A Comparative Study of Partitioning Algorithms for Wireless Sensor Networks

Zeenat Rehena[1], Debasree Das[2], Sarbani Roy[2], and Nandini Mukherjee[2]

[1] School of Mobile Computing & Communication, Jadavpur University, Kolkata, India
[2] Computer Science & Engineering Department, Jadavpur University, Kolkata, India
zeenatrehena@yahoo.co.in, debasree.cse@gmail.com,
{sarbani.roy,nmukherjee}@cse.jdvu.ac.in

Abstract. In many applications large scale Wireless Sensor Networks (WSNs) use multiple sinks for fast data dissemination and energy efficiency. A WSN may be divided into a number of partitions and each partition may contain a sink, thereby reducing the distance between source nodes and sink node. This paper focuses on partitioning algorithms for WSN. Some existing graph partitioning algorithms are studied that can be applied for partitioning a WSN. A novel partitioning approach for WSN is proposed along with its modification. Simulation of the proposed algorithms has been carried out and their performances are compared with some existing algorithms. It is demonstrated that the proposed algorithms perform better than the existing algorithms.

Keywords: Partitioning Algorithms, WSN, Nearest Neighbour Graph.

1 Introduction

For early detection of critical events in large scale wireless sensor networks (WSNs), multiple sink nodes are required to be deployed. Routing the information regarding a critical event only from the source (the node that sensed the event) to its nearest sink also reduces energy consumption in an energy constrained WSN. We propose to partition the entire network into a number of sub-networks where each sub-network contains one sink node and all the sensor nodes in the sub-network forward the data sensed by them to the sink contained within it.

This paper focuses on algorithms for partitioning a network into smaller-sized sub-networks. A number of graph partitioning algorithms have been proposed in the literature. Some of them which are used for partitioning wireless sensor networks have been studied in this research work. We also propose a partitioning algorithm based on k-nearest neighbour. Another algorithm with some improvements over the former algorithm is also described. Both algorithms are implemented in a simulation environment and their performances are compared with the existing algorithms.

The rest of the paper is organized as follows. Section 2 presents the related works. The existing and the proposed partitioning algorithms are discussed in Section 3. Section 4 presents a comparative study of the existing and proposed algorithms and simulation results. Finally, we conclude the paper in Section 5.

N. Meghanathan et al. (Eds.): CCSIT 2012, Part I, LNICST 84, pp. 445–454, 2012.

2 Related Work

Conventionally, the objective of graph partitioning method is to separate the vertices of the graph into a predetermined number of sub-graphs, in which each sub-graph consists of an equal number of vertices and the cut sets among these sub-graphs are minimized. In the literature many heuristic graph partitioning algorithms have been proposed based on spectral, combinatorial, geometric and multilevel techniques.

Pothen, Simon, and Liou [1] introduce an approach to partition the input graph using the spectral information of Laplacian matrix. This technique is referred to as *recursive spectral bisection (RSB)*. Eigenvector of the Laplacian matrix is computed and using its component the graph is initially partitioned into two sets of vertices. Chan and Szeto [3] show the size of the cut sets can be minimized by using the second smallest eigenvalue of the Laplacian matrix. They have done this by introducing the concept of median cut RSB method. In this method the indices of vertices which have values above the median are mapped onto one part and which have values below the median are mapped onto the other part. The partitions are then further partitioned by recursive application of the same procedure. Another variation of RSB method is known as Modified RSB. Here instead of using median value, another statistical function quantile is used to split the graph into desired number of partitions. The authors in [7] use RSB method for partitioning a WSN into two halves and then apply this method recursively to obtain optimal number of clusters.

In [2], authors propose the approximation of the Maximally Balanced Connected Partition problem (MBCP). In [6], the authors used this MBCP to partition the entire WSN into 2^n equal sized sub-partitions where n is the number of iterations.

Both MBCP and RSB techniques finally produce 2^n equivalent smaller sub-networks where n is the number of iterations. But our objective is to partition the network into any desired number of sub-partitions according to the number of sinks available. Each sub-partition will be attached with a sink in the network, so that the nodes can interact with that associated sink only. In contrast to the above methods, we propose algorithms based on nearest neighbour computation. The main difference with the proposed algorithms with other algorithms is that here we make prior assumption regarding sink placement which is generally common in WSN.

3 Existing Partitioning Algorithms

In this section three popular graph partitioning algorithms are discussed. A novel algorithm is also proposed in this section which is based on the nearest neighbour [4] concept. Table 1 lists the notations used in different algorithms in this paper.

3.1 Recursive Spectral Bisection (RSB) [5]

RSB uses the Laplacian matrix of a graph. The construction of the Laplacian matrix is such that its smallest eigenvalue λ_1 is zero for connected graph and all the associated eigenvectors are equal to one. Except λ_1, all the other eigenvalues are greater than

zero. The RSB method that we mention here is based on the Fiedler vector of the Laplacian matrix of a given graph.

In the RSB method, the spectral information is used to partition the graph. The RSB uses *Spectral Bisection* algorithm recursively. Initially the algorithm computes Laplacian matrix *LM* of the given graph and the eigenvectors *EV* corresponding to the second largest eigenvalue of *LM*. Then it computes the median *m* of *EV*. The nodes whose eigenvectors are less than median *m* are placed in one partition and the rest are placed in the other partition. The partitions are further partitioned by recursive application of the same procedure. Above method partitions a graph into power of 2.

Table 1. List of Notations used in all the algorithms

Symbol	Description
p	total number of sink nodes, SINK={$SINK_1$, .., $SINK_n$ }
NextNode	the node used for finding its k-NNG in the next iteration
Flag	used to denote visited or unvisited node
pre_dist	previously stored distances of each node from *NextNode*
cur_dist	current distance of each node from the *NextNode* in each partition
Neighbor_list NN_s	a set of neighbour nodes of each node *s* generated from *k*-NNG, where *k* is the pre-defined number of nodes
dist	A vector storing distances of all neighbor nodes of a given node
Partition_list P	a set of sensor nodes for each partition P_p

3.2 Modified Recursive Spectral Bisection (M-RSB) [5]

Unlike RSB, the modified recursive spectral bisection algorithm partitions a graph into any number of sub-graphs. M-RSB also computes *LM* and *EV* and bisects the graph into two parts based on the value of quantile. In case of RSB, median is used to bisect the graph. In case of M-RSB, instead of median, the quantile percentage *q* of *EV* is calculated and used as the splitting value. The nodes whose eigenvector is less than *q* are placed in one partition and the rest are placed in the other partition. Each partition is then further partitioned by recursive application of the same procedure. Quantile percentage *q* of *EV* determines the number of nodes in each partition.

3.3 Maximally Balanced Connected Partition (MBCP) [6]

MBCP finds the maximally balanced connected partition for a graph *G(V, E)*. It results in a partition (V_1, V_2) of *V* composed of disjoint sets V_1 and V_2 such that both sub-graphs of *G* induced by V_1 and V_2 are connected.

The algorithm starts with two connected partitions V_1 and V_2 for *G*. Initially V_1 consists of the single vertex $v_1 \in V$ near the periphery of the network. V_2 consists $\{V - V_1\}$. In the next step, it creates a set V_0 by choosing a vertex *u* from V_2 such that $(V_1 \cup \{u\})$ and $(V_2 - \{u\})$ would also be a connected partition of *G*. From V_0, a vertex v_i is selected such that v_i is the closest element to V_1. This is done by sorting the list of candidates according to their distances from V_1. The algorithm repeats until the total number of vertices in V_1 is greater than or equal to half of the vertex in *V*.

3.4 Proposed Algorithm: FN_NNG (Farthest Node in Nearest Neighbor Graph)

The algorithm runs in two phases: Initial Phase and Incremental phase. Initially, there are *p* sink nodes and the algorithm outputs *p* partitions at the end. In the initial phase, each sink generates its k-nearest-neighbors, *k*-NNG. These are stored in its *Neighbour_list* as well as in the *Partition_list* associated with it. A Flag is used for each sensor node which is set to 'False' initially.

In the Incremental phase, the farthest neighbour node is found for each sink node. These nodes are set as *NextNode* and their neighbour nodes are found in the next step. The *k*-NNG of each *NextNode* is generated and these neighbour nodes are stored in the *Neighbour_list,* as well as in the *Partition_list.* This phase is repeated until the union of all the *Partition_lists* equals total number of nodes deployed. By setting the *Flag* of each sensor node when it is first visited, we can avoid duplication. Thus, each *Partition_list* contains disjoint set of nodes. Whenever a sensor node is included in a partition, it stores the *id* of the corresponding sink as the destination address. Algorithm for Incremental Phase is shown in the Fig. 1.

The following functions are used in FN_NNG algorithm:

FARTHEST selects the farthest neighbour node of $Root_p$, such that the returned node is not $SINK_p$ and it belongs to NN_p.
GENERATE returns the *k* nearest neighbours of a given node and place them in set *NN*.
FIND returns the $k+i^{th}$ nearest neighbour of a given node if such a node exists, else it returns null.

3.5 Improvement on FN_NNG

One major problem of using *k*-NNG is that the algorithm needs global information of sensor nodes such as location information. For this reason GPS enabled sensor nodes are required which increases the cost of the network. Furthermore, while searching *k*-Nearest Neighbour of a node in any partition, we may select a node *u* for partition P_1. Later if it is found that the node *u* is nearer to another node in partition P_2, it will not be included in partition P_2 according to the above algorithm. In such cases the region covered by partition P_1 is larger than other partitions. This scenario is depicted in Fig. 2.

Thus, some modifications on FN_NNG algorithm are suggested. Instead of using *k*-NNG, the concept of 1_Hop_Neighbour nodes is used. Here the algorithm only needs local information in the network. In order to overcome the second problem we use a variable *pre_dist* which stores the calculated distance of each node from the *NextNode.* This algorithm is also run in two phases: Initial Phase and Incremental phase. The Initial phase is similar to FN_NNG. The initial value of *pre_dist* is set to infinity. The incremental phase is shown in Fig. 3.

```
Input: NNp, Pp, n, k, S'
Output: Partition Pp at each sink node SINKp

Rootp = SINKp, i=0
while |S'| ≠ n do
for each Pp do
    NextNode=FARTHEST (Rootp)
    NNNextNode = GENERATE (NextNode)
      for each node q in NNNextNode do
        if Flagq = =true then
            i=i+1
            NNNextNode =NNNextNode - {q}
            r = FIND (NextNode, k+i)
            if r ≠ null
                NNNextNode =NNNextNode U {r}
            end if
            else Flagq = true
        end if
      end for
      Pp = Pp U NNNextNode
      S' = S' U Pp
      Rootp = NextNode
end for
end while
Return Pp
```

Fig. 1. Algorithm of Incremental Phase

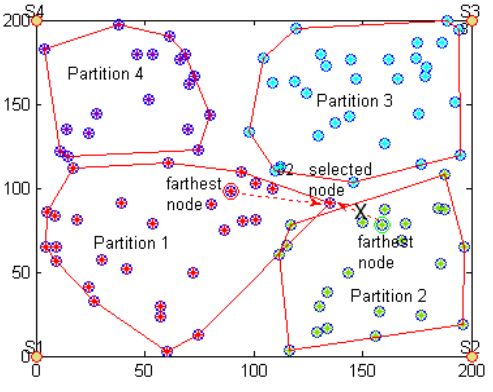

Fig. 2. Partitioning Scenario

In the 1_Hop_Neighbour function, for a particular *Neighbour_list p*, if the node *m* is not previously selected and the *cur_dist* is less than the *pre_dist* then the node is assigned into $Neighbour_p$. The *pre_dist* value is modified by *cur_dist* value. Now in the same iteration if *cur_dist* value of the node *m* in another *Neighbour_list q,* is less than *pre_dist* value, then the node *m* is deleted from previous $Neighbour_p$ and included in new $Neighbour_q$. The algorithm for selecting 1_Hop_Neighbour is given in Fig. 4.

The following additional functions are used in M-FN_NNG algorithms:

SEARCH returns the 1-Hop *Neighbour_list* which are within the communication range of a node and current distance, *cur_dist* of each neighbour node.

GET picks up an element from distance vector *dist* that corresponds to the distance of node *m.*

GETPARTITION returns the partition *id* of the partition in which the node *m* has already been included.

```
Input: NNp, Pp, n, k, S'
Output: Partition Pp at each
        sink node SINKp

Rootp = SINKp
while |S'| ≠ n do
 for each Pp do
   NextNode = FARTHEST (Rootp)
   NNNextNode = 1-Hop_Neighbor(NextNode)
   for each node q in NNNextNode do
       Flagq = true
   end for
   Pp = Pp U NNNextNode
   S' = S' U Pp
   Rootp = NextNode
  end for
end while
Return Pp
```

Fig. 3. Algorithm of Incremental Phase

```
Neighborp = = NULL
[Neighbor_list, dist] = SEARCH(Rootp)
for each node m in Neighbor_list
    cur_distm = GET(dist, m)
    if Flagm= = False
       Neighborp = Neighborp U {m}
       pre_distm = cur_distm
    else
    if (cur_distm <= pre_distm)
       q = GETPARTITION({m})
       Neighborq = Neighborq - {m}
       Neighborp = Neighborp U {m}
       pre_distm = cur_distm
    end if
    end if
end for
return Neighborp
```

Fig. 4. Algorithm of 1_Hop_Neighbour

4 Comparative Analysis and Simulation Results

Spectral Bisection methods find good partitions and are used in many applications. But the calculations of the eigenvector in spectral methods involve expensive computation.

4.1 Comparative Analysis

Using RSB method the network is partitioned into 2^n sub-networks where *n* is the number of iterations. M-RSB partitions the network into any number of partitions. Fig. 5 shows the partitioning structure of a given network using RSB method having four parts. Fig. 6 shows that six partitions are created using M-RSB.

All the methods, except M-FN_NNG, need global information of the nodes. M-FN_NNG needs only 1-Hop neighbour information. Like M-RSB, our proposed methods also partition the network into any number of sub-networks. Fig. 7 depicts four sub-partitions using FN_NNG method and Fig. 8 shows four sub-partitions using M-FN_NNG methods. Fig. 9 and Fig. 10 show six sub-partitions using FN_NNG and M-FN_NNG of a given network respectively.

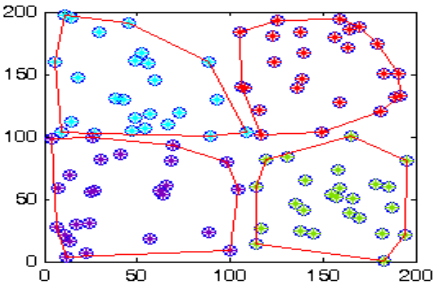

Fig. 5. 4-partitions using RSB

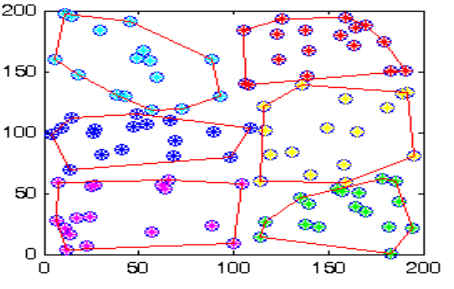

Fig. 6. 6-partitions using M-RSB

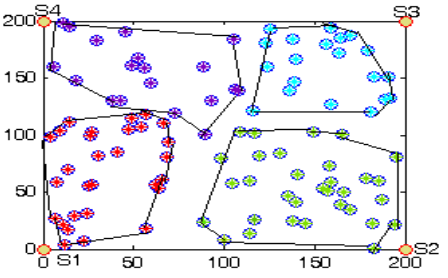

Fig. 7. 4-partitions using FN_NNG

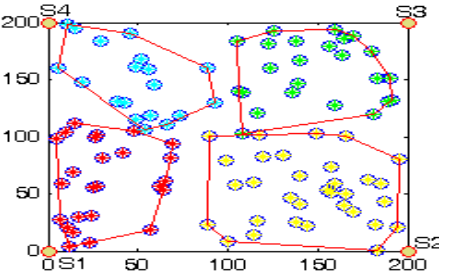

Fig. 8. 4-partitions using M-FN_NNG

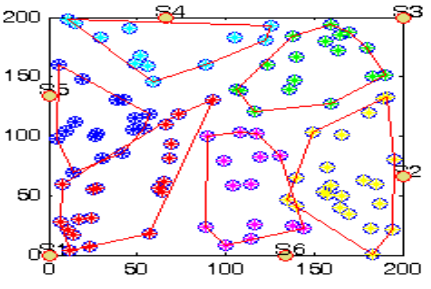

Fig. 9. 6-partitions using FN_NNG

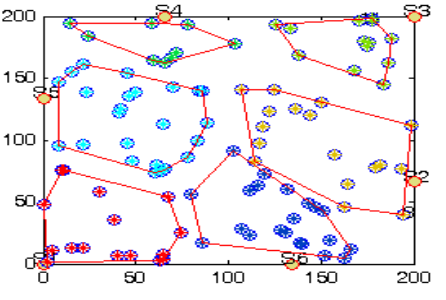

Fig. 10. 6-partitions using M-FN_NNG

The MBCP method also partitions the network into 2^n sub-networks where n is the number of iterations. The authors in [6] have not mentioned clearly how the node u is chosen from set V_0. According to them, node u of V_0 is chosen in such a way that u is closest to the elements of V_1. Thus in most cases this partitioning method generates sub-partitions which are not physically separated. This situation is shown in Fig. 11.

While implementing MBCP, we have made some modifications in the strategy of choosing the node u from set V_0. We calculate the mean of distances between u and each element of V_1. Then we choose a node u which has least mean distance in V_0.

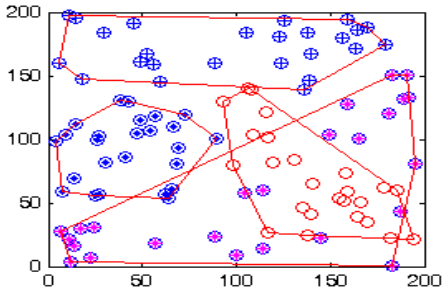

Fig. 11. 4-partitions using MBCP

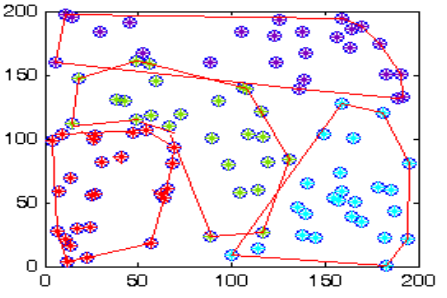

Fig. 12. 4-partitions using M-MBCP

Using the new strategy (M-MBCP), the sub-partitions generated by MBCP are shown in Fig. 12. In the next section we describe the simulation environment and present the experimental results.

4.2 Simulation Environment and Metrics

The simulations of the above algorithms have been done in Matlab environment. A wireless sensor network deployed in a square region is considered.

We use four different sensor networks ranging from 100 nodes to 400 nodes. The 100 node field is generated by randomly placing the nodes in a 200 m x 200 m square area. We assume that the area contains homogeneous sensor nodes with a communication range of 45m. Other sizes are generated by scaling the square and keeping the communication range constant in order to keep average density of sensor nodes constant.

Following metrics are evaluated for performance analysis of the algorithms:

Average Execution Time for a particular method. Execution time needed for computation of the sub-partitions in a given network is measured. We have considered four sub-partitions for each of the algorithms.

Number of Edge cuts in a particular method. Edge cut is defined as follows: $E_C = | E^{'} |$, Where E' is the set of edges with one point in V_1 and the second point in V_2. V_1 and V_2 are the set of vertices of two sub-partitions.

4.3 Result Discussion

Execution times needed for partitioning using RSB, M-RSB, MBCP, M-MBCP, FN_NNG and M-FN_NNG are compared in Fig.13. It is clear from Fig.13 that MBCP needs highest execution time and M-FN_NNG needs lowest execution time. Execution time of RSB increases due to its computation of the eigenvalues and eigenvectors. M-RSB performs slightly better than RSB. Both MBCP and M-MBCP require higher execution time, because time is spent in checking connectivity while including a vertex in each partition. Thus, FN_NNG and M-FN_NNG give much

better performance among all the methods. Fig.14 compares the number of edge cuts for the above mentioned algorithms. RSB and M-RSB return equal number of edge cuts. Fig.14 depicts that FN_NNG and M-FN_NNG also give low edge cut in comparison with other methods. Fig.15 demonstrates execution time needed to run all the methods with different sized networks. As expected, with the increased number of nodes, the execution time increases. However, in case of FN_NNG, M-FN_NNG, RSB and M-RSB methods, execution time increases slowly. But in case of MBCP and M-MBCP, the execution time increases rapidly with the increase in number of nodes.

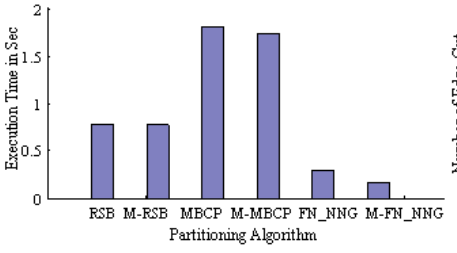

Fig. 13. Execution time

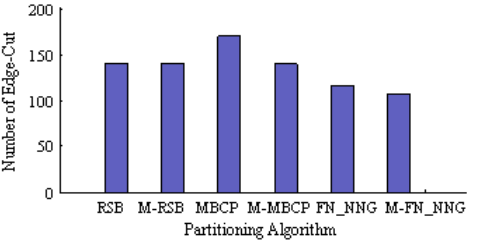

Fig. 14. Number of Edge-cut

5 Conclusion

This paper makes a comparative analysis of different graph partitioning algorithms and proposes novel algorithms which can be applied in wireless sensor networks. The existing partitioning algorithms partition the network into number of sub-partitions, whereas the proposed algorithms partition the network according to the available sinks. The simulation results demonstrate that the proposed algorithms have low execution time and low edge cut. Since WSN applications need fast response therefore the proposed algorithms are suitable for critical WSN applications, like disaster monitoring.

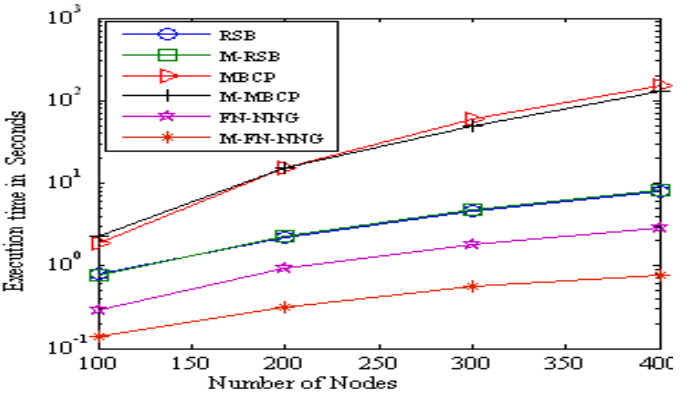

Fig. 15. Execution time with different sizes of network

References

1. Pothen, A., Simon, H.D., Liu, K.: Partitioning sparse matrices with eigenvectors of graphs. SIAM J. Matrix Anal. Appl. 11(3), 430–452 (1990)
2. Chlebikova, J.: Approximating the Maximally Balanced Connected Partition Problem in graphs. Information Processing Letters 60, 225–230 (1996)
3. Chan, T.F., Szeto, W.K.: On the Optimality of the Median Cut Spectral Bisection Graph Partitioning Method. SIAM J. Sci. Comput. 18(3), 943–948 (1997)
4. Eppstein, D., Paterson, M.S., Yao, F.F.: On Nearest-Neighbour Graphs (2000)
5. Kabelıkova, P.: Graph Partitioning Using Spectral Methods. VSB - Technical University of Ostrava (2006)
6. Slama, I., Jouaber, B., Zeghlache, D.: Energy Efficient Scheme for Large Scale Wireless Sensor Networks with Multiple Sinks. In: Wireless Communications and Networking Conference, WCNC-IEEE (2008)
7. Elbhiri, B., El Fkihi, S., Saadane, R., Aboutajdine, D.: Clustering in Wireless Sensor Networks Based on Near Optimal Bi-partitions. In: Next Generation Internet (NGI) (2010)