to the simulation results, TAPU is about 79.5% faster than MCCR and at least 33.3% faster than the greedy algorithms in the networks with 200 nodes.

Figure 12 shows the wall-clock time of algorithms against k for networks with $n = 100$ nodes. All algorithms consume more time to restore the connectivity of networks with higher k values because, generally, the number of unsafe nodes in a network with a higher k value is more than that of a network with a smaller k value. Hence, finding a safe node in a network with a higher k value takes more time. However, the proposed algorithm finishes faster than the MCCR and greedy algorithms for all k values. On average, for $k = 5$, TAPU is 66.5% faster than MCCR and at least 18.7% faster than the greedy algorithms.

Figure 13 compares the average sent bytes of algorithms against the node count. The MCCR and TAPU algorithms sent the same amount of bytes in all typologies. The sent bytes of greedy algorithms are higher than TAPU and MCCR because they move more nodes after each failure. In the networks with 200 nodes, TAPU and MCCR sent less than 800 kB while the greedy algorithms sent more than 85 kB. Figure 14 compares the average sent bytes of algorithms against the k values. For all k values, TAPU and MCCR sent the same amount of bytes while the greedy algorithms sent more bytes than them. For $k = 5$, the average sent bytes of TAPU and MCCR are about 800 kB, while this value for the greedy algorithms is higher than 900 kB.

Figure 15 compares the memory consumption (in terms of bytes) of the algorithms. The memory consumption of all algorithms follows a linear function; however, the memory consumption of TAPU is about half that of the MCCR algorithm in all topologies. In networks with 200 nodes, TAPU consumes less than 3900 bytes while MCCR consumes more than 7700 bytes. The space consumed by the greedy algorithms is slightly lower than that of TAPU, but the maximum difference is less than 450 bytes.

These results show that our theoretical findings generally confirm the experimental analysis that TAPU outperforms MCCR in terms of time and space consumption and outperforms the greedy algorithms in terms of time and generated moving cost.

7. Conclusions

Connectivity restoration is a well-known problem and a variety of algorithms have been proposed for the solution of this problem. However, the k-connectivity restoration problem for generalized k value is still an open research problem and only one algorithm, MCCR, has been proposed for the movement-based k-connectivity restoration problem. In this paper, we propose the TAPU algorithm for this problem, which improves the time complexity of the existing MCCR algorithm from $\Theta(n^2mk + n^3log_2n)$ to $O(mnk)$ in the best case and to $O(n^2mk)$ in the worst case. The space needed by MCCR is reduced from $2m + 3n$ to $m + n$. From testbed experiments and simulation results, we show that our proposed algorithm uses about 50% lower memory and runs up to 79.5% faster than MCCR and generates movements with up to 59% lower cost than the greedy algorithms. These results show that TAPU is a significant contribution to the resource-efficient movement-assisted k-connectivity detection in WSNs.

Acknowledgment

The authors would like to thank TÜBİTAK (the Scientific and Technical Research Council of Turkey) for financial support of this work with project 113E470.

References

[1] Abbasi AA, Younis M, Akkaya K. Movement-assisted connectivity restoration in wireless sensor and actor networks. IEEE Transactions on Parallel and Distributed Systems 2009; 20: 1366–1379.

[2] Akram VK, Dagdeviren O. Deck: A distributed, asynchronous and exact k-connectivity detection algorithm for wireless sensor networks. Computer Communications 2018; 116: 9–20.

[3] Al Turjman FM, Hassanein HS. Towards augmented connectivity with delay constraints in WSN federation. International Journal of Ad Hoc and Ubiquitous Computing 2012; 11: 97–108.

[4] Almasaeid HM, Kamal A. On the minimum k-connectivity repair in wireless sensor networks. In: IEEE Conference on Communications, 2009. pp. 195–199.

[5] Bai X, Xuan D, Yun Z, Lai TH, Jia W. Complete optimal deployment patterns for full-coverage and k-connectivity ($k \leq 6$) wireless sensor networks. In: 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2008. pp. 401–410.

[6] Barrera J, Cancela H, Moreno E. Topological optimization of reliable networks under dependent failures. Operations Research Letters 2015; 43: 132–136.

[7] Censor-Hillel K, Ghaffari M, Kuhn F. Distributed connectivity decomposition. In: Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing, 2014. pp. 156–165.

[8] Cornejo A, Lynch N. Fault-tolerance through k-connectivity. In: Workshop on Network Science and Systems Issues in Multi-Robot Autonomy, 2010. pp. 2–6.

[9] Dagdeviren O, Akram VK. Pack: Path coloring based k-connectivity detection algorithm for wireless sensor networks. Ad Hoc Networks 2017; 64: 41–52.

[10] Deniz F, Bagci H, Korpeoglu I, Yazici A. An adaptive, energy-aware and distributed fault-tolerant topology-control algorithm for heterogeneous wireless sensor networks. Ad Hoc Networks 2016; 44: 104–117.

[11] Even S. An algorithm for determining whether the connectivity of a graph is at least k. SIAM Journal on Computing 1975; 4: 393–396.

[12] Even S, Tarjan RE. Network flow and testing graph connectivity. SIAM Journal on Computing 1975; 4: 507–518.

[13] Fredman ML, Tarjan RE. Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM 1987; 34: 596–615.

[14] Galil Z. Finding the vertex connectivity of graphs. SIAM Journal on Computing 1980; 9: 197–199.

[15] Georgiou O, Dettmann CP, Coon JP. k-Connectivity for confined random networks. Europhysics Letters 2013; 103: 2–8.

[16] Gupta B, Gupta A. On the k-connectivity of ad-hoc wireless networks. In: 7th International Symposium on Service Oriented Systems, 2013. pp. 546–550.

[17] Henzinger MR, Rao S, Gabow HN. Computing vertex connectivity: new bounds from old techniques. Journal of Algorithms 2000; 34: 222–250.

[18] Imran M, Younis M, Said AM, Hasbullah H. Localized motion-based connectivity restoration algorithms for wireless sensor and actor networks. Journal of Network and Computer Applications 2012; 35: 844–856.

[19] Jia X, Kim D, Makki S, Wan PJ, Yi CW. Power assignment for k-connectivity in wireless ad hoc networks. Journal of Combinatorial Optimization 2005; 9: 213–222.

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

[21] Kuhn HW. The Hungarian method for the assignment problem. Naval Research Logistics Quarterly 1955; 2: 83–97.

[22] Natarajan MSG. On the probability of k-connectivity in wireless ad hoc networks under different mobility models. International Journal on Application of Graph Theory in Wireless Ad Hoc Networks and Sensor Networks 2010; 2: 1–13.

[23] Nutov Z. Approximating minimum-power k-connectivity. Ad Hoc and Sensor Wireless Networks 2010; 9: 129–137.

[24] Rahbar AG. Fault tolerant broadcasting analysis in wireless monitoring networks. Turkish Journal of Electrical Engineering & Computer Sciences 2014; 22: 1437–1452.

[25] Ranga V, Dave M, Verma AK. Network partitioning recovery mechanisms in WSANs: a survey. Wireless Personal Communications 2013; 72: 857–917.

[26] Segal M, Shpungin H. On construction of minimum energy k-fault resistant topologies. Ad Hoc Networks 2009; 7: 363–373.

[27] Senturk IF, Akkaya K, Janansefat S. Towards realistic connectivity restoration in partitioned mobile sensor networks. International Journal of Communication Systems 2016; 29: 230–250.

[28] Szczytowski P, Khelil A, Suri N. Dkm: Distributed k-connectivity maintenance in wireless sensor networks. In: 9th Annual Conference on Wireless On-demand Network Systems and Services, 2012. pp. 83–90.

[29] Wan PJ, Yi CW. Asymptotic critical transmission radius and critical neighbor number for k-connectivity in wireless ad hoc networks. In 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2004. pp. 1–8.

[30] Wang S, Mao X, Tang SJ, Li X, Zhao J, Dai G. On movement-assisted connectivity restoration in wireless sensor and actor networks. IEEE Transaction on Parallel and Distributed Systems 2011; 22: 687–694.

[31] Younis M, Akkaya K. Strategies and techniques for node placement in wireless sensor networks:a survey. Ad Hoc Networks 2008; 6: 621–655.

[32] Younis M, Senturk IF, Akkaya K, Lee S, Senel F. Topology management techniques for tolerating node failures in wireless sensor networks: a survey. Computer Networks 2014; 58: 254–283.

[33] Yun Z, Bai X, Xuan D, Lai T. H, Jia W. Optimal deployment patterns for full coverage and k-connectivity ($k \leq 6$) wireless sensor networks. IEEE/ACM Transactions on Networking 2010; 18: 934–947.

[34] Zhang H, Hou J. On the critical total power for asymptotic k-connectivity in wireless networks. In: 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 2005. pp. 466–476.

[35] Zhang L, Wang X, Dou W. Design and analysis of a k-connected topology control algorithm for ad hoc networks. In: International Symposium on Parallel and Distributed Processing and Applications, 2004. pp. 178–187.

[36] Zhao J. Minimum node degree and k-connectivity in wireless networks with unreliable links. In: International Symposium on Information Theory, 2014. pp. 246–250.

[37] Zhao J, Yagan O, Gligor V. Exact analysis of k-connectivity in secure sensor networks with unreliable links. In: 13th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2015. pp. 191–198.

[38] Zhao J, Yagan O, Gligor V. On k-connectivity and minimum vertex degree in random s-intersection graphs. In: Proceedings of the Meeting on Analytic Algorithmics and Combinatorics, 2015. pp. 1–15.