

Constant Approximation Algorithm for MST in Resource Constrained Wireless Sensor Networks

Foad Dabiri, Alireza Vahdatpour, Hyduke Noshadi, Majid Sarrafzadeh
Department of Computer Science
University of California Los Angeles
{dabiri, alireza, hyduke, majid@cs.ucla.edu}

Abstract

We consider lightweight wireless sensor networks constructed from low-profile and resource constrained wireless embedded systems. Each individual unit in these networks has limited computation, memory and power resources. Traditional models of computation and algorithms will no longer be suitable for such systems since each node in the network does not have the luxury of unbounded computations, storage and communication. In this paper a new model of computation has been introduced, in which computation of each processing unit is bounded by a value independent of the network size. Furthermore, based on this model, a fully distributed algorithm is presented to construct the minimum spanning tree with bounded degree in a network. As opposed to previous approximation results, in this algorithm the computation and message exchanging of each node is $O(d)$ where d is the degree of the node and therefore is independent of network size. Moreover, our developed algorithm yields a constant approximation ratio in which the weight of the constructed spanning tree is $O(\text{weight}(MST))$. We evaluated our algorithm through simulation and observed that the approximation ratio is about 1.2 on average.

1 Introduction

Wireless sensor network systems are gaining more interest in a variety of applications. Wireless nodes in sensor networks are usually lightweight embedded systems, which have limited computation, storage and power. In most cases they are individually battery powered hence local power consumption becomes a major factor in setting the lifetime of a node. Therefore not only distributed algorithms are needed to perform a certain functionality but also the limiting constraints should be addressed properly. In this section we first introduce the concept of bounded computation in distributed algorithms and furthermore present a new approximation algorithm for MST problem which meets the constraints imposed by low-profile modules in a network.

1.1 New Model of Computation for Lightweight Embedded Systems

Lightweight embedded systems, recently introduced due to the advancement of fabrication of powerful tiny processors, have the ability to revolutionize, capture, process and actuate in several collaborative and networked systems. The new class of tiny embedded systems has been widely utilized in several domains from medical monitoring applications to collaborative object tracking systems [1]. Many systems similar to the aforementioned applications require low-profile, mobile and cost-effective devices. The physical size and the cost effectiveness immediately deduce several constraints in processing power and communication bandwidth. In addition, it enforces restriction on the size of batteries. These unique limitations require rethinking and reinventing the design process particularly in lightweight embedded systems.

Design and development of the algorithms for these emerging platforms should meet new constraints and challenges that these types of systems introduce. One of the main challenges arises from the fact that lightweight sensor networks usually consist of a large number of tiny processing nodes. Each of the tiny nodes has a small storage capacity and low computational capabilities. Therefore running a distributed algorithm that requires unbounded number of computation in the order of the size of the network may not be suitable. Based on this argument, in most of the problems, it is not possible to adapt the traditional centralized and distributed algorithms to the lightweight embedded networks.

In this paper, we have introduced a new model of computation for distributed systems, called *constant computation*. In this model, which is based on the constraints deduced by real lightweight embedded systems, the main property of the algorithm is to construct the best possible solution for the problem while using bounded (ideally constant) amount of computation/communication and memory on each unit. Lightweight sensor networks usually form a topology af-

ter deployment or are mobile and ad-hoc and therefore do not possess a pre-defined structure. Therefore, global information about the network is either costly or impossible. Although there exist distributed algorithms in which a fair amount of global information is shared between the nodes, but in these scenarios there exist nodes in the network which the message exchanging and computations is asymptotically in the order of $f(n)$ where $f(n)$ is a non-constant function of n (like $\log(n), n^2 \dots$). As discussed earlier, lightweight processors do not have the luxury to increase their computation as the size grows. On the other hand, global information is not fixed and changes during time which requires information update.

In our model of computation, each lightweight embedded system (including the processors) is represented by a 3-tuple $C(p, b, m)$. C represents the embedded system and p, b and m stand for available power, communication bandwidth and memory. There is no direct notion of computation since theoretically, computation requires memory and processing power and therefore it is embedded in p and m . Any algorithm based on this method will utilize available resources and since each lightweight unit is defined individually, resource utilization should be independent of the whole system specification such as size. In this paper we target the very famous problem of *Minimum Spanning Tree* and every step of the proposed algorithm is based on only local information with bounded storage and computation.

1.2 Distributed Minimum Spanning Tree

To introduce the concept of *constant computation* we focus on minimum spanning tree algorithms. The study of distributed minimum spanning tree was first initiated by Gallager et al. [7] in 1983 which presents an algorithm with $O(n \log(n))$ time complexity and message optimality. Later, researcher in [2] improved the algorithm and obtained an algorithm with $O(n)$ time complexity. Furthermore, Garay et al. in [8] proposed an algorithm with the time complexity of $O(\sqrt{n \log(n)} + D)$ where D is the graph diameter. Eventually, Peleg, et al in [12] have shown that the lower bound for the time complexity of the optimal solution is $\Omega(\sqrt{n}/\log(n))$.

Furthermore, several distributed approximation algorithms have been proposed before [6], [10]. A comprehensive study has been addressed in [5] on unconditional lower bounds on the time-approximation trade-offs of the distributed MST problem. This work has improved the previous result in [12] and shown that approximating the MST problem within any constant factor requires $\Omega(\sqrt{(n/b)})$ time, where b is the number of bits that each node can send through a message to the other nodes.

However, none of the papers mentioned above considers the constraint of bounded number of computation/communication on every node. The run-time discussed

in mentioned papers is the overall execution run-time. Although there are works in the area of Euclidian MST like [11] in which computation is bounded but the constructed topology is not a tree and also each node required multi-hop information not just information of the neighbors. The previous work didn't put an upper bound on the amount of computation done by each element rather than the upper bound which is specified for the whole network. In this way, although we may have a sub linear order for the algorithm run-time, but this requirement does not enforce an upper bound less than the previous upper bound on the network.

In this paper, we present a new distributed algorithm to construct an approximated MST in a network. The main characteristic of this new algorithm is that it obeys the principle of *constant computation* that is described in the previous section and is fully distributed. Also, the cost of the constructed spanning tree lies within constant factor of the optimal MST.

2 System Model and Applications

Wireless sensor networks consist of large number of low-profile sensing and computational units distributed in space. These units usually can communicate with each other through wireless communication and have limited communication range. We consider a set of n wireless nodes as the topology studied in this paper. These nodes have a communication range equal to r or more generally, the communication range of a node in this network is represented with a function $R(u)$, u being the node. We construct the underlying graph $G = (V, E)$ as follows: The set of nodes V include the n nodes in the network and if the Euclidian distance between two nodes lies within the range of communication, there is an edge $e \in E$ in G . The distance of two nodes u and v is defined as $D(u, v)$ which is the Euclidian distance of nodes u and v . We assume that a node knows its Euclidian distance to the nodes in its communication range.

Minimum spanning tree has been a commonly occurring and important primitive in design, implementation and operation of communication networks, especially wireless sensor networks. Minimum spanning tree has been used to solve variety of problems in wireless sensor networks such as topology control, sensor coverage, routing, minimum energy broadcast, data gathering and aggregation [14] [13] [9] [3].

3 Low-weight Spanning Tree Construction

The purpose of the algorithm presented in this section is to make an approximate MST and at the same time follow the main principle of constant computation: have a bounded amount of computation in each computational element. The authors conjecture that it is not possible to find an

optimal distributed MST in a graph without having at least one node which performs $O(n)$ computation. We will show that in the proposed algorithm each node will do $O(d)$ computation/communication. Furthermore, in the next section, we will prove that the algorithm generates a spanning tree where the total weight of edges is within a constant order of the cost of minimum spanning tree. We call this graph H and will construct it step by step starting with an arbitrary node in the graph.

Intuitively, to construct an *spanning tree*, an arbitrary node start to discover the network in a similar manner to breath first search algorithm and becomes the source. We make sure that as the tree expands to capture the whole network, it remains to be a directed tree rooted at the arbitrary source. Each node u in the graph will perform several phases of operation to contribute in the approximate MST. These phases can be labeled as I)Node discovery and II) Optimization - Parent Changing. where *optimization* phase includes a *parent changing* step as well. We assign a state variable $s(u)$ for each node in the graph, which represents actions that u can perform. The valid states each node can take are:

- null: Initial state of all nodes. A node v where $s(v) = null$, has not been discovered or added to the spanning tree
- pending: A recently discovered node in the algorithm process before any optimization phase.
- discovered - opt: A recently discovered node which is performing the *optimization* phase.
- discovered: A node which has finalized its parent and is ready to start a *node discovery* phase

$s(u)$ is set to *null* for all nodes in the graph at the beginning of the algorithm. The state of an arbitrary node is set to *discovered* and this node will be the root of the tree and start the algorithm. During *node discovery* phase a node u starts to expand H by discovering adjacent nodes, which have not been included in H so far. A node u , carries a discovery level (discovery time) $l(u) = t$. When during discovery phase u adds a new node v to H , the discovery level of v is set to $t + 1$. This variable can be interpreted as the depth of each node in the tree which can potentially grow large and become in the order of n . Since storing a number which is $O(n)$ requires $O(\log(n))$ bits and in this model we have bounded storage, having such a discovery level variable immediately contradicts the bounded size computation/storage constraint. We will see that nodes do not need to know their exact discovery level. They only need their relative levels to each other. Therefore, we assign $t \bmod 4$ as the discovery level, which not only limits the discovery level to four numbers but also relative levels can be easily determined.

In the *node discovery phase* of the algorithm, a node with $s(u) = discovered$ with $l(u) = t$, will start to discover its new children by sending out *Be-My-Child* message. A nodes v with $s(v) = null$ will respond to this message by sending back a parent request message. These new children, who form the level $t + 1$ (at the time of discovery), might get discovery messages from more than one node. In replying to them, we assume that each node will select only a node which has the shortest distance to it as the parent and sets $s(v) = pending$. Therefore, it is obvious that in this state, each node from the level of $t + 1$ has a parent from level t , and this parent is the nearest among all level t nodes. (Figure 1). Next each node u with $s(u) = discovered$ starts to ask its children to optimize their parent through broadcasting a *optimize-tree* message.

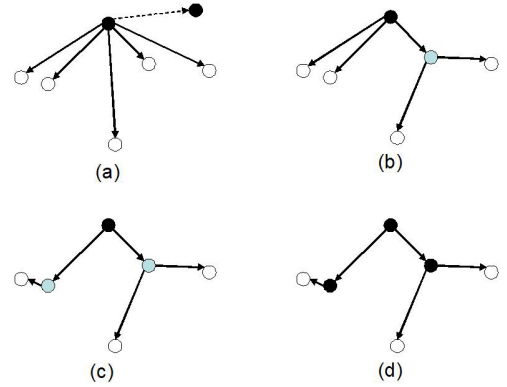


Figure 1. An example of how discovery and parent changing procedures work

In the optimization phase taken place by u , a node v from its children is called and $s(v)$ is set to *discovered - opt*. v broadcasts a *change-parent* message and examines the nodes to be its new children. The same level nodes which have *pending* or state will respond to this message only if they have a shorter distance to the caller node v that their original parent. After each parent switching, all nodes update their children list and the parent. Then u will iterate this operation for all of its children (in the updated list) who are still its children after each stage of parent changing. Figure 1 shows these steps: (a) The black nodes discovers its children which are colored white (null). It also discovers another black node which rejects the connection to it since a black node represent a previously discovered node. (b) The parent picks a child, which is represented in light color, and asks for *optimization* phase. The light-colored node broadcasts *change-parent* message and other nodes change their parents if beneficial. (c) Same procedure is repeated for the rest of the black node's children. (d) Finally, the root sets its children state to *discovered* and same algorithm starts on them independently.

Algorithm 1 The procedure activates when a node, u , gets *Discover* message

- 1: Discover nodes in the range : Broadcast a *Be – My – Child* message
 - 2: Accept all parent requests
 - 3: Set children state to *pending*
 - 4: **for** every child v **do**
 - 5: Set $s(v) = discovered - opt$
 - 6: Send *Optimize – Tree* message
 - 7: Wait for a "Done" reply
 - 8: Update children list
 - 9: **end for**
 - 10: Change the state of children to *discovered*
 - 11: Broadcast the *Discover* message to all children
-

Algorithm 2 The procedure activates when a node, u , gets *Be-My-Child* message

- 1: **if** $s(u) = null$ **then**
 - 2: Set the parent to the sender v , if multiple messages have been received, pick the closest
 - 3: Send parent request to v
 - 4: Set $s(u) = pending$
 - 5: **end if**
-

After calling the last connected child to perform the parent changing phase, each node in level $t + 1$ with state *discovered* will start phase 1 independently and therefore the algorithm will continue to expand the tree till all of the graph gets discovered. These steps have been written in four separate pseudo-codes.

Each of the two phases of the algorithm guarantees that no loop will be made while making the tree. In the first phase, only nodes that are not connected to the tree will be discovered, therefore, no loop will be made in this phase. In the second phase, only nodes will either change their parents or change their children and there will remain a node with discovery level $t + 1$ connected to a node with discovery level t .

Algorithm 3 The procedure activates when a node, u , gets *Optimize – Tree* message from its parent

- 1: Broadcast a *Change – Parent* message
 - 2: Accept all parent requests
 - 3: Return a *Done* message to the parent
-

4 Properties of the Approximation MST Algorithm

Now that we have constructed the spanning tree $G = (V, E_H)$, we will show that the cost of this tree is in fact a constant factor away from optimal MST. The proofs

Algorithm 4 The procedure activates when a node, u , gets *Change – Parent* message

- 1: **if** $s(u) = pending$ **then**
 - 2: **if** distance from current parent v , $D(u, v) >$ distance from message sender w , $D(u, w)$ **then**
 - 3: Change the parent to w and also notify v
 - 4: **end if**
 - 5: **end if**
-

are omitted for brevity. For details of the proof, please contact corresponding author.

Lemma 4.1. *The generated graph $H(V, E_H)$ is a spanning tree.*

Lemma 4.2. *In the graph $H = (V, E_H)$ generated by the algorithm in the previous section, no two edges cross if edges are straight lines drawn between two nodes.*

Lemma 4.3. *Let $(uv), (uw) \in E_H$ be two edges sharing an endpoint in the graph. Let \widehat{vuw} be the smaller angle created by these two edges. Then $\widehat{vuw} \geq 60^\circ$.*

Definition The Isolation Property. Let $c > 0$ be a constant. Let E be a set of edges in k -dimensional space, and let $e \in E$ be an edge of length l . If it is possible to place a hypercylinder B of radius and height $c \cdot l$ each, such that the axis of B is a subedge of e and $B \cap (E - e) = \phi$, then e is said to be isolated. If all the edges in E are isolated, then E is said to satisfy the isolation property.

As stated in [4] the hyper cylinder can be replaced with a cube or sphere. In our case, we replace the cylinder with a circle (two dimensional sphere) with the radius proportional to the edge investigated. Details will be presented shortly. According to the following theorem stated in [4], if a set of edges in a graph satisfy isolation property, their weight (length) sum is proportional to the weight of SMT.

Theorem 4.4. *If a set of edges E in k -dimensional space satisfies the isolation property, then $Wt(G_E) = O(1) \cdot Wt(SMT)$ [4].*

SMT is the Steiner Minimum Tree for the endpoints of the edges in E . $Wt(E)$ is the total weight of the edges in E . The following corollary is an immediate result:

Corollary 4.5. *If in the graph $G = (V, E)$, a set of edges $E' \subset E$ satisfy isolation property, then $Wt(E') = O(1) \cdot Wt(MST)$.*

MST is the minimum spanning tree of the original graph G . Since $Wt(SMT(G'(V_{E'}, E'))) \leq Wt(SMT(G(V, E))) \leq Wt(MST(G(V, E)))$, the above statement is concluded directly from theorem 4.4.

Theorem 4.6. *In the generated spanning tree H , $Wt(H) = O(Wt(MST(G)))$.*

5 Simulation Results

In this section we evaluate the performance of our algorithm with extensive simulations. We generate connected random graphs with different sizes ranging from 50 to 200 nodes and with different attributes such as minimum degree and communication range. For each test case, we increase the communication range from 10 to 100 units where the size of the plane is set to be 600×600 . Aside from random generated graphs, we also studied networks which each node is guaranteed to have a given degree i .

Figure 2 shows a sample generated network with 50 nodes under connectivity condition ¹. To guarantee connectivity we select randomly generated graphs in which are connected. We ran optimal minimum spanning tree algorithm and our approximation algorithm on the graph shown in Figure 2 a. Both spanning trees are presented here for intuitive evaluation of our proposed distributed algorithm. Figure 2 b shows the approximated spanning tree where Figure 2 c illustrates the optimal spanning tree. Minimum spanning tree is not unique in a sense that if two spanning trees have minimum cost, they might not even share an edge. Still, with comparison of the two generated trees, we observed that only 18 edges are different and the difference of the the weight is small. In this example the approximation ratio was only 1.116. In other words:

$$\gamma = \frac{Weight(MST_{approx})}{Weight(MST)} = 1.116$$

where γ is the approximation ratio and MST_{approx} is the approximated spanning tree. Moreover, we exhaustively simulated our algorithm for different sizes and test cases. The first simulation set-up was as follows: We generated random graphs with $n = 50$ to $n = 200$ nodes. For each graph, we increased the communication range from 10 to 100 with step size of 10 units. Later on, for every test case, with enforced a minimum degree of i from 0 to n , where $i = 0$ is basically a random graph with no enforcement and $i = n$ is a complete random graph. For each n we averaged out the cost of the spanning tree of our algorithm to the cost of optimal minimum spanning tree with repeating each test 20 times. The results are shown in Figure 4. As seen in Figure 4 the average approximation ratio lies within

¹x and y distances in the graphs are not scaled similarly; x-axis and y-axis are scaled to fit the column width.

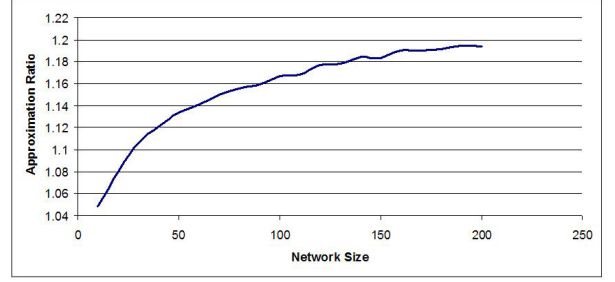


Figure 4. Approximation ratio with respect to network size averaged on different communication ranges

1.05 and 1.2 which is considered very close to optimal. The largest individual approximation ratio observed was 1.34. As shown by the same graph, approximation ratio with respect to network size is convex and converges to a constant number.

The next set of experiments was to evaluate the performance in terms of average node degree. We generated 250 random graphs for sizes $n = 50, 100, 150$ and 200. In each category, we computed the approximation ratio and the average degree. Figure 3 summarized the simulation results. The x-axis is the average degree and the y-axis is the approximation ratio. For sparse graphs where the average degree is low, we expect the MST and out spanning tree have close total weight which the graphs in Figure 3 support this claim. Also in dense graphs (where the average degree is larger), if two nodes u and v share a long edge in the graph, most probably there exist nodes in between u and v and therefore smaller edges can be selected for the minimum spanning tree. As you can see in Figure 3, when the average degree becomes large, the approximation ratio seems to stabilize around 1.2.

6 Conclusion and Future Work

In this paper we considered a network of n distributed units which communicate through wireless medium. We assume these units are lightweight embedded systems which have low-profile capabilities in terms of computation, communication, power and storage. Furthermore, we introduced the model of *constant computation* in which we impose the constraint that each node in the network has bounded capabilities (independent of the size of the network).

We presented a new algorithm for MST problem which is fully distributed and a node uses only information related to its immediate neighbors. Furthermore, the computation and message exchanging on each node is $O(d)$, where d is the degree of the node. We proved that this algorithm generates a spanning tree where its total cost is $O(Wt(MST))$ and every node in the tree has bounded degree. In other words the approximation ratio is constant. Through simulation we observed that the approximation ratio was about 1.2 on av-

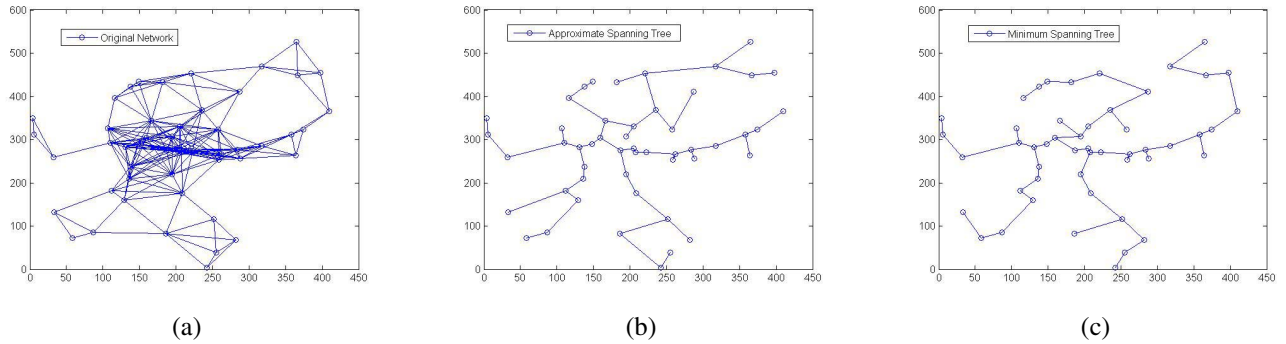


Figure 2. (a) A sample connected random network with 50 nodes. (b) The spanning tree generated by our proposed distributed algorithm. (c) Optimal minimum spanning tree.

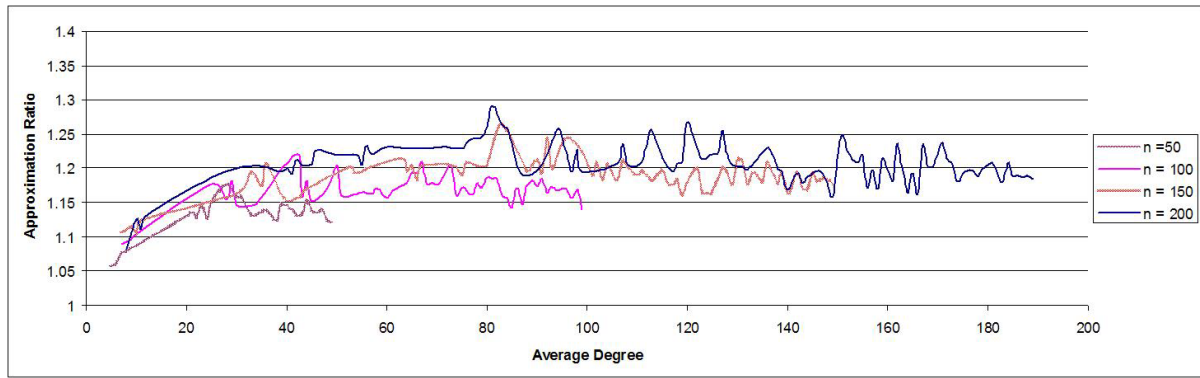


Figure 3. Approximation ratio with respect to average node degree for four different network sizes

erage. Authors conjecture that the approximation ratio is indeed 2. In future, we will study nonlinear metrics and topologies.

References

- [1] Javed Aslam, Zack Butler, Florin Constantin, Valentino Crespi, George Cybenko, and Daniela Rus. Tracking a moving object with a binary sensor network. In *SenSys '03*, pages 150–161, New York, NY, USA, 2003. ACM Press.
- [2] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *STOC '87*, pages 230–240, New York, NY, USA, 1987. ACM Press.
- [3] Xiuzhen Cheng, Bhagirath Narahari, Rahul Simha, Maggie Xiaoyan Cheng, and Dan Liu. Strong minimum energy topology in wireless sensor networks: Np-completeness and heuristics. *IEEE Transactions on Mobile Computing*, 2(3):248–256, 2003.
- [4] Gautam Das, Giri Narasimhan, and Jeffrey Salowe. A new way to weigh malnourished euclidean graphs. In *SODA '95*, pages 215–222, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.
- [5] Michael Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *STOC '04*, pages 331–340, New York, NY, USA, 2004. ACM Press.
- [6] Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. *J. Comput. Syst. Sci.*, 72(8):1282–1308, 2006.
- [7] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.
- [8] Juan A. Garay, Shay Kutten, and David Peleg. A sub-linear time distributed algorithm for minimum-weight spanning trees (extended abstract). In *STOC '93*, pages 659–668, 1993.
- [9] K. Kar and S. Banerjee. Node placement for connected coverage in sensor networks. In *Proceedings of Second Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2003.
- [10] M. Khan and G. Pandurangan. A fast distributed approximation algorithm for minimum spanning trees. *Journal of Distributed Computing*, 4167, 2006.
- [11] Xiang-Yang Li. Approximate mst for udg locally. *Lecture Notes in Computer Science*, 2697/2003:364–373, 2004.
- [12] David Peleg and Vitaly Rubinovitch. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.
- [13] Hyun sook Kim and Ki jun Han. A power efficient routing protocol based on balanced tree in wireless sensor networks. In *DFMA '05*, pages 138–143, Washington, DC, USA, 2005. IEEE Computer Society.
- [14] H. O. Tan and I. Korpoglu. Power efficient data gathering and aggregation in wireless sensor networks. *SIGMOD Rec.*, 32(4):66–71, 2003.