

On Perimeter Coverage in Wireless Sensor Networks with Minimum Cost

Ka-Shun Hung and King-Shan Lui
Department of Electrical and Electronics Engineering
The University of Hong Kong
Pokfulam Road, Hong Kong

Many sensor network applications require the tracking and the surveillance of target objects. However, in current research, many studies have assumed that a target object can be sufficiently monitored by a single sensor. This assumption is invalid in some situations, especially, when the target object is so large that a single sensor can only monitor a certain portion of it. In this case, several sensors are required to ensure a 360° coverage of the target. In this paper, we study how to identify a set of sensors that can cover the target with the minimum cost. We develop a novel distributed algorithm that requires fewer messages than existing mechanisms. Our algorithm can be extended to solve the problem when sensor range is adjustable. We provide a formal proof of correctness and convergence time of our algorithm. We further demonstrate the performance through extensive simulations.

I. INTRODUCTION

Wireless sensor networks have caught lots of attentions in recent years, because applications that are too dangerous for humans to operate could be performed by WSNs. Examples of such applications include environmental monitoring, industrial control, battlefield surveillance, home automation and security, health monitoring, and asset tracking [1], [2]. In monitoring applications, small battery-powered sensor nodes are deployed in a large scale. Each sensor node is equipped with some sensing equipment, e.g., a video camera, which can be used to sense the environment up to a certain sensing range and therefore the sensing function of each sensor can only cover a limited physical area. Due to the limited sensing range, sensor nodes usually cooperate to achieve a certain monitoring objective. The monitoring objective is usually transformed to a coverage problem. There are two common monitoring objectives suggested and widely studied [3]. They are area coverage and target coverage.

Area coverage refers to the cover of a certain target area, so that any changes within the target area can be discovered immediately and an appropriate action can then be made on time. For instance, a sudden

increase of temperature in a monitored forest area may represent a possible fire. Target coverage refers to the cover of one or more target objects within the area considered. For instance, in an art gallery, several invaluable artpieces are monitored instead of the whole gallery.

In this paper, however, we are specifically interested in a scenario in which the perimeter of a large target object is our main concern. One typical application scenario is to monitor 360° of the white house so as to ensure its security. Each sensor is associated with a cost. To reduce the total cost for monitoring, we would like to identify a set of sensors that can cover the perimeter of the large target with the minimum cost. For instance, in [4], different energy-related cost metrics are suggested to extend the lifetime of monitoring. This problem is very similar to the minimum weight circle-cover problem in a circular-arc graph in which a number of centralized algorithms have been proposed [5], [6], [7], [8]. Unfortunately, all these proposed algorithms cannot be directly applied in a wireless sensor network scenario which is distributed in nature. Previously, we [9], [10] proposed a distributed algorithm to solve this problem that requires all the nodes passing through 0° to initiate the search and thus the overhead is not minimal. In this paper, we further enhance our distributed protocol, so that we are able to find the set of sensors with minimum cost with a message overhead proportional to the size of the network. We also provide a formal proof of correctness of our proposed algorithm.

Many existing networks assume that the sensing range of a sensor is fixed. However, recent research shows that adjustable sensing range is possible. The use of smaller sensing range will generally result in smaller cost [11]. In view of this, we enhance our algorithm to consider also sensors with adjustable sensing range without increasing the message overhead.

The paper is organized as follows. In Section II, we briefly discuss the related work about the coverage problems in wireless sensor networks and the weighted circle-cover problems in circular-arc graphs. Then, the network model and our distributed algorithms under the fixed sensing range scenario will be discussed in detail in Section III together with a formal proof of correctness. In Section IV, we present the modification required to our distributed algorithm under the adjustable sensing range scenario. We carried out extensive simulations and the results are presented in Section V. Finally, we conclude our schemes in Section VI.

II. RELATED WORK

In this section, we briefly discuss the related work on the coverage problems in wireless sensor networks and the related work on the centralized algorithms proposed for the circle-cover problems in circular-arc graphs.

The two traditional coverage problems are area coverage and target coverage. Area coverage problem refers to the cover of the whole target area by the sensors. There are a number of variations, including single area coverage [12], [13], [14], [15], multiple area coverage (e.g., k-coverage [16], [17]) and fractional area coverage [18], [19], etc. On the other hand, target coverage problem refers to the cover of a certain target object or a number of target objects within a certain area. Centralized algorithms have been proposed to tackle the node placement [20] and energy-efficiency [21], [11] issues on the target coverage problem. However, centralized solution is not preferable in the sensor network environment and a distributed algorithm [22] has also been proposed to tackle the energy-efficiency issue.

In this paper, we are specifically interested in the angle/perimeter coverage problem. This problem is very similar to finding a minimum circle-cover of a circular-arc graph, which aims at finding a minimum number of arcs which can cover the whole circle. Circular-arc graph problems have been studied for quite a long time. The centralized algorithms for finding the minimum circle-cover are described in [23], [5], [24], [25]. Recently, two distributed algorithms were proposed to find the minimum circle-cover in [9], [26] and [27]. Unlike these previous works, our main focus in this paper is to find the minimum weight circle-cover of a weighted circular-arc graph.

Bertossi [5] was known to be the first to propose a parallel algorithm to tackle this problem. The time complexity of the algorithm is $O(qN^2)$ where every portion of the circle is covered by at least q arcs and there are a total of N arcs. Ibarra *et al.* [6] also proposed a similar method to solve the problem with a complexity of $O(qN \log N)$ in total. Atallah *et al.* [7] further improved the complexity to $O(qN)$ by introducing a special data structure. This is the fastest algorithm known to solve the problem.

All the algorithms discussed above find the minimum weighted circle-cover of the circular-arc graphs without any specific applications in mind. Besides, all of them are centralized with one or more processors. To the best of our knowledge, we are the first to study the perimeter coverage problem in wireless sensor networks. Previously, we developed a distributed algorithm to find the set of nodes with minimum cost that covers 360° of the target object [9], [10]. This is also known to be the first distributed algorithm

developed. Unfortunately, the message overhead of this algorithm can be very high. In this paper, we propose another approach that can reduce the message overheads and then provide a formal proof of correctness of our developed approach.

On the other hand, adjustable sensing range technique arouses more and more attentions recently. To reduce energy consumption and so as to extend network lifetime, Cardei *et al.* [11] suggested using adjustable sensing range to cover the target area. Recently, Cheng *et al.* [28] studied the use of directional sensing range to cover the target area. Each sensor can only cover a sector of the sensing range. They showed that the problem of maximizing area coverage is NP-hard and suggested a greedy algorithm to solve the problem. When there are obstacles blocking the view of directional sensors, Tezcan and Wang [29] proposed maximizing the coverage area by adjusting the orientations of the directional sensors. However, these methods cannot be applied directly to our problem and this paper is the first study in perimeter coverage with adjustable sensing range.

III. DISTRIBUTED MINIMUM COST COVER ALGORITHMS UNDER FIXED SENSING RANGE SCENARIO

A. Notations and Definitions

Before we describe our distributed protocol, we define our problem in this section.

1) *Cover Range, Cover and Cost of a Cover*: For the ease of discussion, we model the perimeter of an object as a circle and use $[0^\circ, 360^\circ)$ to denote the whole perimeter. A sensor i can cover only part of the perimeter and we denote the *cover range* as $[s_i, t_i]$. $[s_i, t_i]$ spans in the clockwise manner as illustrated in Figure 1. For a sensor i that does not cover 0° , $s_i < t_i$. $t_i < s_i$ if i covers 0° .

The cover range of two or more sensors is the union of their ranges. Although we model the perimeter as a circle, it is worth noting that our protocol works for any perimeter of arbitrary shape as long as sensors can determine their cover ranges. How a sensor determines the range is application dependent and it is outside the scope of this paper. Interested readers are referred to [30], [31], [29].

Given a set of sensors S , a subset of sensors $D \subseteq S$ is a *cover* if for each angle $\gamma \in [0^\circ, 360^\circ)$, there exists a sensor $j \in D$ such that $\gamma \in [s_j, t_j]$. In other words, $\bigcup_{i \in D} [s_i, t_i] = [0^\circ, 360^\circ)$. Figure 2 illustrates a scenario of 9 sensors surrounding a target object. Each arrow represents the cover range of a node.

Each sensor i is associated with a cost $f(i)$. In Figure 2, the number in circle represents the cost of each node. The cost of a cover D , $f(D)$, is the total cost of the sensors in the cover, that is, $f(D) = \sum_{i \in D} f(i)$. In the figure, the sets $\{1, 3, 5, 7, 8\}$, $\{1, 2, 3, 5, 6, 9\}$, and $\{1, 3, 5, 7, 9\}$ are all valid covers. The costs of

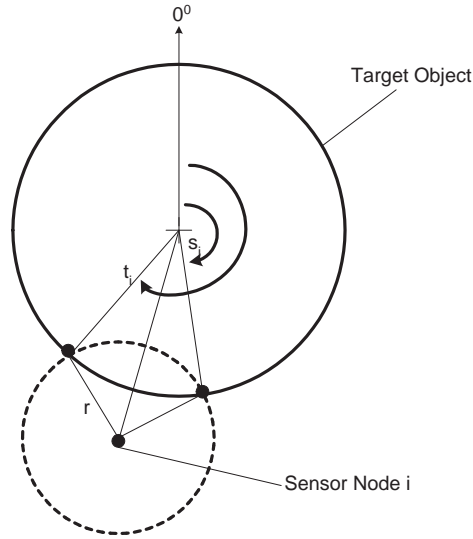


Fig. 1. A sensor node i with cover range $[s_i, t_i]$ on the perimeter of the target object.

the covers are 16, 22 and 17, respectively.

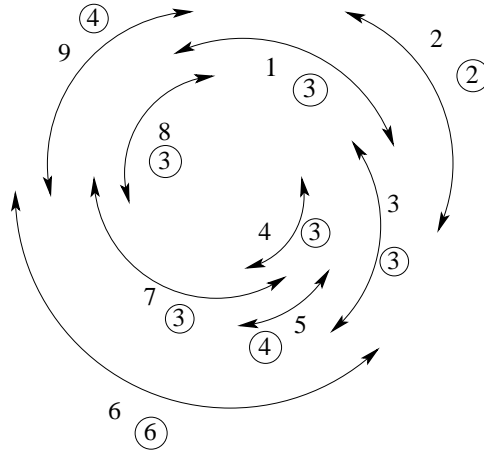


Fig. 2. An example to illustrate the concept of minimum cost cover.

2) *Minimum Cost Cover*: *Minimum Cost Cover (MCC)* is a cover with minimum cost among all covers. Formally, M is a minimum cost cover if $f(M) \leq f(D)$ for every cover $D \subseteq S$. For example, in Figure 2, $\{1, 2, 4, 7, 8\}$ is a Minimum Cost Cover, and the cost of the cover is 14. We denote $MCC(i)$ to be a cover that contains i while it is the minimum cost one among all covers that contain i . In other words, $f(MCC(i)) \leq f(D)$ for every cover D where $i \in D$.

3) *Backward and Forward Neighbors*: Two nodes are *neighbors* if their cover ranges overlap. Formally, i and j are neighbors if $s_i < s_j < t_i$ or $s_i < t_j < t_i$ ¹. With the assumption that the communication range

¹This applies when both i and j do not cover 0° . The definition can be extended easily to ranges that cover 0° but we leave it out for the ease of discussion.

is twice the sensing range, each node can communicate directly with neighbors only. It is possible that $[s_j, t_j]$ completely contains $[s_i, t_i]$ like sensors 9 and 8 in Figure 2. When two sensors have overlapping cover ranges and none of them is contained in the other, one of them is a *backward neighbor* and the other is a *forward neighbor*. i is a backward neighbor of j and j is a forward neighbor of i if $s_i < s_j < t_i$. Refer to Figure 2, sensors 2 and 3 are forward neighbors of sensor 1, while sensors 9 and 8 are backward neighbors of sensor 1. As sensor 8 is completely contained in sensor 9, they are not backward and forward neighbors of each other. It is worth noting that 8 and 9 would not appear together in the same minimum cost cover.

Theoretically, if we can find the cost of $MCC(i)$ for every $i \in S_0$, where S_0 contains the nodes with cover range passes through 0° , we know the cost of the minimum cost cover. Therefore, we first describe how to find $MCC(i)$.

B. Finding $MCC(i)$

When i wants to identify $MCC(i)$, it initiates the search by sending a search message to all its forward neighbors. This message carries a cost value that keeps the total cost to cover the range starting from s_i to the node that receives the message. When the message goes through the set of nodes in the clockwise direction, it will eventually be sent back to i and i will know which backward neighbor is in $MCC(i)$ and the cost of $MCC(i)$. This algorithm allows each node to determine whether it is in $MCC(i)$ or not but no node, even i , knows all the nodes that are in $MCC(i)$.

We now describe the mechanism in more details and provide an example. When node i wants to find $MCC(i)$, it sends the search message $\langle f(i), s_i, i \rangle$ to all its forward neighbors. When a node j receives a message $\langle c, s_i, i \rangle$ from its backward neighbor, it keeps track of the smallest c it receives so far. After j has received a message from *each* of its backward neighbors that *starts on or after* s_i , it sends $\langle c_{min} + f(j), s_i, i \rangle$ to all its forward neighbors, where c_{min} is the minimum c among all the messages it received. Note that j needs to receive the messages from backward neighbors that start after s_i only, it is because nodes that start before s_i will receive the search message later. Subsequently, each node will send out a search message and i will receive one search message from each of its backward neighbors. i can then determine which backward neighbor is in $MCC(i)$.

We use an example to illustrate the mechanism. Suppose Node 1 in Figure 3 wants to find $MCC(1)$. Note that Node 0 is ignored in this example as it is contained in Node 1. It initiates the search by

sending $\langle f(1) = 3, s_1, 1 \rangle$ to all of its forward neighbors, i.e., Nodes 2 and 3. Node 2 has only one backward neighbor that starts on or after s_1 , i.e., Node 1. Therefore, Node 2 records $f(1)$ and sends $\langle f(1) + f(2) = 7, s_1, 1 \rangle$ to its forward neighbors, which are Nodes 3 and 4. Now, Node 3 has received a message from both its backward neighbors, 1 and 2. Definitely, it finds the cost value carried by the message sent by 1 is smaller and so it sends $\langle f(1) + f(3) = 6, s_1, 1 \rangle$ to its forward neighbors 4 and 5. This time, 4 receives messages from all its backward neighbors. Let the smaller cost among the messages that a node received is c . 4 sends $\langle c + f(4) = 6 + 2 = 8, s_1, 1 \rangle$ to its forward neighbors 5 and 6. Subsequently, 5 sends out $\langle c + f(5) = 6 + 1 = 7, s_1, 1 \rangle$ and 6 sends out $\langle c + f(6) = 7 + 2 = 9, s_1, 1 \rangle$. Eventually, Node 1 would receive all messages from its backward neighbors and it can determine the cost of $MCC(1) = 7$.

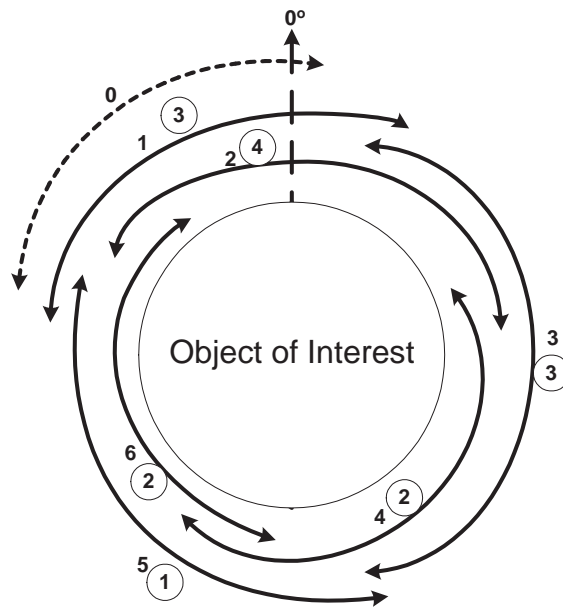


Fig. 3. An $MCC(i)$ Search Algorithm Example.

Figure 4 illustrates the state diagrams of the source node i and an intermediate node n in the $MCC(i)$ search algorithm. The pseudocode of the $MCC(i)$ search algorithm of different states can be found in the Appendix. When i wants to start the search, it enters the $INIT$ state. In this state, i sends a search message to all its forward neighbors and then transits to the $WAIT$ state. For other nodes n , when they start up, they enter the $INIT$ state. After initiating the variables in the $INIT$ state, it transits to the $WAIT$ state to wait for the search message from all its backward neighbors. Once it has received all

backward neighbors' messages, i.e., $wait_messages = 0$, it sends out the search message and enters FIN_{SH} state. During this stage, node n has completed its task in searching for $MCC(i)$ and so it waits for $\langle SELECT \rangle$ message to see if it is selected in the $MCC(i)$. Once i receives messages from all its backward neighbors, it can determine $MCC(i)$ by selecting the one with minimum cost, i.e., min_cost in the diagram. At this stage, it can enter FIN state and send back the $\langle SELECT \rangle$ message to the node $previous$ which denoted the node included in $MCC(i)$. In Figure 4, $previous$ is used to store the id of the backward neighbor in which the search message with smaller cost comes from. When node n overhears a $\langle SELECT \rangle$ message and it is the recipient of the message, it realizes that it is selected in $MCC(i)$. Then, it further sends a $\langle SELECT \rangle$ message to the node $previous$. This process continues until the node $previous$ is node i . Nodes that contain i or contained in i are not involved in the search. When they overhear the search message and realized that it is initiated by i , they can go to the sleep mode.

Since each node needs to send a search message and only those nodes selected to be in the cover need to send the $\langle SELECT \rangle$ message, the message complexity of the algorithm is $O(N)$ where N is the number of sensors in the network. We now formally proof the correctness of our mechanism.

Lemma 1 $MCC(i)$ Search Algorithm is correct.

Proof:

To ease the discussion, we assume s_i is 0° . We label the nodes according to their start angles. A node a starts before b if $s_a < s_b$. We also say b starts after a . We label i , the node that initiates the search, as n_0 and the nodes start after i as n_1, n_2 , and so on where n_x starts before n_y if $x < y$. Since n_2 can send out its search message after receiving all search messages from its backward neighbors that start after s_i instead of all backward neighbors, we refer to the backward neighbors that a node needs to wait for search messages as *important backward neighbors*. In general, the search messages that n_x must receive before it sends out a search message must be from the nodes in the set $\{n_0, n_1, \dots, n_{x-1}\}$. Besides, n_y can never be an important backward neighbor of n_x if $y > x$.

We prove this lemma by induction. n_0 initiates the search by sending $\langle f(n_0), s_{n_0}, n_0 \rangle$ to its forward neighbors. As n_0 is the only important backward neighbor of n_1 , n_1 identifies the minimum cost c and sends out the search message $\langle c, s_{n_0}, n_0 \rangle$ to its forward neighbors.

Assume that n_k can identify the minimum cost from n_0 to itself. It should be noted that n_j where $j < k$ should have sent out its message too. When n_k sends out the search message, n_{k+1} should have received a search message from each of its important backward neighbors since every node n_j where $j \leq k$ has sent a message. Thus, n_{k+1} can identify which important backward neighbor offers the minimum cost from n_0 to itself.

By induction, we prove that the minimum cost to reach n_{k+1} can also be determined correctly after n_k sends out its search message. Eventually, the search message will pass back to n_0 , and n_0 can identify which backward neighbor should be included in the cover and no more search message is generated. Therefore, when the algorithm terminates, the cover $MCC(i)$ is identified. □

C. Distributed Minimum Cost Cover Algorithm (DMCC)

One way to find the minimum cost cover is to find the cost of each $MCC(i)$ where $i \in S_0$ and identify the minimum one. The overhead would be $O(|S_0|N)$ where there are N sensors in the network. Since all nodes need to send out a search message for every individual $MCC(i)$ search, it is possible to reduce the overhead by “combining” the messages of several searches into a single message to reduce overhead. We now describe our DMCC algorithm that can identify the minimum cost cover with $O(N)$ number of messages in detail.

1) *Algorithm Description:* We call a node that covers 0° a *zero node*. That is, every node in S_0 is a zero node. The main idea of our algorithm is that every node $u \in S$ does not send out a search message until it can identify the minimum cost from each zero node i to itself. The search message sent by u should contain information about all $MCC(i)$. The major challenges of this idea are: Who should initiate the algorithm? How should u determine it has obtained all the necessary information to construct its search message? What should the search message look like? Can we prune the search of some $MCC(i)$? When will the algorithm terminate?

Combined Search Message Format: We first describe the search message. In the $MCC(i)$ search algorithm, each message is of format $\langle c, s_i, i \rangle$. The first element represents the cost, the second and the third values represent the start angle of the node that initiates the search and the identity of the node, respectively. To carry the minimum cost information of several searches in one message, we enhance the message to $\langle\langle c_{q_1}, s_{q_1}, q_1 \rangle, \dots, \langle c_{q_j}, s_{q_j}, q_j \rangle, \dots, \langle c_{q_{|S_0|}}, q_{|S_0|} \rangle\rangle$ where c_{q_j} represents the minimum

cost to reach the current node from the zero node q_j .

Message Combining Mechanism: We label the nodes in S_0 according to the ascending order of their start angles as $q_1, q_2, \dots, q_{|S_0|}$. The algorithm is best to be initiated by q_1 . It is because if q_j , where $j \neq 1$, initiates the process, the process cannot combine the search message of $MCC(q_k), \forall k = 1, \dots, j - 1$. q_1 initiates the search by sending out $\langle f(q_1), s_{q_1}, q_1 \rangle$. Based on Lemma 1, we know that q_2 knows the minimum cost to reach itself from q_1 after receiving the search message of q_1 . At this moment, it can combine $MCC(q_1)$ and $MCC(q_2)$. The combined message becomes $\langle \langle f(q_1) + f(q_2), s_{q_1}, q_1 \rangle, \langle f(q_2), s_{q_2}, q_2 \rangle \rangle$. The first tuple is used to continue the search of $MCC(q_1)$, while the second one is for searching $MCC(q_2)$. Similar process will be carried out by $q_3, \dots, q_k, \dots, q_{|S_0|}$ accordingly. In other words, each node $q_k \neq q_1$ has two tasks. First, it records the minimum costs from its backward neighbors in S_0 to itself. Second, it initiates the search of $MCC(q_k)$. After q_k has received all the search messages from its important backward neighbors, it sends the combined search message to its forward neighbors. For all other non-zero nodes, they only have to carry out the first task.

Pruning Mechanism: It is obvious that if the combined message contains the information of the search of every zero node, we can identify the MCC finally. However, some of this information, in fact, can be pruned without affecting the correctness of the mechanism. After n has received the search messages from all its backward neighbors, n compares the costs among them. Let $\langle c_{q_i}, s_{q_i}, q_i \rangle$ and $\langle c_{q_j}, s_{q_j}, q_j \rangle$ be two tuples received by n . If q_i starts earlier than q_j ($s_{q_i} \leq s_{q_j}$) and $c_{q_i} \leq c_{q_j}$, $MCC(q_j)$ would not be the only minimum cost cover and n can prune the search of $MCC(q_j)$. As q_i starts earlier, the range covered by q_j to n ($[q_j, n]$) must be contained in the range from q_i to n ($[q_i, n]$). Since $c_{q_i} \leq c_{q_j}$, if we replace the sensors in $MCC(q_j)$ that cover $[q_j, n]$ by the sensors in $MCC(q_i)$ that cover $[q_i, n]$, this new cover must have a cost not larger than $MCC(q_j)$. Therefore, n does not have to continue the search of $MCC(q_j)$. Finally, n sends the combined message which contains only the unpruned searches to all its forward neighbors.

Search Termination: Based on Lemma 1, we know that when i initiates the search of $MCC(i)$, the search can be terminated once the message comes back to i through a set of nodes in the clockwise direction. Recall that zero nodes $\{q_1, q_2, \dots, q_{|S_0|}\}$ are sorted according to their start angles. Since our algorithm is initiated by q_1 , through a set of nodes in the clockwise direction, the combined search message will eventually come back to q_1 . At this moment, the cost of $MCC(q_1)$ can be determined.

Instead of terminating the algorithm, q_1 has to continue all the remaining unpruned searches and inform other zero nodes the cost of $MCC(q_1)$. Again, two tasks are performed by a zero node q_k once it has received all the search messages. First, q_k terminates the search of $MCC(q_k)$, determines the cost of $MCC(q_k)$, and announces the cost to other zero nodes if $MCC(q_k)$ is unpruned. In fact, to determine whether $MCC(q_k)$ is pruned, q_k can check if the search messages received still contains the entry about $MCC(q_k)$. Second, it continues the search of any unpruned $MCC(q_l)$, where $l > k$ by sending the combined message containing the tuple $\langle c_{q_l} + f(q_k), s_{q_l}, q_l \rangle$. Therefore, the whole search process can be terminated when $q_{|S_0|}$ can determine the cost of $MCC(q_{|S_0|})$. Since all zero nodes can overhear each other, when $q_{|S_0|}$ announces the cost of $MCC(q_{|S_0|})$, all zero nodes should be able to determine which $MCC(q_i)$ is the smallest. q_i can then inform its previous neighbor by a $\langle SELECT \rangle$ message as described in Section III-B.

Example: To better illustrate the algorithm, we describe an example. Refer back to Figure 3, Node 1 is q_1 and Node 2 is $q_{|S_0|}$. Node 1 initiates the DMCC algorithm by sending $\langle f(1), s_1, 1 \rangle$ to its forward neighbors, in which $f(1) = 3$ in our example. After Node 1 sends out the search message, Node 2 records the minimum cost to reach Node 2 from Node 1 as $min_cost(1) = f(1) + f(2) = 3 + 4 = 7$. Since Node 2 is also a node in S_0 , it will also initiate the search of $MCC(2)$. Therefore, Node 2 sends out $\langle \langle min_cost(1) = 7, s_1, 1 \rangle, \langle f(2) = 4, s_2, 2 \rangle \rangle$ to its forward neighbors. Then, when Node 3 receives the search message from Nodes 1 and 2, it records the minimum cost to reach Node 3 from Node 1 as $min_cost(1) = f(1) + f(3) = 3 + 3 = 6$ and that of Node 2 as $min_cost(2) = f(2) + f(3) = 4 + 3 = 7$. At this moment, Node 3 receives all the search messages from its backward neighbors. In this case, Node 3 can prune the search of $MCC(2)$ as Node 2 starts later than Node 1 and with the minimum cost larger than that of Node 1. Hence, Node 3 sends out $\langle min_cost(1) = 6, s_1, 1 \rangle$ to its forward neighbors. Then, Node 4 receives two search messages. They are $\langle 4, s_2, 2 \rangle$ from Node 2 and $\langle 6, s_1, 1 \rangle$ from Node 3. It records the minimum cost to reach Node 4 from Node 1 and 2 as $min_cost(1) = 6 + f(4) = 6 + 2 = 8$ and $min_cost(2) = 4 + f(4) = 4 + 2 = 6$, respectively. Therefore, Node 4 sends out $\langle \langle min_cost(1) = 8, s_1, 1 \rangle, \langle min_cost(2) = 6, s_2, 2 \rangle \rangle$ to its forward neighbors. Afterwards, Node 5 prunes the search of $MCC(2)$ and only sends out the search message of $MCC(1)$, i.e., $\langle min_cost(1) = 7, s_1, 1 \rangle$. Node 6 sends out the combine search messages for $MCC(1)$ and $MCC(2)$, i.e., $\langle \langle min_cost(1) = 7 + 2 = 9, s_1, 1 \rangle, \langle min_cost(2) = 6 + 2 = 8, s_2, 2 \rangle \rangle$.

After receiving all the necessary search messages, Node 1 can determine the cost of $MCC(1)$ is 7 through Node 5. However, to complete the search of $MCC(2)$, Node 1 sends out $\langle min_cost(2) = 8 + 3 = 11, s_2, 2 \rangle$ to Node 2 together with the cost of $MCC(1)$. Node 2 then determines the cost of $MCC(2) = 8$ through Node 6 and it informs Node 1 the cost. At this moment, the search process is terminated and Node 1 knows that $MCC(1)$ is the minimum cost cover.

In the previous discussion, we assume the zero node that starts the earliest initiates the algorithm. However, it is possible that a zero node does not have any zero node backward neighbors even it does not start the earliest. An example is Node 0 in Figure 3. In this case, this zero node should initiate the algorithm by itself. The state diagrams and pseudocodes of DMCC can be found in the Appendix.

DMCC algorithm requires every zero node to send out two search messages, while all other non-zero nodes need to send out one. This contributes to $O(N + |S_0|)$ number of messages. Since $|S_0| \leq N$, DMCC finds MCC in $O(N)$ number of messages.

IV. ADJUSTABLE SENSING RANGE

When describing DMCC above, we assumed sensors are using a single sensing range only. We now describe how DMCC can be enhanced to find the minimum cost cover when sensors can adjust their sensing ranges to several different levels.

A. Problem Statement

Each node can adjust its sensing range to different levels. Different nodes can have different numbers of levels and different sets of sensing ranges. Recall that S denotes the set of sensors in the network. A sensor $i \in S$ can adjust its sensing range to r . This can be represented as an instance $i.r$ in the network. The set of all possible instances of sensors which are able to cover the perimeter of the target object is denoted as S' . The cover range of an instance $i.r \in S'$ is $[s_{i.r}, t_{i.r}]$ and the cost of this instance is $f(i.r)$. Under the adjustable sensing range environment, forward and backward neighbors concept defined in Section III-A.3 applies to the instances of nodes instead of the nodes themselves. This is because the nodes do not contain cover range in this scenario. Given a cover I , if $i.r \in I$, $i.r' \notin I$ for all possible sensing ranges $r' \neq r$ of node i . The cost of a cover I , $f(I)$, is the total cost of the instances in the cover, that is, $f(I) = \sum_{i.r \in I} f(i.r)$.

B. Adjustable Distributed Minimum Cost Cover algorithm (ADMCC)

One of the major differences between the adjustable sensing range scenario and the fixed sensing range scenario is that in the fixed sensing range situation, every subset in S is a valid selection. In contrast, some subsets in S' are not allowed since at most one instance of a node can be included in the cover. Fortunately, due to the properties among instances, our DMCC algorithm can be applied in S' with little modification.

1) DMCC under adjustable sensing range:

Property 1 Suppose that r and r' are two different sensing ranges of i where $i \in S$. If $r < r'$, then $i.r \in S'$ is contained inside $i.r' \in S'$.

Property 1 shows that the instance of node i with the smaller sensing range is contained inside the instance of the same node with the larger sensing range. This property is illustrated in Figure 5.

Since MCC would not contain two sensors where one is contained in the other, we have the following property.

Property 2 An MCC of S' does not contain both $i.r$ and $i.r'$ for any $i \in S$ and two different sensing ranges r and r' .

Due to Property 2, DMCC can still be directly applied to S' with a minor modification that a node $i \in S$ has to send and receive messages on behalf of its instances $i.r \in S'$.

2) *Algorithm Description of ADMCC:* Unfortunately, applying DMCC to S' directly will incur $O(|S'|)$ messages. Therefore, the overhead is increased by a factor of the number of sensing levels. To reduce overhead, a node sends out a combined message for all its instances instead of sending one message for each instance. In this case, who should send out the first message? How to combine the message? What will be the search message format?

Initiator Node: Recall that, in DMCC, the zero node that starts earliest or a zero node that does not have a zero backward neighbor should initiate the search. However, the node with the zero instance (an instance that covers 0°) in S' that starts earliest may not be the appropriate node to initiate the search if we would like to combine the message for all its instances. Consider the configuration in Figure 6. $j.r_1$ is the zero instance that starts the earliest. If j initiates the computation process, it also sends out a

search message for $MCC(j.r_2)$. It is not correct since $j.r_2$ is a forward neighbor of $i.r_1$ and violates the condition in DMCC that a node should not send out a search message until it has received the search messages from all its zero backward neighbor instances. Before we describe which node should start the search, we first define several definitions.

In the fixed range situation, we call a node *zero node* if it covers 0° . In the adjustable sensing range case, a node i is a *zero node* if there exists an instance of i that covers 0° . It is possible that not all instances of a zero node cover 0° . Nevertheless, the cover ranges of all instances of the same node share a common mid-point and, in fact, this is another property of our adjustable sensing range model as shown in Figure 5. The *mid-cover-point* of node i , denoted as $mcp(i)$, is the center of its cover ranges in different instances. For any $i, j \in S$, we define $mcp(i) < mcp(j)$ if $mcp(i)$ is on the left of $mcp(j)$ along the circle as shown in Figure 6. We have the following property:

Property 3 *For any $i, j \in S$, if $mcp(i) < mcp(j)$, then $i.x$ would not be a forward neighbor of $j.y$ for any sensing level x and y .*

In the example shown in Figure 6 where i and j are both zero nodes, we say $mcp(i) < mcp(j)$ since $mcp(i)$ is to the left of $mcp(j)$. To show this property, suppose that there is an instance $i.x$ which is a forward neighbor of an instance $j.y$. That is, $s_{j.y} < s_{i.x}$. Since $mcp(j)$ is the mid-point of the cover range, $t_{j.y} = mcp(j) + mcp(j) - s_{j.y}$. Similarly, $t_{i.x} = 2 * mcp(i) - s_{i.x}$. Since $mcp(i) < mcp(j)$ and $s_{j.y} < s_{i.x}$, we have $t_{i.x} < t_{j.y}$, which means $i.x$ is contained in $j.y$. It leads to contradiction since $j.y$ is not a forward neighbor of $i.x$ according to our definition in Section III-A.3.

Property 3 suggests that we should select a zero node with the smallest mid-cover-point to initiate the algorithm since the zero instances of this node would not be forward neighbors of other zero instances. We thus select node i to start the algorithm where node i satisfies:

- 1) i is a zero node
- 2) $mcp(i) < mcp(j)$ for any zero node j (Note that $x < y$ means x is to the left of y along the circle.)

We mentioned earlier that not all instances of a zero node are zero instances as well. Therefore, it is possible that the initiator node i contains an instance that does not cover 0° as shown in Figure 7. In this case, i should only carry the search information of those zero instances. In fact, if the non-zero instance of i is to the right of 0° , it can be pruned ($i.r_2$ in Figure 7). It is because a backward neighbor of this

non-zero instance must cover 0° , which implies that there exist another zero node with mid-cover-point smaller than $mcp(i)$. However, this is not possible since $mcp(i)$ is the smallest. Hence, i is selected to start the algorithm.

Message Combining Mechanism: After node i sends out the search message that carries the information of $MCC(i.r)$ for every zero instance $i.r$, other nodes send out a combined search message after collecting a search message for each backward neighbor of each of its instances. We now explain there will be no deadlock in the whole procedure. That is, once the search has been started by i , every node will eventually receive enough messages from its backward neighbors and create its own search message.

Lemma 2 *Every node will eventually send out a search message in the ADMCC algorithm.*

Proof:

We label the nodes as $n_0, n_1, \dots, n_{|S|}$. Let the initiator node be n_0 . There are two cases:

1) $mcp(n_0)$ is to the right of 0°

In this case, all the non-zero nodes k with $0 < mcp(k) < mcp(n_0)$ can be pruned. We arrange the nodes according to the mid-cover-points. That is, $mcp(n_x) < mcp(n_y)$ if $x < y$.

2) $mcp(n_0)$ is to the left of 0°

In this situation, nodes are divided into two sets. Set A contains the non-zero nodes k with $mcp(n_0) < mcp(k) < 0^\circ$ while $B = S \setminus A$. Note that $n_0 \in B$. We first arrange the nodes in B according to the mid-cover-point and then the nodes in A . For example, the nodes are arranged in Figure 8 according to the following order of mcp, i.e., $mcp(i), mcp(k'), mcp(j), \dots, mcp(k)$.

By defining the order in this way, Property 3 ensures that the important backward neighbors of the instances of node n_x must be instances of nodes in the set $\{n_0, n_1, \dots, n_{x-1}\}$.

We prove this lemma by induction. n_0 initiates the search by sending one combined search message for all its zero instances to its forward neighbors. If all the instances of n_0 are zero instances as shown in Figure 6, n_1 has received a search message from each of the important backward neighbors of its instances. If NOT all n_0 's instances cover 0° , the non-zero instances are either all to the left of 0° or all to the right of 0° . If they are on the right, they have been pruned as in Figure 7 and should not be considered by n_1 . If they are on the left, they are not important backward neighbors as defined in the proof of Lemma 1 and need not be considered as well. Therefore, we can conclude that n_1 has received all

the search messages from all of its important backward neighbors and it can send out its search message.

Assume that n_k has received all the search messages on behalf of all its instances. It should be noted that n_j where $j < k$ should have sent out its combined search message too. When n_k sends out the combined search message, n_{k+1} should have received a combined search message from each of its important backward neighbors and it can send out its own search message. Therefore, there is no deadlock in the algorithm. □

By Lemma 2, we know that a node i can send out one combined search message for all its instances after it has received messages from all their backward neighbors. By combining the search messages of all its instances, the number of messages can effectively be reduced to $O(N)$.

Search Termination: In ADMCC, when a zero node q_i receives all the search messages for all its instances, it can determine the cost of all unpruned $MCC(a)$, where a represents a zero instance of q_i . Instead of terminating the algorithm, it still has to continue the search of the remaining unpruned $MCC(b)$, where b is another zero instance of the zero node q_k and $mcp(q_i) < mcp(q_k)$. At the same time, q_i announces the costs of all the unpruned $MCC(a)$. The whole search process can be terminated when the zero node with the largest mid-cover-point receives search messages for all its zero instances and then announces the costs. At this moment, the costs of all unpruned searches can be determined. By overhearing others' announcements, each zero node knows which instance is selected to form the minimum cost cover.

Search Message Format: In the combined message, we need to identify the searches of different instances of different nodes. In ADMCC, the search message format is $\langle\langle e_1, \langle \text{combined search message of instance } e_1 \rangle \rangle, \langle e_2, \langle \text{combined search message of instance } e_2 \rangle \rangle, \dots \rangle$ where e_k is an instance of the node which sends out the combined search message. The format of the combined search message of instance e_k is similar to the combined search message of the node in DMCC defined in Section III-C.1.

Example: As shown in Figure 9, Node 1 has two zero Instances 1.1 and 1.2. Since Node 1 is the only zero node, it initiates the process of computing the minimum cost cover. Hence, Node 1 sends $\langle\langle 1.1, \langle f(1.1) = 5, s_{1.1}, 1.1 \rangle \rangle, \langle 1.2, \langle f(1.2) = 2, s_{1.2}, 1.2 \rangle \rangle \rangle$ to all the forward neighbors. After receiving the search message from Node 1, Node 2 updates the minimum cost to reach Instance 2.1 from Instance 1.2 as $c_{min_{1.2,2.1}} = f(1.2) + f(2.1) = 5 + 2.5 = 7.5$. It also determines the minimum

cost to reach Instance 2.2 from Instance 1.1 as $c_{min_{1.1,2.2}} = f(1.1) + f(2.2) = 2 + 6 = 8$ and that from Instance 1.2 as $c_{min_{1.2,2.2}} = f(1.2) + f(2.2) = 5 + 6 = 11$. Since Node 2 has received the search messages of all of its instances (i.e., all the backward neighbors of its instances), Node 2 sends the combined message $\langle\langle 2.1, \langle c_{min_{1.2,2.1}} = 7.5, s_{1.2}, 1.2 \rangle, \langle 2.2 \langle c_{min_{1.1,2.2}} = 8, s_{1.1}, 1.1 \rangle, \langle c_{min_{1.2,2.2}} = 11, s_{1.2}, 1.2 \rangle \rangle\rangle$ to the forward neighbors.

Upon receiving the search message from Node 2, Node 3 determines the minimum cost to reach Instance 3.1 from zero Instances 1.1 and 1.2 as $c_{min_{1.1,3.1}} = c_{min_{1.1,2.2}} + f(3.1) = 8 + 3 = 11$ and $c_{min_{1.2,3.1}} = c_{min_{1.2,2.2}} + f(3.1) = 11 + 3 = 14$, respectively. Moreover, it determines the minimum cost to reach Instance 3.2 from Instances 1.1 and 1.2 as $c_{min_{1.1,3.2}} = c_{min_{1.1,2.2}} + f(3.2) = 8 + 8 = 16$ and $c_{min_{1.2,3.2}} = c_{min_{1.2,2.2}} + f(3.2) = 11 + 8 = 19$, respectively. At this moment, Node 3 has received all the search messages to make its own. It sends $\langle\langle 3.1, \langle c_{min_{1.1,3.1}} = 11, s_{1.1}, 1.1 \rangle, \langle c_{min_{1.2,3.1}} = 14, s_{1.2}, 1.2 \rangle \rangle, \langle 3.2, \langle c_{min_{1.1,3.2}} = 16, s_{1.1}, 1.1 \rangle, \langle c_{min_{1.2,3.2}} = 19, s_{1.2}, 1.2 \rangle \rangle\rangle$ to the forward neighbors of its instances. Once Node 1 receives the combined search message from Node 3. It determines the cost of $MCC(1.2)$ is $c_{min_{1.2,3.1}} = 14$, while the cost of $MCC(1.1)$ is $c_{min_{1.1,3.2}} = 16$. Since Node 1 is the only zero node, the cost of MCC can be determined once the costs of $MCC(1.1)$ and $MCC(1.2)$ are determined. The one with the minimum cost is MCC which is $MCC(1.2)$.

The state diagrams and the pseudocodes of ADMCC can be found in the Appendix.

V. SIMULATION

For comparison, we have implemented the algorithm described in [9], [10] and our proposed algorithms (DMCC and ADMCC) in this paper. In the algorithm described in [9], [10], each zero node i finds $MCC(i)$ individually. We denote this algorithm as Exhaustive Distributed Minimum Cost Cover Algorithm (Ex-DMCC) in this literature.

A. Simulation Settings

Our simulation environment is similar to the one adopted in [9]. We considered a grid size of 100 units \times 100 units, in which every grid has a probability of 0.8 to contain a sensor. We vary the sensing range of the sensors in our simulations. The target object is located at the center (50, 50) with the object radius of 25. The cost value associated with each node is randomly generated from 1 – 2. Under the adjustable sensing range scenario, we have $f(i.r) \propto r^2$. We randomly generated 30 topologies for each set of

results. Three performance metrics are measured and they are the total number of messages generated, the expected energy expenditure due to the transmission of messages and the minimum cost of the covers found by the algorithm.

B. Simulation Results

1) *Fixed Sensing Range Scenario*: We compare the message overheads and energy needed to find the minimum cost cover using Ex-DMCC and DMCC. Figure 10 illustrates the log number of messages to find the covers with different fixed sensing ranges. The figure shows that Ex-DMCC requires a large amount of messages to find the MCC. This is because it requires each zero node q_i to look for $MCC(q_i)$ individually and $|S_0|$ can be large in some situations. On the other hand, DMCC greatly reduces the message overhead by combining the messages, so that most of the nodes only need to send out the combined search message once. On the other hand, both algorithms show an increase in the number of messages with an increase in the sensing range. This is because when nodes have a larger sensing range, more nodes would cover 0° and so $|S_0|$ is larger.

Other than message overheads, energy needed to find the minimum cost cover is also our concern. In fact, each Ex-DMCC message contains three tuples $\langle c, s_i, i \rangle$, while each DMCC message contains several combined Ex-DMCC messages. Therefore, the size of a DMCC message can be multiple times the size of an Ex-DMCC message. Practically, c and s_i can be rounded to 1 byte integers with a certain loss of precision. i depends on the size of the network and it can be represented by an integer smaller than 2 bytes in general. Therefore, the size of an Ex-DMCC message is around 4 bytes, while that of DMCC message is around $|S_0| \times 4$ bytes if none of the search is pruned. In [32], [33], the authors suggested that CSMA should be performed before sending a message so as to avoid collision. The average CSMA time is around $41.0ms$. On the other hand, we know that transmitting with the maximum range incurs a current of $21.5mA$, while listening to the channel incurs a current of $7mA$. In other words, the transmitting and the listening operations incur a power of $64.5mW$ and $21mW$ under a $3V$ power supply. The transmission rate is around $16kbps$. The energy consumed for sending a message can be estimated by $P_t \times T_x + P_l \times I_x$, where P_t and P_l are the power required for transmitting and listening, respectively. T_x is the time durations required for transmitting the whole message and can be estimated by $\frac{message\ size}{transmission\ rate}$, while I_x is the average CSMA time. From the simulation, we know that $|S_0|$ is generally less than 50 after pruning. Therefore, a combined message contains approximately 200 bytes only. Based on the information above,

the energy consumed for transmitting a 4-byte message and a 200-byte message are $1mJ$ and $7.3mJ$, respectively. Figure 11 illustrates the estimated transmission energy expenditure of both algorithms. The figure shows that DMCC uses up less energy. This is mainly due to the reason that DMCC requires much fewer messages to find the minimum cost cover.

2) *Adjustable Sensing Range Scenario*: In our simulations, we measure the performance of four situations which have different numbers of sensing levels. Particularly, 1SR, 2SR, 4SR, and 6SR represent one level, two levels, four levels, and six levels, respectively. We assume that there exists a maximum sensing range r_m . For xSR , the sensing range $r_i = \frac{i \times r_m}{x}$ is associated with the cost $f(i) = f(r_m) \times (\frac{r_i}{r_m})^2$, for $i = 1$ to x . This arrangement is based on the settings in [11], [9], [10].

Figure 12 shows that the message overhead produced by DMCC increases as the number of the sensing levels increases. In contrast, the message overhead of ADMCC, in which a node sends out a combined message for all its instances, changes only a little. Figure 13 shows that the estimated transmission energy expenditure of both DMCC and ADMCC increases as the number of the sensing levels increases. ADMCC requires fewer messages than that of DMCC and this leads to smaller amount of transmission energy expenditure. Due to this reason, the energy expenditure differences between ADMCC and DMCC increases with the numbers of sensing levels as a lot of messages are required to find the minimum cost cover in DMCC, but nearly the same amount of messages are required to find the minimum cost cover in ADMCC.

Figure 14 illustrates the costs of the minimum cost cover of various maximum sensing range settings. As expected, the more the sensing levels, the smaller the cost of covers due to the reason that finer adjustment on sensing range can be applied to each node so as to cover the target object. However, the advantage in the cover costs becomes smaller when the number of sensing levels further increases. This is because the advantage brought by the finer adjustment of the sensing range becomes smaller and smaller.

VI. CONCLUSION

In this paper, we focus on the angle/perimeter coverage problem in which a large target object is located in the center and multiple sensors are expected to collaborate to monitor 360° of this object. This coverage problem is very different from that of some common coverage problems in which a certain area or point are to be monitored instead. To the best of our knowledge, we are the first to propose a distributed algorithm for finding the minimum cost cover with the communication complexity of $O(\text{size})$

of the network). We also give the formal proof of correctness of our algorithm. Moreover, we are the first to study this problem under the adjustable sensing range scenario.

Through extensive simulations, our proposed DMCC algorithm outperforms Ex-DMCC in terms of the communication overhead, and ADMCC can find a cover with smaller cost than the existing algorithms in the adjustable sensing range scenario.

REFERENCES

- [1] D. Estrin, D. Culler, K. Pister, and G. Sukhatme, "Connecting the physical world with pervasive networks," *Pervasive Computing*, vol. 1, pp. 59–69, Jan–Mar 2002.
- [2] D. Chen and P. K. Varshney, "QoS Support in wireless sensor networks: A survey," in *Proc. of the 2004 International Conference on Wireless Networks (ICWN 2004)*, 2004.
- [3] M. Cardei and J. Wu, "Coverage in wireless sensor networks," *Handbook of Sensor Networks*, M. Ilyas and I. Magboub (eds.), 2004.
- [4] K.-Y. Chow, K.-S. Lui, and E. Y. Lam, "Wireless sensor networks scheduling for full angle coverage," *accepted to Multidimensional Systems and Signal Processing*, 2008.
- [5] A. A. Bertossi, "Parallel circle-cover algorithms," *Information Processing Letters*, vol. 27, pp. 133–139, 1988.
- [6] O. H. Ibarra, H. Wang, and Q. Zheng, "Minimum cover and single source shortest path problems for weighted interval graphs and circular-arc graphs," in *Proceedings of Allerton*, 1992.
- [7] M. J. Atallah, D. Z. Chen, and D. T. Lee, "An optimal algorithm for shortest paths on weighted interval and circular-arc graphs, with applications," *Algorithmica*, vol. 14, pp. 429–441, November 1995.
- [8] M. G. Andrews and D. T. Lee, "Parallel algorithms on circular-arc graphs," in *Computational Geometry – Theory and Applications*. Elsevier Science Publishers, 1995.
- [9] K.-Y. Chow, K.-S. Lui, and E. Y. Lam, "Maximizing angle coverage in visual sensor networks," in *IEEE International Conference on Communications (ICC 2007)*, June 2007, pp. 3516–3521.
- [10] —, "Achieving 360 angle coverage with minimum transmission cost in visual sensor networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*, March 2007, pp. 4112–4116.
- [11] M. Cardei, J. Wu, and M. Lu, "Improving network lifetime using sensors with adjustable sensing range," *International Journal of Sensor Networks*, vol. 1, pp. 41–49, 2006.
- [12] H. Gupta, Z. Zhou, S. R. Das, and Q. Gu, "Connected sensor cover: Self-organization of sensor networks for efficient query execution," in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc)*, 2003, pp. 198–200.
- [13] J. Carle and D. Simplot-Ryl, "Energy-efficient area monitoring for sensor networks," *Computer*, vol. 37, pp. 40–46, February 2004.
- [14] G. W. G. Cao, T. L. Porta, S. Phoha, G. Wang, and W. Zhang, "Distributed algorithms for deploying mobile sensors," *Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks*, pp. 427 – 439, 2006.
- [15] H. Gupta, S. R. Das, and Q. Gu, "Connected sensor cover: self-organization of sensor networks for efficient query execution," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, pp. 55–67, February 2006.
- [16] C.-F. Huang and Y.-C. Tseng, "The coverage problem in a wireless sensor network," in *ACM International Conference Wireless Sensor Networks and Applications (WSNA)*, 2003, pp. 115 – 121.
- [17] —, "The coverage problem in a wireless sensor network," *Mobile Networks and Applications*, vol. 10, pp. 519–528, August 2005.
- [18] M. Ye, E. Chan, G. Chen, and J. Wu, "Energy Efficient Fractional Coverage Schemes for Low Cost Wireless Sensor Networks," in *26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCS Workshops)*, 2006, pp. 79–79.
- [19] K.-S. Hung, K.-S. Lui, and Y.-K. Kwok, "A Trust-based Geographical Routing Scheme in Sensor Networks," in *IEEE Wireless Communication and Networking Conference (WCNC)*, 2007, pp. 3123 – 3127.
- [20] K. Kar and S. Banerjee, "Node placement for connected coverage in sensor networks," in *Proceedings of WiOpt: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2003, pp. 14 – 17.
- [21] M. Cardei and D.-Z. Du, "Improving wireless sensor network lifetime through power aware organization," *Wireless Networks*, vol. 11, pp. 333–340, May 2005.
- [22] M. T. Thai, Y. Li, F. Wang, and D.-Z. Du, "O(log n)-localized algorithms on the coverage problem in heterogeneous sensor networks," in *IEEE International Performance Computing and Communications Conference (IPCCC 2007)*, April 2007, pp. 85–92.
- [23] C. C. Lee and D. T. Lee, "On a circle-cover minimization problem," *Information Processing Letters*, vol. 18, pp. 109–115, February 1984.
- [24] M. Atallah and D. Z. Chen, "An optimal algorithm for the minimum circle-cover problem," *Information Processing Letters*, vol. 32, pp. 159–165, 1989.
- [25] M. S. Yu, C. L. Chen, and R. C. T. Lee, "Optimal parallel circle-cover and independent set algorithms for circular arc graphs," in *Proc. 1989 International Conference on Parallel Processing*, 1989, pp. 126–129.
- [26] K.-Y. Chow, "Angle coverage in wireless sensor networks," *MPhil Thesis, The University of Hong Kong*, October 2007.
- [27] K.-S. Hung and K.-S. Lui, "On perimeter coverage of wireless sensor networks," *Technical Report TR-2008-004, Department of Electrical and Electronic Engineering, The University of Hong Kong*, September 2008.
- [28] W. Cheng, S. Li, X. Liao, C. Shen, and H. Chen, "Maximal Coverage Scheduling in Randomly Deployed Directional Sensor Networks," in *IEEE International Conference Parallel Processing Workshops (ICPPW)*, September 2007, pp. 68–68.

- [29] N. Tezcan and W. Wang, "Self-Orienting Wireless Multimedia Sensor Networks for Maximizing Multimedia Coverage," in *IEEE International Conference on Communications (ICC)*, May 2008, pp. 2206–2210.
- [30] H. Lee and H. Aghajan, "Vision-enabled node localization in wireless sensor networks." in *COGNitive systems with Interactive Sensors (COGIS)*, March 2006.
- [31] C. McCormick, P.-Y. Lalgand, H. Lee, and H. Aghajan, "Distributed agent control with self-localizing wireless image sensor networks." in *COGNitive systems with Interactive Sensors (COGIS)*, March 2006.
- [32] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh, "Simulating the Power Consumption of Large-Scale Sensor Network Applications." in *Proceedings of the 2nd international conference on Embedded networked sensor systems (ACM Sensys)*, November 2004, pp. 188–200.
- [33] —, "PowerTOSSIM: Efficient Power Simulation for TinyOS Applications," in <http://www.eecs.harvard.edu/shnayder/ptossim>, 2008.

APPENDIX

A. Pseudocode of various distributed minimum cost cover algorithms

1) *MCC(i) Search Algorithm*: The following pseudocode illustrates the *MCC(i)* Search Algorithm.

Algorithm 1 Pseudocode of distributed *MCC(i)* Search Algorithm of source node i .

```

/* ----- INIT state ----- */
1:  $cost(i) = f(i)$ 
2:  $previous(i) = \perp$ 
3:  $total\_cost = \infty$  /*current minimum cost*/
4: Send  $\langle f(i), s_i, i \rangle$  to all forward neighbors. Then,  $i$  turns into WAIT state.
/* ----- WAIT state ----- */
1: /* Receiving Search Message  $\langle c, s_i, i \rangle$  from Backward neighbor  $w$  */
2: if  $total\_cost > c$  then
3:   /* Update the cost of MCC(i) */
4:    $total\_cost = c$ 
5:    $previous(i) = w$ 
6: end if
7: if received all backward neighbors' messages then
8:   enter the FIN state
9: end if
/* ----- FIN state ----- */
1: /*  $f(MCC(i)) = total\_cost$  */
2: send  $\langle SELECT \rangle$  to  $previous(i)$ 

```

Algorithm 2 Pseudocode of distributed *MCC(i)* Search Algorithm of the other node n .

```

/* ----- INIT state ----- */
1:  $cost(n) = \infty$ 
2:  $previous(n) = \perp$ 
3: enter the WAIT state.
/* ----- WAIT state ----- */
1: /* Receiving Search Message  $\langle c, s_i, i \rangle$  from backward neighbor  $w$  */
2: if  $cost(n) > c + f(n)$  then
3:   /* Update the minimum cost to reach  $n$  from  $i$  */
4:    $cost(n) = c + f(n)$ 
5:    $previous(n) = w$ 
6: end if
7: if all backward neighbors' messages are received then
8:   Send  $\langle cost(n), s_i, i \rangle$  to all forward neighbors.
9:   Enter FIN state.
10: end if
/* ----- FIN state ----- */
1: if  $\langle SELECT \rangle$  message is targeted for  $n$  then
2:   send  $\langle SELECT \rangle$  to  $previous(i)$ 
3: end if

```

2) *DMCC: State Diagrams*: There are three different kinds of nodes and Figure 15 illustrates their corresponding state diagrams. As shown in Figure 15(a), the zero node q_{min} with the smallest start angle initiates the process of finding the minimum cost cover. q_{min} sends the message $\langle f(q_{min}), s_{q_{min}}, q_{min} \rangle$ to the forward neighbors and transits to the *WAIT* state to wait for the combined search messages from its backward neighbors, i.e., $wait_messages = |BN(q_{min})|$. Throughout the paper, we use the notation $BN(i)$ to denote the set of backward neighbors of i .

For other zero node $q_k \in S_0$ as shown in Figure 15(b), q_k enters the *INIT* state when it starts up. Then, it transits to the $WAIT_{S_0}$ state to wait for the search message coming from important backward neighbors, i.e., $wait_messages = |important\ BN(q_k)|$. When $wait_messages = 0$, it can compute the minimum cost $min_cost(q_j)$ from each unpruned zero node q_j , and then advertises this information together with the search of $MCC(q_k)$. Then, it transits to $WAIT_{SH}$ state to wait for the search messages coming from all its backward neighbors, i.e., $wait_messages = |BN(q_k)|$.

For other non-zero node n , as shown in Figure 15(c), n enters the *INIT* state when it first starts up. Then, it transits to the *WAIT* state. When $wait_messages = 0$, it can compute the minimum cost $min_cost(q_j)$ from each unpruned zero node q_j . n can then transit to the *FIN* state.

After the zero node q_{max} with the maximum start angle determines and announces the cost of $MCC(q_{max})$, all the zero nodes can determine which one is *MCC*. Therefore, the $\langle SELECT \rangle$ message can be sent by q_i with $min(f(MCC(q_i)))$ to the node $previous(q_i)$ recorded in q_i . The same process can be done by the node which is the target of the $\langle SELECT \rangle$ message. This process terminates when a node is reached which records $previous(q_i) = q_i$. In the DMCC algorithm described above, for the ease of discussion, the procedures about the $\langle SELECT \rangle$ message circulation have been eliminated.

Pseudocode: The following pseudocodes illustrate the distributed minimum cost cover algorithm (DMCC). Algorithm 5 illustrates the action of zero node q_{min} with smallest start angle, while algorithm 6 illustrates the action of other zero node q_k . On the other hand, algorithm 7 illustrates the action of non-zero node n .

Algorithm 3 Initialization of DMCC.

```

1: current node is  $i$ .
2: for each  $q_j \in S_0$  do
3:    $min\_cost(q_j) = \infty$  /* The minimum cost known about zero node  $q_j$ .*/
4:    $previous(q_j) = \perp$  /* The backward neighbor that zero node  $q_j$  should pass through to reach  $i$ .*/
5: end for
6: if Current Node  $i$  is a zero node then
7:    $total\_cost = \infty$  /* The cost of  $MCC(i)$ .*/
8:    $min\_cost(i) = f(i)$ 
9: end if

```

3) *ADMCC*: **State Diagrams:** Similar to that of DMCC, there are three different kinds of nodes in ADMCC. A zero node q_{min} with minimum mid-cover-point should initiate the ADMCC algorithm. q_{min} enters the *INIT* state and initiates the ADMCC algorithm as shown in Figure 16(a). It is possible for q_{min} to have both zero instances and non-zero instances. An instance of q_{min} is denoted as a , while a zero

Algorithm 4 Pruning Mechanism of DMCC.

```
1: for Unpruned zero node  $q_i$  and  $min\_cost(q_i) \neq \infty$  do
2:   for Unpruned zero node  $q_j$  and  $min\_cost(q_j) \neq \infty$  do
3:     if  $q_i \neq q_j$  then
4:       if  $s_{q_i} < s_{q_j}$  and  $min\_cost(q_i) < min\_cost(q_j)$  then
5:          $min\_cost(q_j) = \infty$ 
6:          $previous(q_j) = \perp$ 
7:       end if
8:     end if
9:   end for
10: end for
```

Algorithm 5 Pseudocode of Distributed Minimum Cost Cover Algorithm (DMCC) for zero node q_{min} with the minimum start angle.

```
/* ----- INIT state ----- */
1: Performs Initialization in Algorithm 3.
2: Send  $\langle f(q_{min}), s_{q_{min}}, q_{min} \rangle$ 
3: Enter to the WAIT state.
/* ----- WAIT state ----- */
1: /* Receiving Search Message M  $\langle \langle cost(q_j), s_{q_j}, q_j \rangle \rangle$  from a backward neighbor  $w \in S$  */
2: for each tuple  $\langle cost(q_j), s_{q_j}, q_j \rangle$  in M do
3:   if  $q_j = q_{min}$  then
4:     /*The Message is for finding  $MCC(q_{min})$ */
5:     Update the cost of  $MCC(q_{min})$ .
6:   else
7:     /*The Message is for finding unpruned  $MCC(q_j)$ , where  $q_j$  is a zero node and  $s_{q_j} > s_{q_{min}}$  */
8:     Update the minimum cost to reach  $q_{min}$  from zero node  $q_j$  in clockwise direction.
9:   end if
10: end for
11: if Received search messages from all backward neighbors  $w \in S$  then
12:   Performs pruning mechanism in Algorithm 4.
13:   Send  $\langle \langle min\_cost(q_j), q_j \rangle \rangle, \forall min\_cost(q_j) \neq \infty$  and  $s_{q_j} \geq s_{q_{min}}$  to all forward neighbors
14:   enter to the FIN state.
15: end if
```

Algorithm 6 Pseudocode of Distributed Minimum Cost Cover Algorithm (DMCC) for other zero node q_k .

```
/* ----- INIT state ----- */
1: Performs Initialization in Algorithm 3.
2: Enter the WAIT1 state.
/* ----- WAIT1 state ----- */
1: /* Receiving Search Message M  $\langle \langle cost(q_j), s_{q_j}, q_j \rangle \rangle$  from a backward neighbor  $w \in S_0$  */
2: for each tuple  $\langle cost(q_j), s_{q_j}, q_j \rangle$  in M do
3:   Update the minimum cost to reach  $q_k$  from zero node  $q_j$ .
4: end for
5: if Received messages from all backward neighbors  $w \in S_0$  then
6:   Performs Pruning Mechanism in Algorithm 4.
7:   /* The tuple  $\langle f(q_k), s_{q_k}, q_k \rangle$  has also been included in the combined message to initiate the search of  $MCC(q_k)$  as  $min\_cost(q_j) \neq \infty$ .*/
8:   Send  $\langle \langle min\_cost(q_j), s_{q_j}, q_j \rangle \rangle, \forall min\_cost(q_j) \neq \infty$  to all forward neighbors.
9:   Enter the WAIT2 state.
10: end if
/* ----- WAIT2 state ----- */
/* Same as the WAIT state of Algorithm 5. */
```

Algorithm 7 Pseudocode of Distributed Minimum Cost Cover Algorithm (DMCC) for non-zero node n

```
/* ----- INIT state ----- */
1: Performs Initialization in Algorithm 3.
2: Enter the WAIT state.
/* ----- WAIT state ----- */
1: /* Receiving Search Message  $\langle \langle cost(q_j), s_{q_j}, q_j \rangle \rangle$  from backward neighbor  $w \in S$  */
2: for each tuple  $\langle cost(q_j), s_{q_j}, q_j \rangle$  in M do
3:   Update the minimum cost to reach  $n$  from zero node  $q_j$ .
4: end for
5: if Received Search Message from all backward neighbors  $w$  then
6:   Perform Pruning Mechanism in Algorithm 4.
7:   Send  $\langle \langle min\_cost(q_j), s_{q_j}, q_j \rangle \rangle, \forall min\_cost(q_j) \neq \infty$  to all forward neighbors.
8: end if
```

instance of q_{min} is denoted as a_0 . Therefore, Node q_{min} fills in the search message $\langle \langle a_0, \langle f(a_0), s_{a_0}, a_0 \rangle \rangle \rangle$ for each zero instance a_0 and forwards the message to their forward neighbors in one combined message and then enters the *WAIT* state.

Other than q_{min} , zero node(s) q_k can have both zero and non-zero instances. Again, an instance of q_k is denoted as c and a zero instance is denoted as c_0 . q_k enters the *INIT* state when it starts up and then enters the *WAIT1* state as shown in Figure 16(b). In this state, q_k needs to wait for all zero nodes q_j with mid-cover-point smaller than q_k to send out the search message. Other than that, it also needs to wait for non-zero nodes with mid-cover-point smaller than q_k if $mcp(q_k)$ is along the right of 0° . After q_k receives all the necessary search messages, it carries out the search of $MCC(c_0)$ for each c_0 by filling in the fields $\langle c_0, \langle f(c_0), s_{c_0}, c_0 \rangle \rangle$ in the combined search message and forwards the combined message to the forward neighbors. q_k then enters the *WAIT2* state.

Non-zero node n will enter the *INIT* state when it starts up. Then, it enters the *WAIT* state as shown in Figure 16(c). Similar to that of DMCC, node n updates the records on the minimum cost to reach its instances from the unpruned zero instances carried inside the search message that they received. Then, it forwards the combined search message to its instances' forward neighbors once it has received all the necessary search messages. Afterwards, it enters the *FIN* state.

After the zero node q_{max} with the maximum mid-cover-point determines and announces the costs of all its unpruned $MCC(q_{max}.r_0)$, where $q_{max}.r_0$ represents the zero instances of q_{max} . All the zero nodes q_i with zero instances a_0 should have found and announced the costs of all the unpruned $MCC(a_0)$. The one with minimum cost is MCC . Therefore, the $\langle SELECT \rangle$ message can be sent by q_i with $min(f(MCC(a_0)))$ to the node $previous(a_0)$ recorded in q_i . The same process can be done by the node with instance which is the target of the $\langle SELECT \rangle$ message. This process terminates when a node

is reached which records $previous(a_0) = a_0$. In the ADMCC algorithm described above, for the ease of discussion, the procedures about the $\langle SELECT \rangle$ message circulation have been eliminated.

Pseudocode: The following pseudocode illustrates the adjustable distributed minimum cost cover algorithm (ADMCC). Algorithm 10 illustrates the action of zero node q_{min} with the minimum mid-cover-point. Algorithm 11 illustrates the action of other zero node q_k . Lastly, Algorithm 12 illustrates the action of non-zero node n .

Algorithm 8 Initialization for ADMCC.

```

1: current node  $i$  with instances  $b$ .
2: for each current node instance  $b$  do
3:   for each zero instance  $a_0$  of node  $q_i$  in the network do
4:      $min\_cost(a_0, b) = \infty$ 
5:      $previous(a_0, b) = \perp$ 
6:   end for
7: end for
8: if  $i$  is a zero node with zero instances  $b_0$  then
9:   for each  $b_0$  do
10:     $min\_cost(b_0, b_0) = f(b_0)$ 
11:     $total\_cost(b_0) = \infty$ 
12:   end for
13: end if

```

Algorithm 9 Pruning Mechanism for ADMCC.

```

1: Current node  $i$  with instances  $c$ .
2: Any zero node  $q_i$  with zero instance  $a_0$  and the search of  $MCC(a_0)$  is unpruned.
3: Any zero node  $q_j$  with zero instance  $b_0$  and the search of  $MCC(b_0)$  is unpruned.
4: for each instance  $c$  do
5:   for each zero instance  $a_0$  and  $min\_cost(a_0, c) \neq \infty$  do
6:     for each zero instance  $b_0$  and  $min\_cost(b_0, c) \neq \infty$  do
7:       if  $a_0 \neq b_0$  then
8:         if  $s_{a_0} < s_{b_0}$  and  $min\_cost(a_0, c) < min\_cost(b_0, c)$  then
9:            $min\_cost(b_0, c) = \infty$ 
10:           $previous(b_0, c) = \perp$ 
11:         end if
12:       end if
13:     end for
14:   end for
15: end for

```

Algorithm 10 Pseudocode of Adjustable Distributed Minimum Cost Cover Algorithm (ADMCC) for a node q_{min} with zero instances a_0 and instances a .

```

/* ----- INIT state ----- */
1: Perform Initialization in Algorithm 8.
2: Send  $\langle a_0, \langle f(a_0), s_{a_0}, a_0 \rangle \rangle$  to forward neighbors.
3: Enter the WAIT state.
/* ----- WAIT state ----- */
1: /* Receiving Search Message M  $\langle e, \langle cost(b_0, e), s_{b_0}, b_0 \rangle \rangle$  from node  $w$  with instances  $e$ .  $b_0$  is a zero instance where the search of  $MCC(b_0)$  is unpruned. */
2: for each tuple  $\langle e, \langle cost(b_0, e), s_{b_0}, b_0 \rangle \rangle$  from backward neighbor instance  $e$  carried in M do
3:   for each instance  $a$  of  $q_{min}$  do
4:     if the instance  $e$  in the tuple is a backward neighbor of  $a$  then
5:       if the instance  $a$  is a zero instance, i.e.,  $a$  is also  $a_0$  and  $a_0 = b_0$  then
6:         /*This tuple is for searching the unpruned  $MCC(a_0)$ */
7:         Update the cost of  $MCC(a_0)$ .
8:       else
9:         Update the current minimum cost to reach  $a$  from another zero instance  $b_0$ .
10:      end if
11:    end if
12:  end for
13: end for
14: if Received all search message for all its instances  $a$  then
15:   Perform Pruning Mechanism in Algorithm 9.
16:   Send  $\langle a, \langle min\_cost(b_0, a), s_{b_0}, b_0 \rangle \rangle, \forall min\_cost(b_0, a) \neq \infty$  and  $s_a \leq s_{b_0}$  to all forward neighbors
17:   Enter the FIN state.
18: end if

```

Algorithm 11 Pseudocode of Adjustable Distributed Minimum Cost Cover Algorithm (ADMCC) for a zero node q_k with zero instances c_0 and instances c .

```

/* ----- INIT state ----- */
1: Perform Initialization in Algorithm 8.
2: Enter the WAIT1 state.
/* ----- WAIT1 state ----- */
1: /* Receiving Search Message M  $\langle e, \langle cost(b_0, e), s_{b_0}, b_0 \rangle \rangle$  from  $w$  with instances  $e$  and  $b_0 \in S'_0$ . */
2: for each tuple  $\langle cost(b_0, e), s_{b_0}, b_0 \rangle$  from instance  $e$  carried in M do
3:   for each instance  $c$  of  $q_k$  do
4:     if  $e$  is a backward neighbor of  $c$  then
5:       Update the minimum cost to reach  $c$  from zero instance  $b_0$ .
6:     end if
7:   end for
8: end for
9: if Received all search message for all its instances  $c$  from their important backward neighbors. then
10:   Perform Pruning Mechanism in Algorithm 9.
11:   Send  $\langle c, \langle min\_cost(b_0, c), s_{b_0}, b_0 \rangle \rangle, \forall min\_cost(b_0, c) \neq \infty$  to all forward neighbors
12:   Enter the WAIT2 state.
13: end if
/* ----- WAIT2 state ----- */
/* Same as the WAIT state of Algorithm 10. */

```

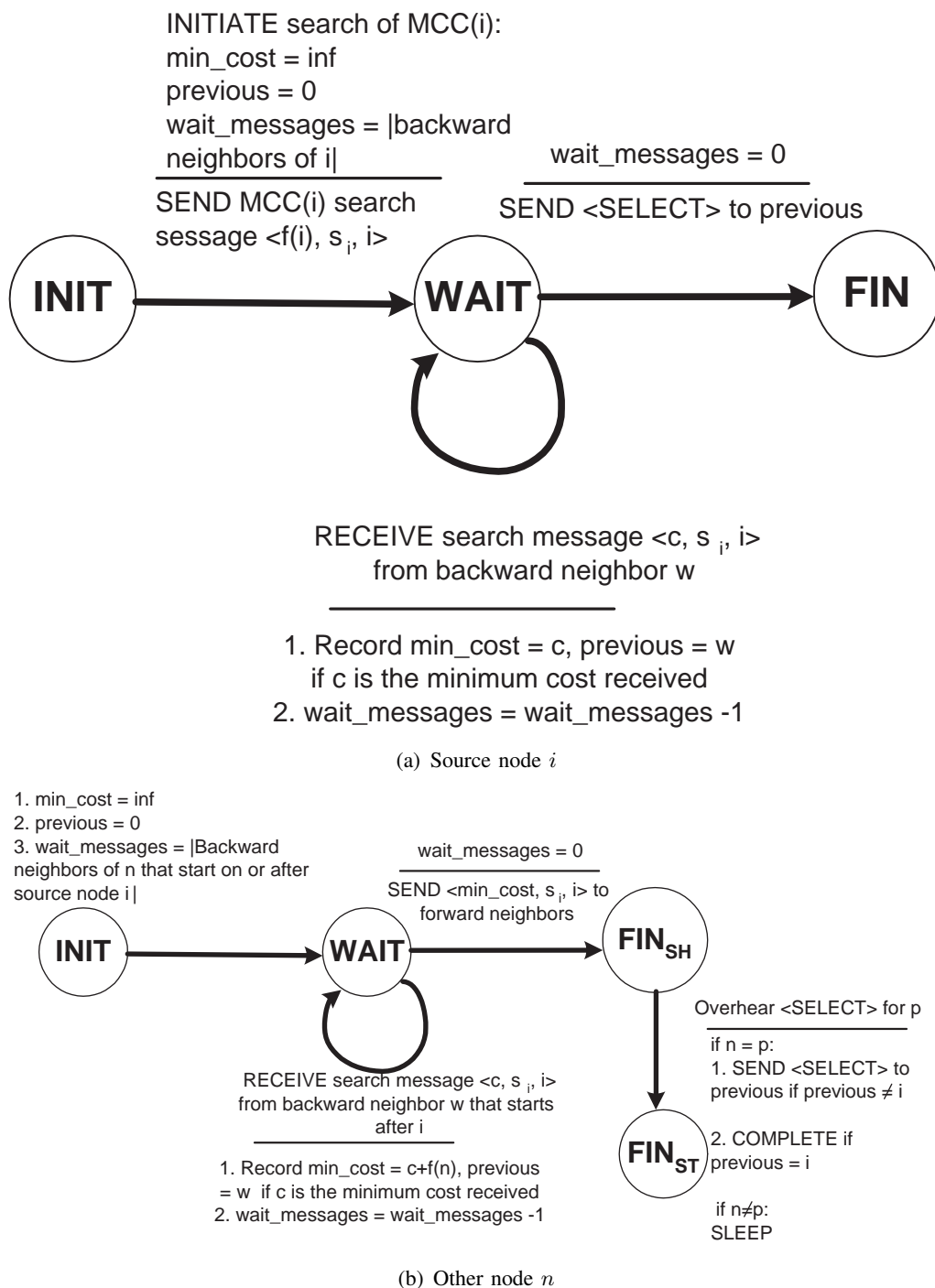


Fig. 4. State diagram of $MCC(i)$ search algorithm.

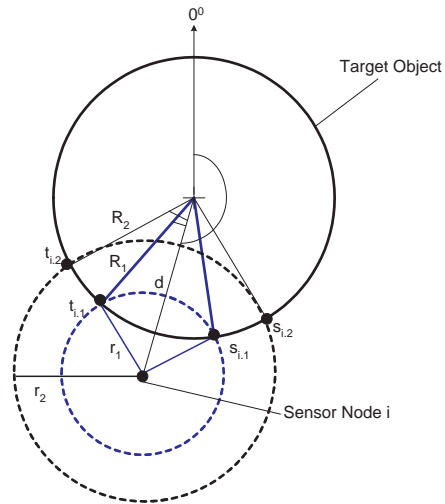


Fig. 5. Adjustable Sensing Range Example

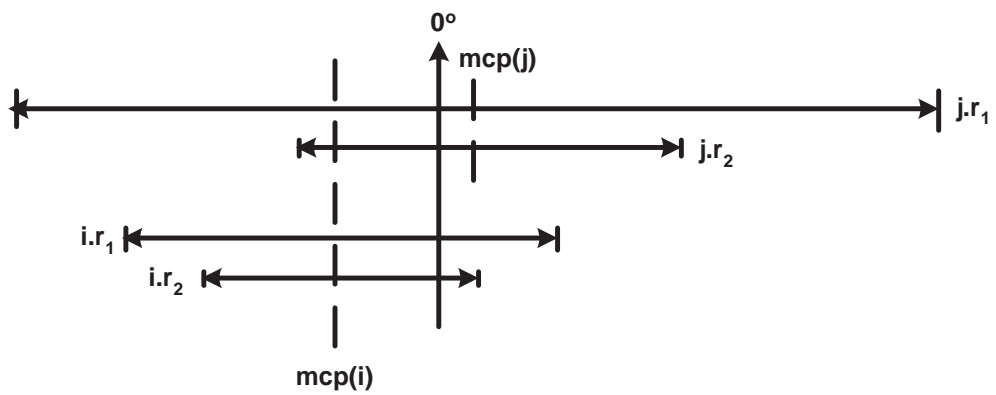


Fig. 6. An example of $mcp(i) < mcp(j)$.

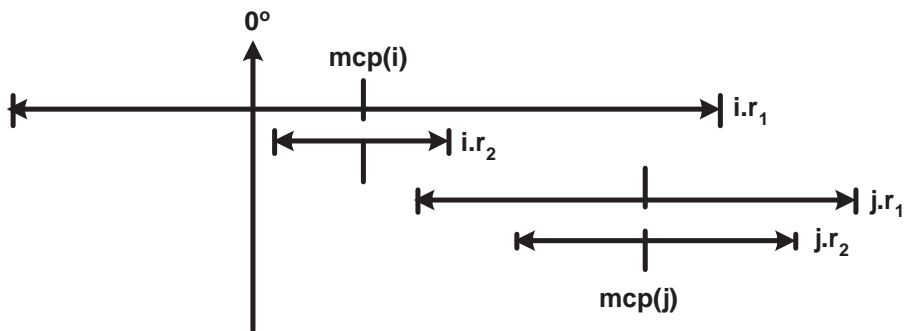


Fig. 7. An initiator with both zero and non-zero instances.

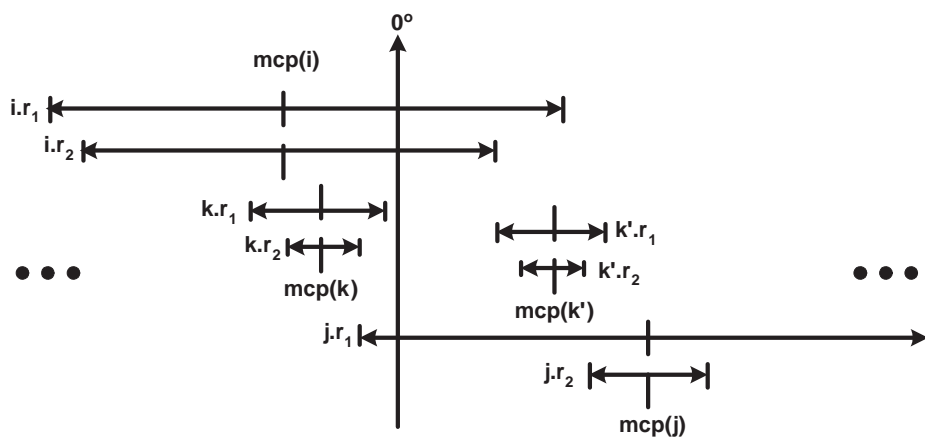


Fig. 8. Order of nodes in ADMCC

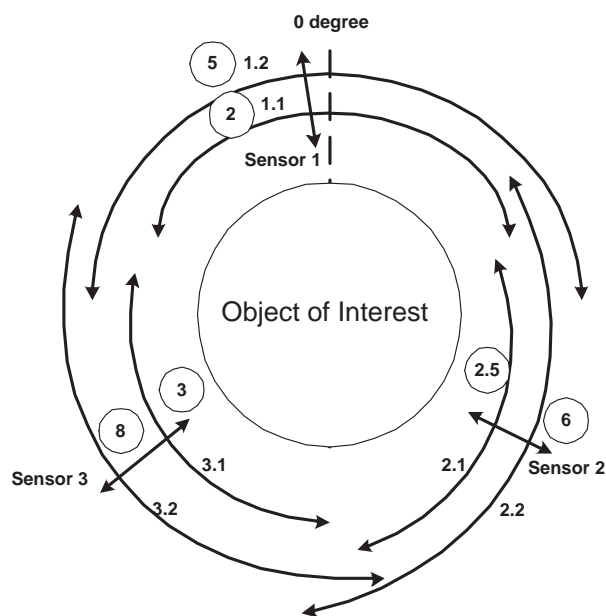


Fig. 9. Adjustable Distributed Minimum Cost Cover Algorithm Example

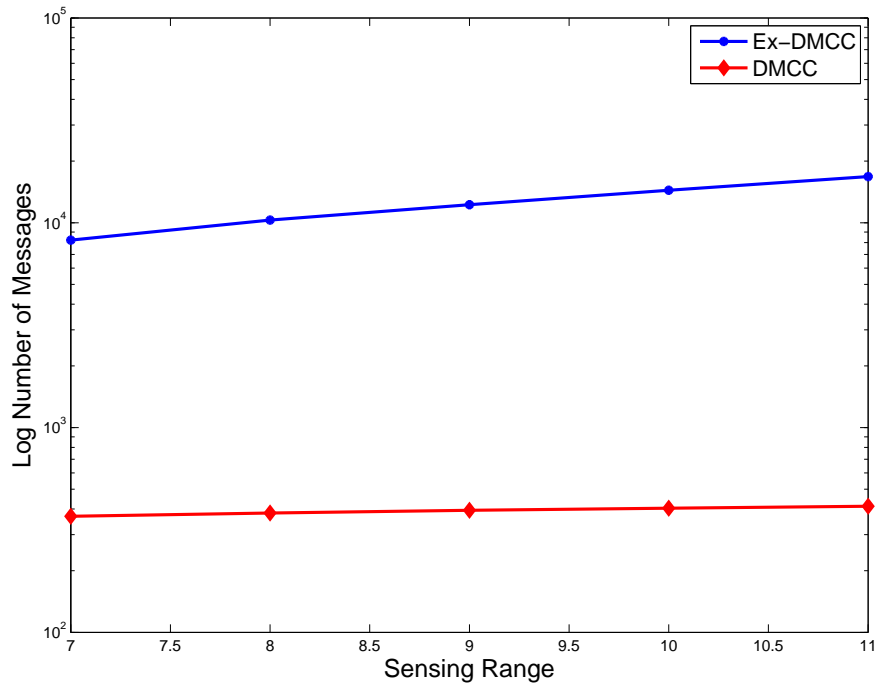


Fig. 10. Log Number of Messages Vs. Sensing Range.

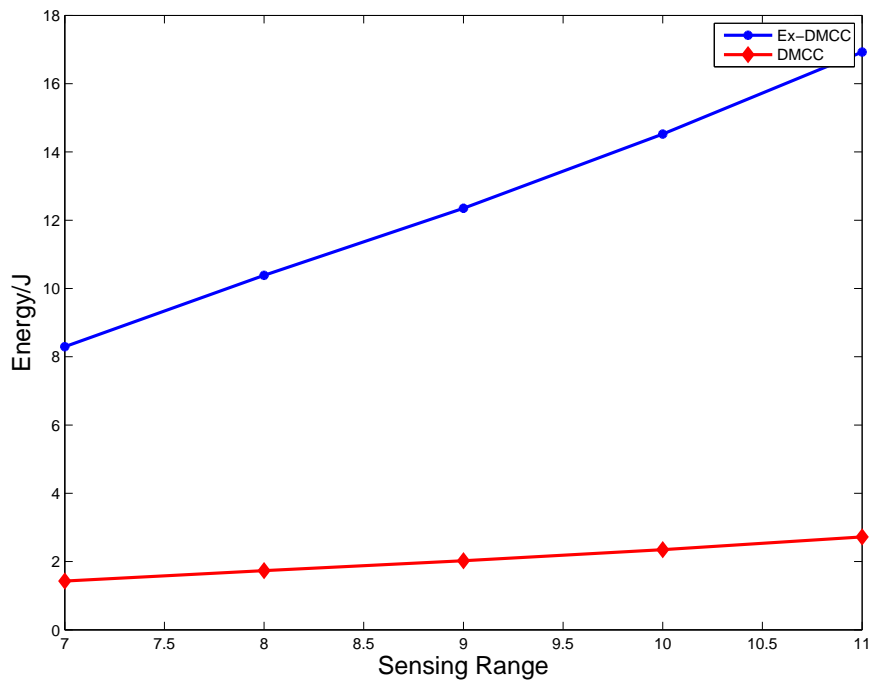


Fig. 11. Estimated Transmission Energy Expenditure Vs. Sensing Range.

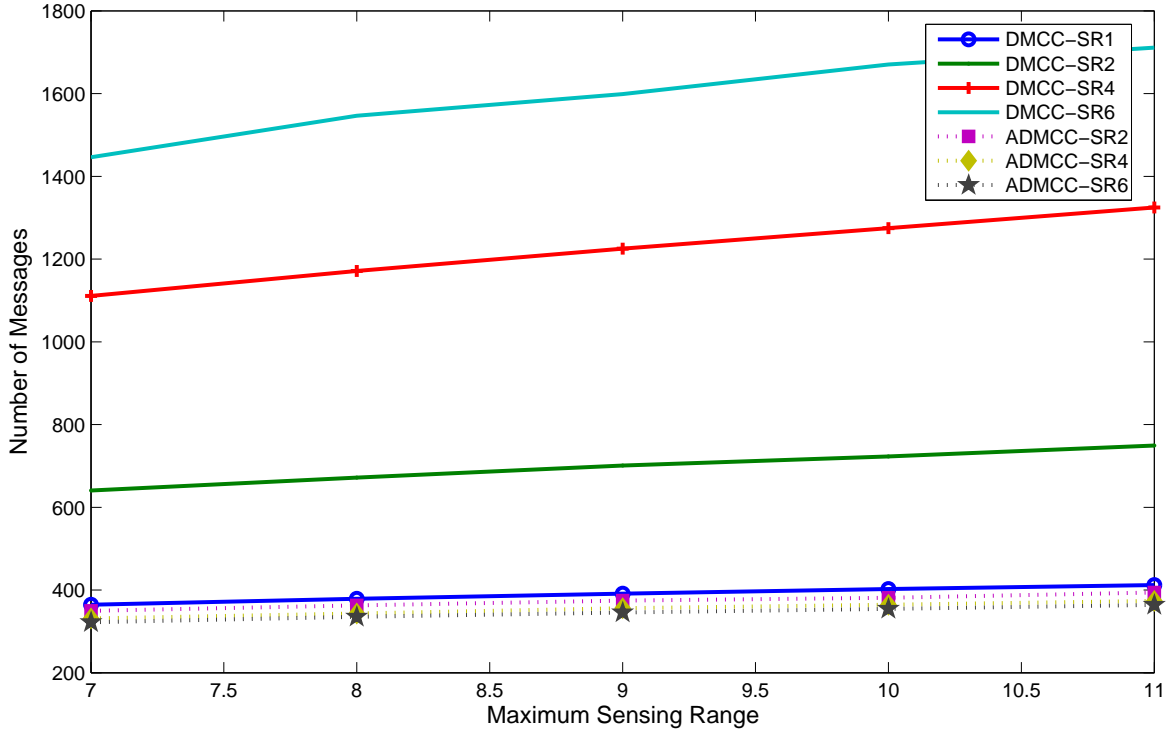


Fig. 12. Number of Messages Vs. Maximum Sensing Range.

Algorithm 12 Pseudocode of Adjustable Distributed Minimum Cost Cover Algorithm (ADMCC) for non-zero node n with instances d

```

/* ----- INIT state ----- */
1: Perform Initialization in Algorithm 8.
2: Enter the WAIT state.
/* ----- WAIT state ----- */
1: /* Receiving Search Message M  $\langle e, \langle cost(b_0, e), s_{b_0}, b_0 \rangle \rangle$  from  $w$  with instances  $e$ .  $b_0$  is the zero instance of unpruned  $MCC(b_0)$ . */
2: for each tuple  $\langle e, \langle cost(b_0, e), s_{b_0}, b_0 \rangle \rangle$  from instance  $e$  carried in M do
3:   for each instance  $d$  of node  $n$  do
4:     if  $e$  is a backward neighbor of instance  $d$  then
5:       Update the minimum cost to reach the instance  $d$  from the zero instance  $b_0$ .
6:     end if
7:   end for
8: end for
9: if Received all search message for all its instances  $d$  from their backward neighbors. then
10:   Perform Pruning Mechanism in Algorithm 9.
11:   Send  $\langle d, \langle min\_cost(b_0, d), s_{b_0}, b_0 \rangle \rangle, \forall min\_cost(b_0, d) \neq \infty$  to all forward neighbors
12:   Enter the FIN state.
13: end if

```

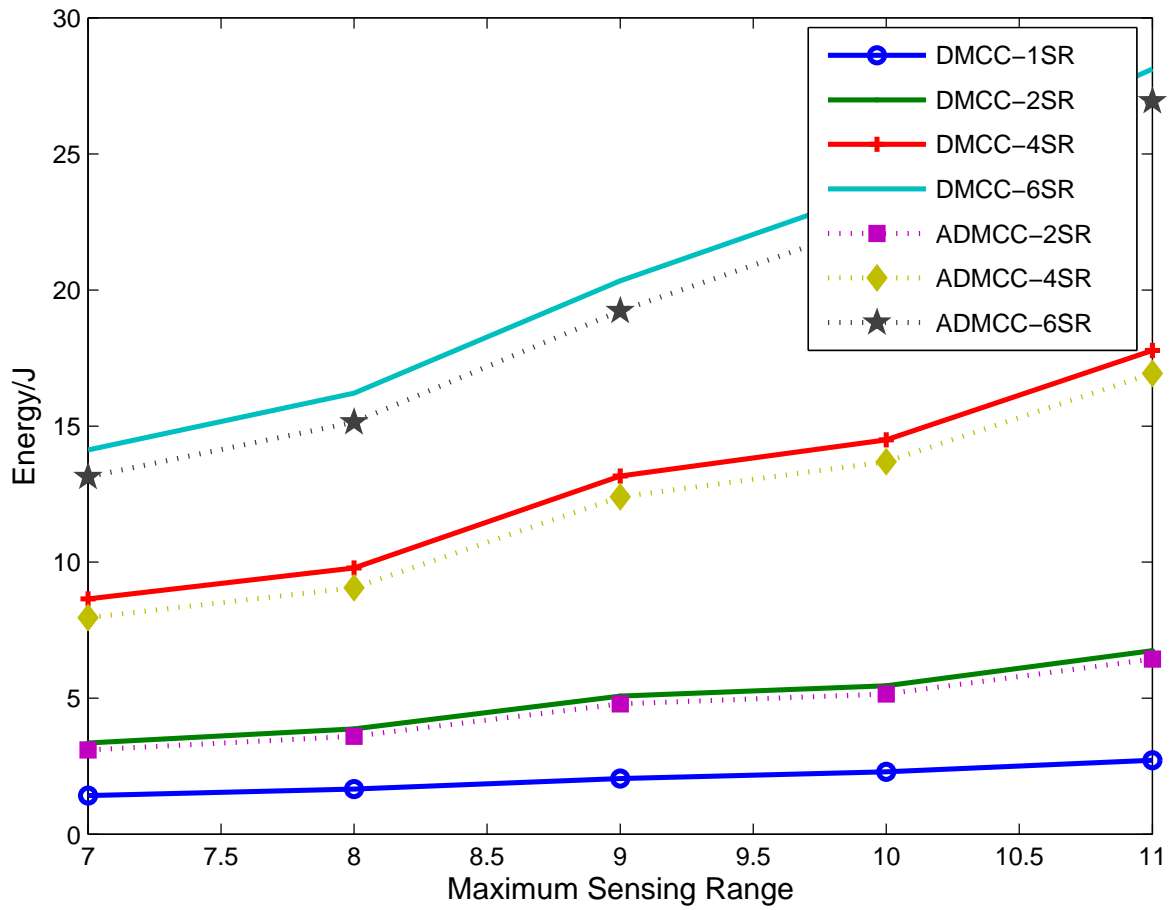


Fig. 13. Estimated Transmission Energy Expenditure Vs. Sensing Range.

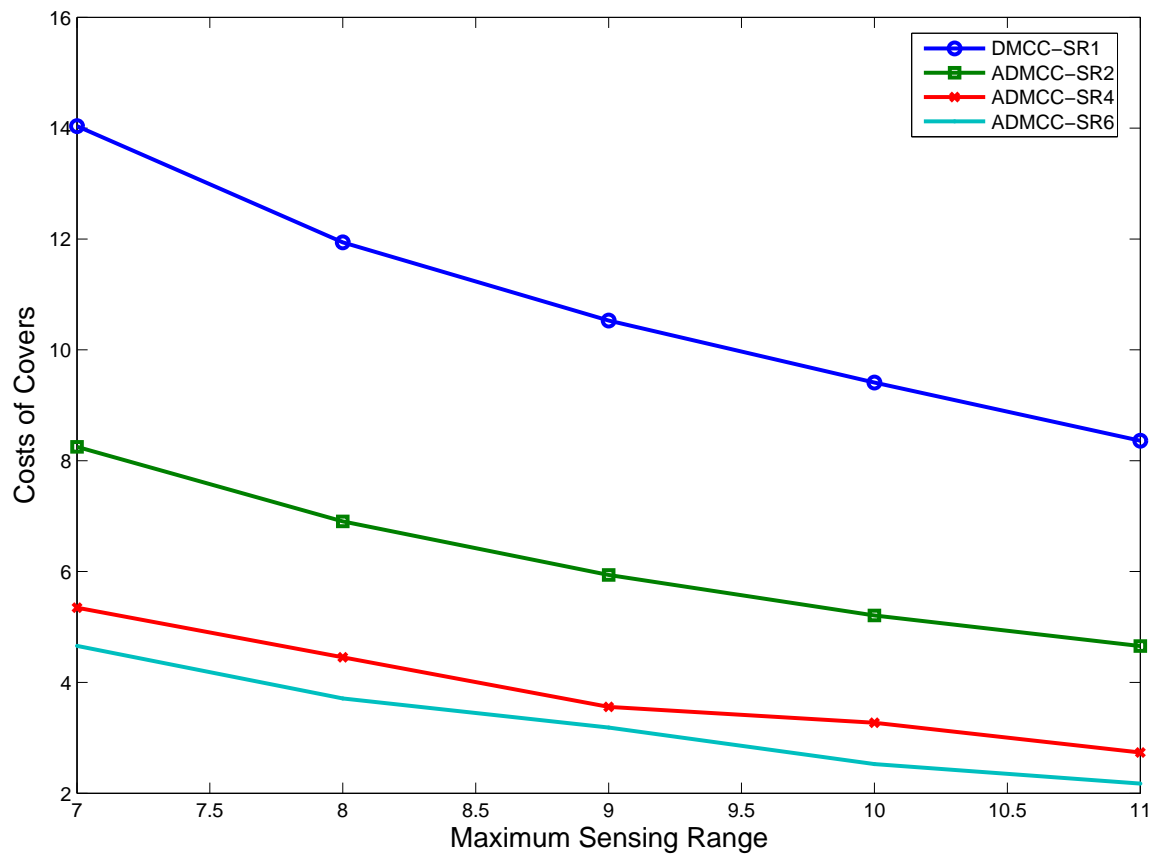
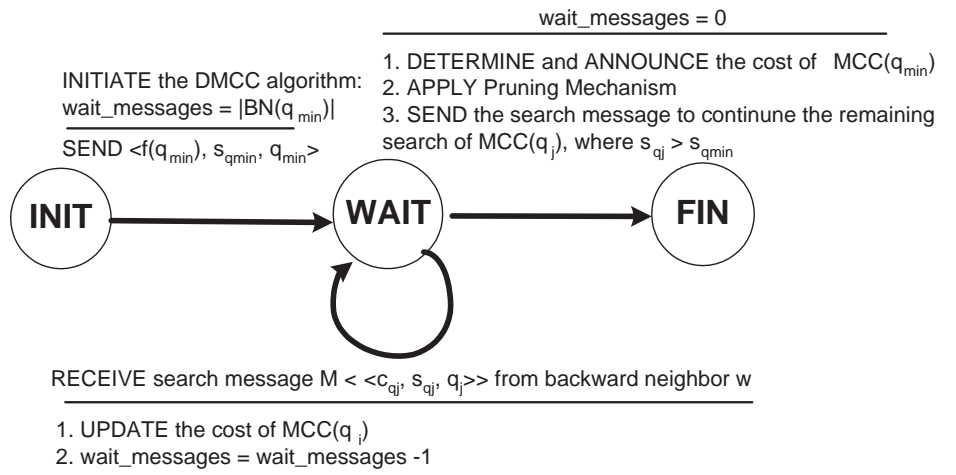
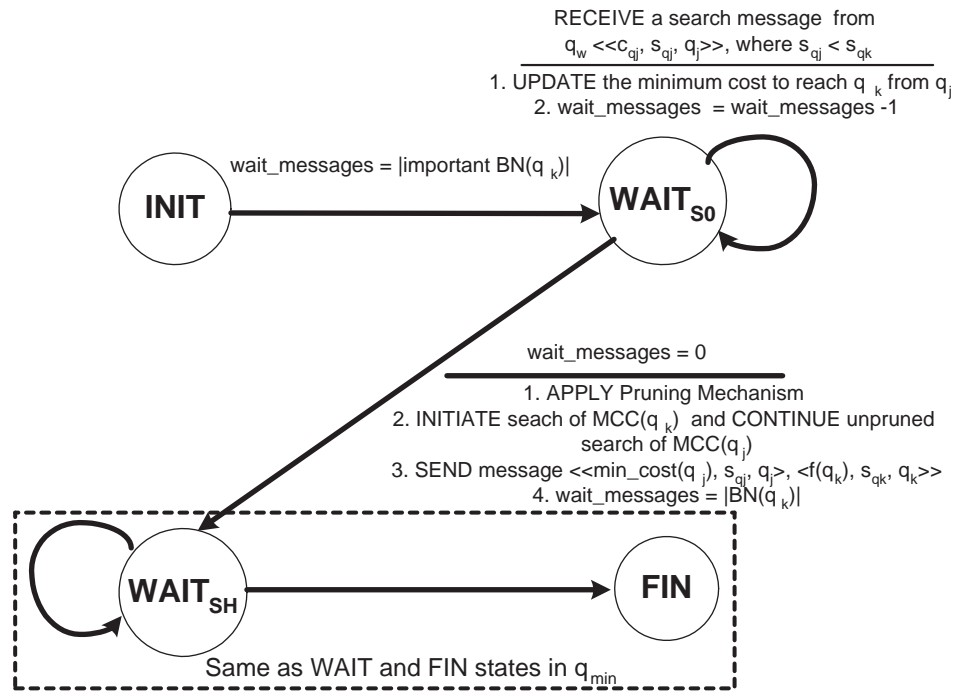


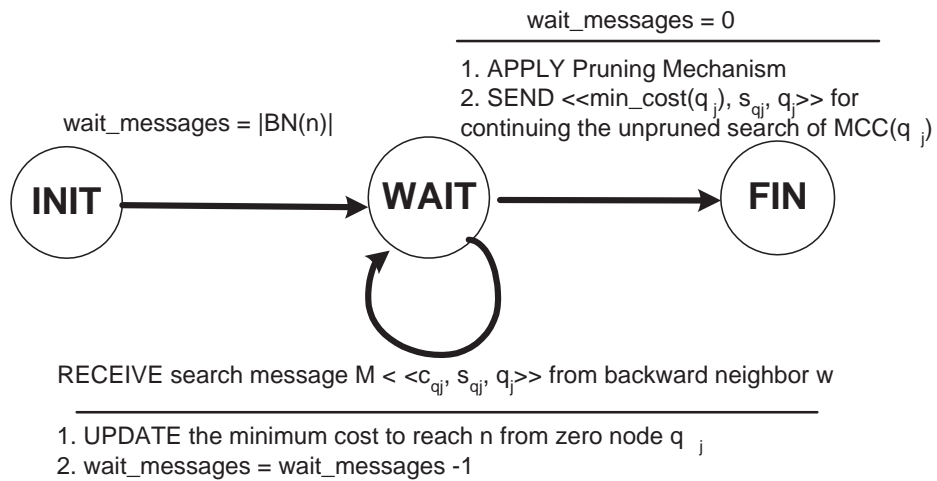
Fig. 14. Cover Costs Vs. Maximum Sensing Range.



(a) Zero node q_{min} with smallest start angle.



(b) Other zero node q_k .



(c) Non-zero node n .

Fig. 15. State diagrams of different nodes in DMCC.

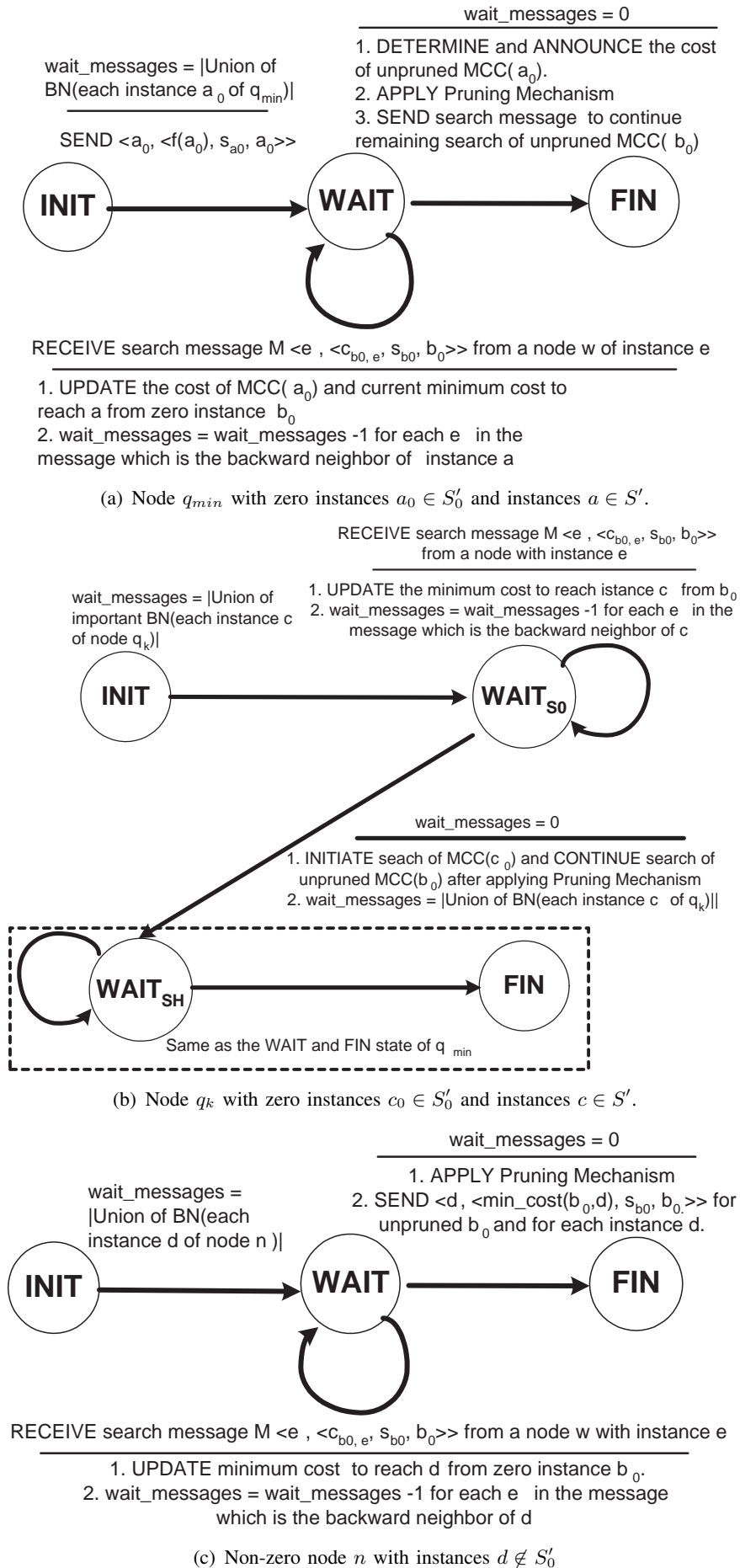


Fig. 16. State diagrams of different nodes in ADMCC.