Obstacle Avoiding Routing in Wireless Sensor Network using Mobile Sink (Survey Paper)

Mr. Praveenkumar Prabhakar Pawar, Mr. Umashankar M L,
Student, Computer Science, VTU
Assistant professor, Computer Science, VTU

Abstract

In wireless sensor networks (WSNs), the benefits of exploiting the sink mobility to prolong network lifetime have been well recognized. In physical environments, all kinds of obstacles could exit in the sensing field. Therefore, a research challenge is how to efficiently dispatch the mobile sink to find an obstacle-avoiding shortest route. This paper presents an energy-efficient routing mechanism based on the cluster-based method for the mobile sink in WSNs with obstacles. According to the cluster-based method, the nodes selected as cluster heads collect data from their cluster members and transfer the data collected to the mobile sink. In this paper, the mobile sink starts the data-gathering route periodically from the starting site, then directly collects data from these cluster heads in a single-hop range, and finally returns to the starting site. However, due to the complexity of the scheduling problem in WSNs with obstacles, the conventional algorithms like Dijkstra difficult to resolve. To remedy this issue, we propose an efficient scheduling mechanism based on spanning graphs in this paper.

Keyword:- *wireless sesnsor network, mobile sink.*

I.Introduction

Wireless Sensor Networks(WSNs) have been applied in many respects including health monitoring, environmental monitoring, military surveillance, and many others as Internet of Thing (IoT) [1]-[5]. Energy efficiency has become the most key issue for WSNs. However, power supplies for sensor nodes are limited and hard to replace. In addition, compared with other nodes, nodes near the base station (also called the sink) consume more energy, since the nodes relay the data collected by sensor nodes far away from the sink. Hence, once these sensors near the sink fail, the data collected by other sensors cannot be transferred to the sink. Then, the entire network becomes disconnected, although most of the nodes can still have a lot of energy. Therefore, to extend the network lifetime, minimizing the energy consumption of sensor nodes is the key challenges for WSNs. Compared with static nodes, mobile nodes have more energy and more powerful capabilities. Mobile nodes, which are usually mounted on a mobile vehicle equipped with enough energy, can collect data from all static nodes by moving across the sensing field. The data from static nodes can be collected by mobile nodes in one-hop or multi-hop way. The papers [7]-[10] have proposed several different approaches. Mobile nodes are used as the mobile sink which moves across the sensing field to collect data. On the one hand, the mobile sink reduces the communication overhead for sensor nodes close to the base station or the sink, which leads to the uniform energy consumption. One the other hand, with the movement of the sink, we can better handle the disconnected and sparse network. Therefore, the network lifetime can be significantly extended by the optimum control of the route of the mobile sink. In physical environments, the sensing field could contain various obstacles. Hence, to prolong the network lifetime, a research challenge is how to find an obstacle-avoiding shortest route for the mobile sink.

The mobile sink will move through the network with obstacles to find an obstacle-avoiding shortest route. At the same time, the mobile sink must consider the energy consumption balance among nodes while moving across the sensing field. To dispatch the mobile sink efficiently, we utilize the cluster-based method that is presented in [10]. According to the cluster-based method, all sensor nodes in the sensing field are divided into two categories: cluster heads and cluster members. Cluster heads collect data from corresponding cluster members which collect environment information, and then pass data to the mobile sink. We assume that WSNs can tolerate some extent of delay so that the mobile sink collects all sensing data from cluster heads. The mobile sink begins its periodical movement from starting site and finally returns. During its movement, the mobile sink collects the sensing data from cluster heads. Once its moving path is planned, the mobile sink can move near the cluster heads and consume less energy. Hence, the network lifetime can be prolonged significantly. The network lifetime is defined as the time interval from sensor nodes start working until the death of all static sensors. However, in physical environments, the sensing field may contain various obstacles which make the scheduling for the mobile sink more complex. Here, the mobile sink can move to any site except the site of obstacles. Therefore, a research challenge is how to efficiently dispatch the mobile sink to find an obstacle-avoiding shortest route in the presence of obstacles.

To solve the scheduling for the mobile sink, we take some steps to make the dispatch problem simpler in WSNs with obstacles. Given the complexity of the problem, we present a grid-based method by which the sensing region is divided into the same size grid cells. Grid cells are considered to be the basic unit and their size is closely related to communication radius of static sensors. As the two-dimensional plane is divided into the same size grid cells, obstacles will contain some grid cells. Edges of obstacles intersect grid cells and obstacles may occupy part of some grid cells. Once obstacles occupy part of one grid cell, we assume that the grid cell is regarded as obstacles. Therefore, we obtain regularization shape of obstacles so that the scheduling for the mobile sink becomes easier. We can then construct a spanning graph in terms of the regularization shape of obstacles. With the search space of the mobile sink from all grid cells to the spanning graph obtaining grid cells, the scheduling for the mobile sink will become more efficient. Hence, we can finally find an obstacle-avoiding shortest route for the mobile sink.

II.Dijkstra Algorithm

Once the static sensor identifies the event location, mobile sensor will be dispatched to the event location. While moving mobile sensor towards the event location, it should not collide with any obstacles in the network. Sensor nodes are battery oriented, energy minimization is very important in WSN and mobile sensors have less moving energy it should reach the event location in a minimum distance. The path that allows mobile sensor to move without colliding with any obstacle is called as collision free path.

Our goal is to find the shortest collision-free path from Mobile Sensor si’s current position to event location’s position (xj, yj) which is treated as the destination or target, considering the existence of obstacles. Specifically, the movement of si should not collide with any obstacle. Several studies have addressed this issue. Work in [10] addresses how to dispatching mobile sensor to event location in the presence of obstacles. The authors used dijkstra’s algorithm to find the shortest path from mobile sensor to the event location, which is treated as target. But this method is having a drawback. The goal of the paper is to overcome this drawback. To overcome this drawback spanning graph algorithm is proposed.

The classical Dijkstra’s Algorithm is used to solve shortest path planning problem from one start point to other destination points in connected-graph, and only lengths of paths from start point to all other destination points are provided, but those middle points, which the final path from start point to each destination point passes through, is not provided, this is treated as the drawback in the classical Dijkstra’s algorithm. So, this paper uses modified Dijkstra Algorithm solve dispatch problem in the presence of obstacles. There are two aspects of Dijkstra Algorithm, which are mainly modified. One aspect, which was modified, is the finishing condition of the algorithm. In modified Dijkstra Algorithm, when algorithm obtains a path from start point to a certain point, algorithm compares the point with the destination point Vd, if it is the very Vd, algorithm can stop, otherwise, must continue. The other aspect, which was also modified, is the array, which array is named Dist, and it is used to store lengths of paths from start point to other destination points in Dijkstra Algorithm. In modified Dijkstra Algorithm, element structure of array Dist was extended as below:

```
typedef shuct DistRecorddouble
{
Value; //record length of path from VS to current point
int Prior; //record No of prior point of current point in final
path
} DIST;
```

Having that Dist array, when the destination point Vd is searched during executing algorithm, we may use the information of the Prior field of Dist to make tracks forward for the start point. Finally, all vertices that the final path from Vs to Vd passes through can be obtained, thereby, the final path can he protracted.Executing the above modified Dijkstra Algorithm, we can get a series of control points of path, and store control points into an array. In order to get smooth moving path, we can use those control points as interpolation points to do spline approximation to path. But this method is having a drawback. The goal of the paper is to overcome this drawback. To overcome this drawback spanning graph algorithm is proposed.

III.Spanning graph

In essence, the obstacle-avoiding shortest route problem is similar to the Traveling Salesman Problem (TSP) which is a classical problem. We can utilize the minimum spanning tree to solve the TSP. Hence, according to the minimum spanning tree, we can also find an obstacle-avoiding shortest route for the mobile sink. a spanning graph is an undirected graph which contains all minimum spanning trees. In this section, we will discuss how to construct the spanning graph. Several studies have addressed the spanning graph construction. We can utilize the sweep line algorithm to construct the spanning graph [10]. The obstacle-avoiding spanning graph is the set of edges that can be formed by making connections between terminals and obstacle corners. Once a spanning graph is constructed, the infinite possible sites for the mobile sink movement will be reduced to a finite set of sites. Therefore, the algorithm based on the spanning graph makes it more efficient to schedule for the mobile sink.

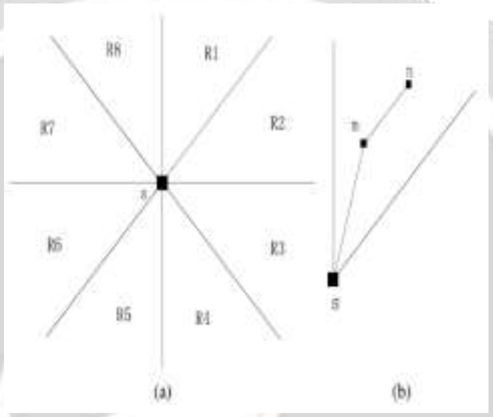


Fig.1. Eight regions of an endpoint and the uniqueness property. (a) Eight regions. (b) The uniqueness property.

We will describe how to construct the obstacle-avoiding spanning graph on the basis of eight regions of an endpoint in Figure 1(a). The plane with respect to the endpoint s is divided into 8 regions (*R*1 ~*R*8) by the two rectilinear lines and the two 45 degree lines through *s*, as shown in Figure 1(a). It has been proven that for an endpoint *s*, a region *R* has the uniqueness property with respect to s if for every pair of points *m*,*n* ϵ *R*, ||*mn*|| < max(||*sm*||; ||*sn*||). Note that the notation ||*mn*|| represents the distance between points *m* and *n*. A minimal spanning tree is acyclic, so the max(||*sm*||; ||*sn*||) in the region *R* is not included in the minimal spanning tree. For example, in Figure 1(b), the minimal spanning tree for the region *R* contains the edges (*s*,*m*) and (*m*,*n*), and the longest edge (*s*,*n*) is not contained in the minimal spanning tree. In order to simplify the calculations, we divide a plane into 4 regions in this paper. It is proved that the quadrant partition, compared with the octant partition, also exhibits good results.
Figure 2 describes the quadrant partition for an obstacle corner and a pin vertex, respectively. Algorithm 1 describes the spanning graph construction which is performed by using edge connection for different region of all pin vertices and all obstacle corners. Once the function EdgeGraph in Algorithm 2 is called in the region *R*2 and *R*6 of all obstacle corners, the corresponding edges will join the spanning graph (lines 1-4). Here, we sort elements of the list *V* by non-decreasing *x* coordinates. The line sweep sequence is reverse between the region *R*2 and *R*6, so the same sorted list V can be applied by changing the line sweep direction. Likewise, in the region *R*4 and *R*8 of all obstacle corners, we also call the function EdgeGraph in Algorithm 2.

However, unlike sorted list *V* in the Region *R*2 and *R*6, we sort elements of the list *V* by non-decreasing *y* coordinates in the region *R*4 and *R*8 of all obstacle corners (lines5-8). Besides the region *R*1 and *R*5 of all obstacle corners, the region *R*1 and *R*3 of all pin vertices is also added to our search region (lines 9-13). We sort elements of the list *V* by non-decreasing *x* C *y* coordinates in the region *R*1 and *R*5 of all obstacle corners and the region *R*1 and *R*3 of all pin vertices. To construct the spanning graph, we call the function EdgeGraph in Algorithm 2. Likewise, in the region *R*3 and *R*7 of all obstacle corners and the region *R*2 and *R*4 of all pin vertices, we also call the function EdgeGraph in Algorithm 2.

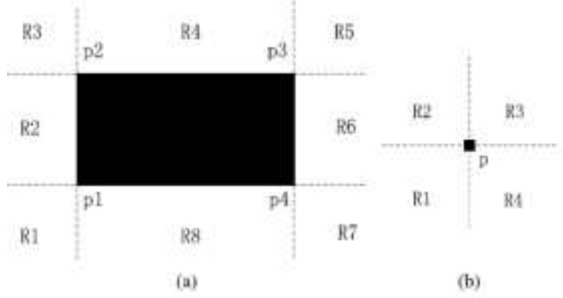
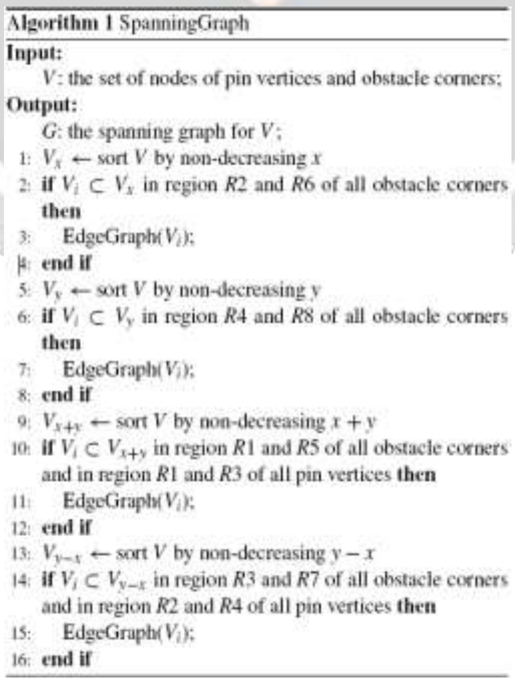
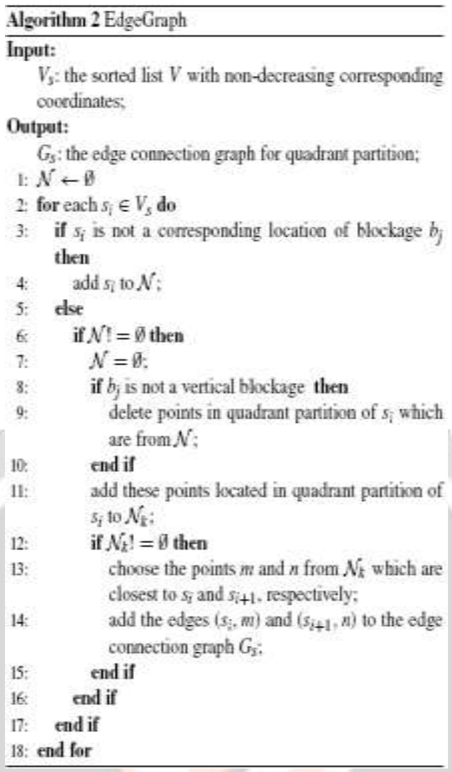


Fig.2. Search regions for blockage and pin (a) Obstacle corner (b) Pin vertex

Here, the elements of the list *V* are sorted by non-decreasing *y* + *x* coordinates (lines 14-18). According to the quadrant partition for an obstacle corner and a pin vertex, we finally construct the spanning graph. Consequently, with the search space of the mobile sink from the grid graph to the spanning graph, the possible locations for the mobile sink movement can be greatly reduced. Therefore, our algorithm makes it more efficient for the mobile sink to find an obstacle-avoiding shortest tour. The algorithm of the construction of the edge connection graph in different search regions is summarized in Algorithm 2. We sort all points in *Vs* by non-decreasing order from the start. To use the sweep line algorithm, we need an active set for one point that knows all points in the set, which is a premise for using the sweep line algorithm. Consequently, we create an active set N where all points can be dynamically added and deleted. We initialize the active set N to an empty set and decide if each point *si* is in the sorted list *Vs* (lines 1-2). Then, we will add *si* into set N if *si* is not a corresponding location of a blockage (lines 3-4). Conversely, if *si* is a corresponding location of a blockage, the edge connection for *si* will be performed (lines 5-18). Here, we assume *si* is one of the four corners of the blockage *bj*. Firstly, we choose the point *si*C1 from list *Vs*, where the point *si* locates front-left of the point *si*C1. Secondly, we decide if each point *si* is in the an active set N. Once the point *si* locates in the corresponding region of *bj*, the point *si* is added to the set N*k* . If the blockage *bj* is not a vertical blockage, we will delete points from N which are located in quadrant partition of *si*. Then, the point *m* closest to *si* and the point *n* closest to *si*C1 are selected. Finally, we get two edges (*si*;*m*) and (*si*C1; *n*) which will be added to the spanning graph *G*. Note the point *m* could coincide with the point *n* in the set N*k* . We deal with the next point in list *Vs* after the set N*k* is vacated.



```
Algorithm 1 SpanningGraph
Input:
    V: the set of nodes of pin vertices and obstacle corners;
Output:
    G: the spanning graph for V;
 1: $V_x$ ← sort V by non-decreasing x
 2: if $V_i \subset V_x$ in region R2 and R6 of all obstacle corners
    then
 3:     EdgeGraph($V_i$);
 4: end if
 5: $V_y$ ← sort V by non-decreasing y
 6: if $V_i \subset V_y$ in region R4 and R8 of all obstacle corners
    then
 7:     EdgeGraph($V_i$);
 8: end if
 9: $V_{x+y}$ ← sort V by non-decreasing $x + y$
10: if $V_i \subset V_{x+y}$ in region R1 and R5 of all obstacle corners
    and in region R1 and R3 of all pin vertices then
11:     EdgeGraph($V_i$);
12: end if
13: $V_{y-x}$ ← sort V by non-decreasing $y - x$
14: if $V_i \subset V_{y-x}$ in region R3 and R7 of all obstacle corners
    and in region R2 and R4 of all pin vertices then
15:     EdgeGraph($V_i$);
16: end if
```



Algorithm 2 EdgeGraph

```
Input:
    $V_s$: the sorted list $V$ with non-decreasing corresponding
    coordinates;
Output:
    $G_s$: the edge connection graph for quadrant partition;
1:  $\mathcal{N} \leftarrow \emptyset$
2:  for each $s_i \in V_s$ do
3:      if $s_i$ is not a corresponding location of blockage $b_j$
        then
4:          add $s_i$ to $\mathcal{N}$;
5:      else
6:          if $\mathcal{N}! = \emptyset$ then
7:              $\mathcal{N} = \emptyset$;
8:              if $b_j$ is not a vertical blockage then
9:                  delete points in quadrant partition of $s_i$ which
                    are from $\mathcal{N}$;
10:             end if
11:             add these points located in quadrant partition of
                $s_i$ to $\mathcal{N}_k$;
12:             if $\mathcal{N}_k! = \emptyset$ then
13:                 choose the points $m$ and $n$ from $\mathcal{N}_k$ which are
                    closest to $s_i$ and $s_{i+1}$, respectively;
14:                 add the edges $(s_i, m)$ and $(s_{i+1}, n)$ to the edge
                    connection graph $G_s$;
15:             end if
16:         end if
17:     end if
18: end for
```

IV.Conclusion

By Using spanning graph algorithm we can find the shortest path for the mobile sink node than using dijkstra algorithm, we utilized the mobile sink to prolong the network lifetime. In physical environments, the sensing field could contain various obstacles. To simplify the scheduling for the mobile sink, we introduced the grid-based technique to the WSN with obstacles. At the same time, we constructed the spanning graph for the mobile sink to find an obstacle-avoiding shortest route. Based on the cluster-based method, we applied the heuristic obstacle-avoiding algorithm to dispatch the mobile sink. We finally found an obstacle-avoiding shortest route for the mobile sink and the network lifetime was prolonged.

REFERENCES

[1] J. C. Cuevas-Martinez, J. Canada-Bago, J. A. Fernandez-Prieto, and M. A. Gadeo-Martos, ``Knowledge-based duty cycle estimation in wireless sensor networks: Application for sound pressure monitoring," *Appl. Soft Comput.*, vol. 13, no. 2, pp. 967_980, 2013.
[2] H.-L. Fu, H.-C. Chen, and P. Lin, ``Aps: Distributed air pollution sensing system on wireless sensor and robot networks," *Comput. Commun.*, vol. 35, no. 9, pp. 1141_1150, 2012.
[3] Z. Shen *et al.*, ``Energy consumption monitoring for sensor nodes in snap," *Int. J. Sensor Netw.*, vol. 13, no. 2, pp. 112_120, 2013.
[4] B. Zhou, S. Yang, T. H. Nguyen, T. Sun, and K. T. V. Grattan, ``Wireless sensor network platform for intrinsic optical _ber pH sensors," *IEEE Sensors J.*, vol. 14, no. 4, pp. 1313_1320, Apr. 2014.
[5] M. Dong, X. Liu, Z. Qian, A. Liu, and T. Wang, ``QoE-ensured price competition model for emerging mobile networks," *IEEE Wireless Commun.*, vol. 22, no. 4, pp. 50_57, Aug. 2015.
[6] G. Han *et al.*, ``Cross-layer optimized routing in wireless sensor networks with duty-cycle and energy harvesting," *Wireless Commun. Mobile Comput.*, vol. 15, no. 16, pp. 1957_1981, 2015.
[7] M. Zhao, Y. Yang, and C. Wang, ``Mobile data gathering with load balanced clustering and dual data uploading in wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 14, no. 4, pp. 770_785, Apr. 2015.

[8] L. Ji, Y. Yang, and W. Wang, ``Mobility assisted data gathering with solar irradiance awareness in heterogeneous energy replenishable wireless sensor networks," *Comput. Commun.*, vol. 69, pp. 88_97, Sep. 2015.
[9] G. K. Shwetha, S. Behera, and J. Mungara, ``Energy-balanced dispatch of mobile sensors in hybrid wireless sensor network with obstacles," *IOSR J.Comput. Eng.*, vol. 2, no. 1, pp. 47_51, 2012.
[10] H. A. S. Kumari and K. Shivanna, ``Dispatch of mobile sensors in the presence of obstacles using modi_ed Dijkstra algorithm,"