This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2007 proceedings.

Target-Oriented Scheduling in Directional Sensor Networks

Yanli Cai, Wei Lou, Minglu Li and Xiang-Yang Li

***Abstract*— Unlike convectional omni-directional sensors that always have an omni-angle of sensing range, *directional sensors* may have a limited angle of sensing range due to technical constraints or cost considerations. A directional sensor network consists of a number of directional sensors, which can switch to several directions to extend their sensing ability to cover all the targets in a given area. Power conservation is still an important issue in such directional sensor networks. In this paper, we address the *multiple directional cover sets problem (MDCS)* of organizing the directions of sensors into a group of non-disjoint cover sets to extend the network lifetime. One cover set, in which the directions cover all the targets, is activated at one time. We prove the MDCS to be NP-complete and propose three heuristic algorithms for the MDCS. Simulation results are also presented to demonstrate the performance of these algorithms.**

I. Introduction

In recent years, sensor networks have emerged as promising platforms for many applications, such as environmental monitoring, battlefield surveillance, and health care [1], [2]. A sensor network may consist of a large number of small sensor nodes that are composed of sensing, data processing and communicating components. The conventional research of sensor networks are always based on the assumption of *omni-directional sensors* that have an omni-angle of sensing range. However, sensors may have a limited angle of sensing range due to technical constraints or cost considerations, which are denoted as *directional sensors* in this paper. Video sensors [3], [4], ultrasonic sensors [5] and infrared sensors [2] are examples of widely used directional sensors. Note that the directional characteristic we discuss in this paper is from the point of view of the sensing, but not from the communicating activity of sensor nodes.

There are several ways to extend the sensing ability of directional sensors. One way is to put several directional sensors of the same kind on one sensor node, each of which faces to a different direction. One example using this way is in [5], where four pairs of ultrasonic sensors are equipped on a single node to detect ultrasonic signals from any direction. Another way is to equip the sensor node with a mobile device that enables the node to move around. The third way is to equip the sensor node with a device that enables the sensor on the node to switch (or rotate) to different directions. We adopt the third way so that a sensor can face to several directions. In this paper, we assume that each sensor node equips exactly one sensor on it. Therefore, we do not differentiate the terms *sensor* and *node* in the rest of the paper.

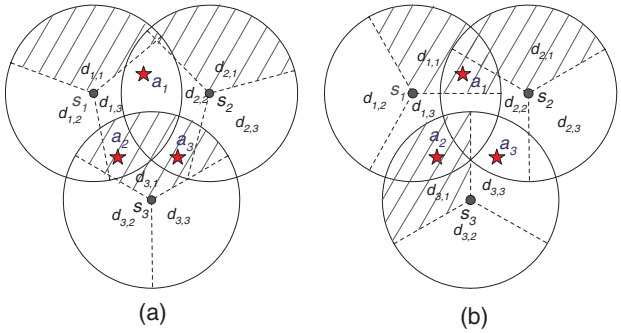

Fig. 1. Simple directional sensor networks

We also consider the following scenario. Some targets with known locations are deployed in a two-dimensional Euclidean plane. A number of directional sensors are randomly scattered close to these targets. We assume the sensing region of each direction of a directional sensor is a sector of the sensing disk centered at the sensor with a sensing radius. Each sensor has a uniform sensing region and the sensing regions of different directions of a sensor do not overlap. However, the algorithms proposed in this paper do not put restrictions on the shape of the sensing region or overlaps between different directions. When the sensors are randomly deployed, each sensor initially faces to one of its directions. These sensors form a directional sensor network so that data can be gathered and transferred to the sink, a central processing base station.

If a directional sensor faces to a direction, we say that the sensor *works in* this direction and the direction is the *work direction* of the sensor. When this sensor works in a direction and a target is in the sensing region of the sensor, we say that the direction of the sensor *covers* the target. Because a directional sensor has a smaller angle of sensing range than an omni-directional sensor or even does not cover any target when it is deployed, we need to schedule sensors in the network to face to certain directions to cover all the targets. We call a subset of directions of the sensors, in which the directions cover all the targets, as a *cover set*. Note that no more than one direction of a sensor can be in a cover set. The problem of finding a cover set, called *directional cover set problem*

Yanli Cai is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, and the Department of Computing, The Hong Kong Polytechnic University, Hong Kong (Email: cai-yanli @cs.sjtu.edu.cn).

Wei Lou is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong (Email: csweilou@comp.polyu.edu.hk).

Minglu Li is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China (Email: li-ml@cs.sjtu.edu.cn).

Xiang-Yang Li is with the Department of Computer Science, Illinois Institute of Technology, USA (Email: xli@cs.iit.edu).

0743-166X/07/$25.00 ©2007 IEEE

(DCS), is proved to be NP-complete in this paper.

Fig. 1(a) shows a simple directional sensor network. The black point s_1 is a directional sensor that can switch to three directions $d_{1,1}$, $d_{1,2}$ and $d_{1,3}$. Direction $d_{1,1}$ is the direction to which the sensor faces when it is deployed, and the shadowed sector above $d_{1,1}$ is the sensing region of s_1 when it works in $d_{1,1}$. The stars a_1, a_2 and a_3 are three targets. Although the direction $d_{1,1}$ does not cover any target, s_1 can switch to $d_{1,3}$ to cover both a_1 and a_2. The directions $d_{1,3}$ of s_1 and $d_{3,1}$ of the sensor s_3 together cover all the targets in Fig. 1(a). Therefore, $\{d_{1,3}, d_{3,1}\}$ is a cover set for the three targets.

Power conservation is still an important issue in directional sensor networks due to the following reasons. First, most sensors have limited power sources and are non-rechargeable. Also, the batteries of the sensors are hard to replace due to hostile or inaccessible environments in many scenarios. We assume that each sensor is non-rechargeable and dies when it runs out its power. To conserve energy, we can leave necessary sensors in the *active* state and put redundant sensors into the *sleep* state, while keeping all the targets covered.

In this paper, our objective is to maximize the network lifetime of a directional sensor network, where the network lifetime is defined as the time duration when each target is covered by the work direction of at least one active sensor. Our approach is to organize the directions of sensors into non-disjoint subsets, each of which is a cover set, and allocate the work time for each cover set. Note that non-disjoint cover sets allow a direction or a sensor to participate in multiple cover sets. We alternately activate only one cover set at any time. When one cover set is activated, each sensor that has a direction in this cover set is in the active state and works in this direction, while all the other sensors are in the sleep state. We call the problem of finding non-disjoint cover sets and allocating the work time for each of them to maximize the network lifetime as *multiple directional cover sets problem (MDCS)*.

The main contributions of this paper are as follows: We formally define the DCS and the MDCS. We prove that both problems are NP-complete. We present a heuristic algorithm named *Progressive* for the MDCS. Then an enhanced algorithm named *Prog-Resd* is proposed to gain a longer network lifetime. We also propose an algorithm named *Feedback* that uses the results obtained in previous iterations as a feedback to the next iteration. This algorithm aims to get a longer network lifetime and fewer cover sets which are more efficient and practical.

The remainder of the paper is organized as follows: Section II briefly surveys the related works in the literature. In Section III, the DCS and the MDCS are formally defined and proved to be NP-complete. In Section IV, we formulate the MDCS as an optimization problem. In Section V, the *Progressive* and *Prog-Resd* algorithms are presented. In Section VI, the *Feedback* algorithm is described. Simulation results are presented to show the effectiveness of the proposed algorithms in Section VII. We conclude the paper and outline directions of future work in Section VIII.

II. Related work

A number of scheduling algorithms have been proposed to prolong the network lifetime for omni-directional sensor networks. Sleeping protocols, such as PEAS [6], PECAS [7], ACOS[8] and OGDC [9], have used different strategies to extend the network lifetime, while trying to achieve the largest area coverage, which represents how well a region of interest is monitored. Scheduling problems are studied in [10], [11], [12] when a set of targets are deployed in a region. [11] assumes that a sensor can watch only one target at a time, and builds a target watching timetable for each sensor to maximize the network lifetime. [12] organizes sensors into mutually exclusive subsets that are activated successively, where the size of each subset is restricted and not all of the targets need to be covered by the sensors in one subset. Unlike [11] and [12], [10] aims to extend the lifetime of an omni-directional sensor network by organizing the sensors into non-disjoint subsets, where each target must be covered by at least one sensor in each subset. This problem is proved to be NP-complete in [10], although finding a subset of omni-directional sensors to cover all the targets can be done in a polynomial time. Note that the problem discussed in [10] is a special case of the MDCS where a sensor has only one direction.

Recently, some efforts have been devoted to the research of directional sensor networks. [13] provides a directional sensor model where each sensor is fixed to one direction and analyzes the probability of full area coverage. In [14], a similar directional sensor model is proposed, where a sensor is allowed to work in several directions. The objective of [14] is to find a minimal set of directions that can cover the maximal number of targets. It is different from the one in this paper that aims to find a group of non-disjoint cover sets in each of which the directions cover all the targets so as to maximize the network lifetime.

III. Multiple Directional Cover Sets Problem

In this section, we first define the notations. Then we give some simple examples of the MDCS to briefly describe this problem. We also formally define the DCS and the MDCS and prove that both problems are NP-complete.

A. Notations and Assumptions

We adopt the following notations throughout the paper.

- M: the number of targets.
- N: the number of sensors.
- W: the number of directions per sensor.
- a_m: the m^{th} target, $1 \leq m \leq M$.
- s_i: the i^{th} sensor, $1 \leq i \leq N$.
- $d_{i,j}$: the j^{th} direction of the i^{th} sensor, $1 \leq i \leq N$, $1 \leq j \leq W$. We define $d_{i,j} = \{a_m | \, a_m$ is covered by $d_{i,j}, \forall a_m \in A\}$ and $s_i = \{d_{i,j} | \, j = 1 \ldots W\}$. Hence, if $a_m \in d_{i,j}$, a_m is covered by $d_{i,j}$.
- A: the set of targets. $A = \{a_1, a_2, \ldots, a_M\}$.
- S: the set of sensors. $S = \{s_1, s_2, \ldots, s_N\}$.

- D: the set of the directions of all the sensors. $D = \{d_{i,j} | \, i = 1 \ldots N, j = 1 \ldots W\}$. Notice that $\bigcup_{i=1}^{N} s_i$ is a non-overlapped partition of D.
- L_i: the lifetime of a sensor s_i, which is the time duration when the sensor is in the active state all the time.

In this paper, we assume that a directional sensor with W directions has a lifetime W times of that of the same omni-directional sensor. For simplicity, we assume each sensor initially has an equal lifetime. Moreover, we assume that the energy consumed for switching a sensor from one direction to another can be omitted.

B. Simple Examples of MDCS

To briefly describe the MDCS, simple examples are illustrated here. Fig. 1 shows two directional sensor networks, both of which have three sensors s_1, s_2 and s_3 deployed to monitor three targets a_1, a_2 and a_3. Each sensor has an initial lifetime of 1 (time unit). Sensor s_1 has three directions $d_{1,1}$, $d_{1,2}$ and $d_{1,3}$, s_2 has $d_{2,1}$, $d_{2,2}$ and $d_{2,3}$, and s_3 has $d_{3,1}$, $d_{3,2}$ and $d_{3,3}$.

For the network deployment in Fig. 1(a), we can get the following cover sets: $D_1 = \{d_{1,3}, d_{3,1}\}$ with 0.5, $D_2 = \{d_{1,3}, d_{2,2}\}$ with 0.5, and $D_3 = \{d_{2,2}, d_{3,1}\}$ with 0.5. This results in a network lifetime of 1.5. On the other hand, if a sensor is not allowed to participate in multiple cover sets, for the network deployment in Fig. 1(a), we can get $D_1 = \{d_{1,3}, d_{3,1}\}$ with its work time 1, which is the maximal network lifetime.

For the network deployment in Fig. 1(b), we can get a cover set $D_1 = \{d_{1,3}, d_{2,2}\}$ with its available work time 1. This results in a network lifetime of 1.

C. Problem Definition

To prove the NP-completeness of the MDCS, we formally provide the following definitions:

Definition 1. *Cover Set:* Given a collection D of subsets of a finite set A and a partition S of D, a *cover set* for A is a subset $D^{'} \subseteq D$ such that every element in A belongs to at least one member of $D^{'}$ and every two elements in $D^{'}$ cannot belong to the same member of S.

Definition 2. *Directional Cover Set Problem (DCS)*: Given a collection D of subsets of a finite set A and a partition S of D, find a cover set for A.

Definition 3. *Multiple Directional Cover Sets Problem (MDCS)*: Given a collection D of subsets of a finite set A and a partition S of D, find a family of K cover sets $D_1, D_2, \ldots, D_K \subseteq D$ for A, with nonnegative weights $t_1, t_2, \ldots, t_K$, such that $t_1 + t_2 + \ldots + t_K$ is maximized, and for each $s \in S$, $\sum_{i=1}^{K} |s \cap D_i| \cdot t_i \leq L$, where L is a given positive number.

Note that $|s \cap D_i|$ indicates the number of the directions of s that are in D_i, where $|s \cap D_i| = 0$ or 1 since no more than one direction of a sensor can work in a cover set.

D. NP-completeness

In this subsection, we first prove the DCS to be NP-complete by reduction from the 3-Conjunctive Normal Form-Satisfiability (*3-CNF-SAT*) problem [15]. Then we prove the MDCS to be NP-complete by reduction from the DCS.

The decision versions of both the DCS and the MDCS are defined as follows:

Definition 4. *Decision Version of the DCS*: Given a collection D of subsets of a finite set A and a partition S of D, determine if there exists a cover set for A.

Definition 5. *Decision Version of the MDCS*: Given a collection D of subsets of a finite set A and a partition S of D, determine if there exists a family of K cover sets $D_1, D_2, \ldots, D_K \subseteq D$ for A, with nonnegative weights $t_1, t_2, \ldots, t_K$, such that $t_1 + t_2 + \ldots + t_K \geq p$, and for each $s \in S$, $\sum_{i=1}^{K} |s \cap D_i| \cdot t_i \leq L$, where L is a given positive number.

The following theorems show that both the DCS and the MDCS are NP-complete.

Theorem 1. *The DCS is NP-complete.*

Proof: We first show that the DCS $\in$ NP. Suppose that a set $D^{'}$ is given as a certificate. The verification algorithm first affirms $D^{'} \subseteq D$, and then it checks that if each element in A belongs to at least one member of $D^{'}$. At last, it checks that if each member of S contains no more than one element in $D^{'}$. The verification can be done in a polynomial time. Therefore, the DCS $\in$ NP.

To prove that the decision version of the DCS is NP-hard, we show a polynomial time reduction from the *3-CNF-SAT* problem to the DCS.

For the *3-CNF-SAT* problem, a boolean formula F consisting of m clauses and n variables is in 3-conjunctive normal form, i.e., $F = c_1 \wedge c_2 \wedge \ldots \wedge c_m$, where each clause $c_j = x_{j,1} \vee x_{j,2} \vee x_{j,3}$ and each literal $x_{j,k} \in \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$. From the given formula F, an instance of the DCS is constructed as follows:

1) $A = \{c_j | \, j = 1 \ldots m\}$.
2) For each x_i, define a set $d_{i,1} = \{c_j | \, c_j \text{ contains } x_i, \ 1 \leq j \leq m\}$.
3) For each $\overline{x}_i$, define a set $d_{i,2} = \{c_j | \, c_j \text{ contains } \overline{x}_i, \ 1 \leq j \leq m\}$.
4) $D = \{d_{i,1} | \, i = 1 \ldots n\} \cup \{d_{i,2} | \, i = 1 \ldots n\}$.
5) $s_i = \{d_{i,1}, d_{i,2}\}$, $S = \{s_i | \, i = 1 \ldots n\}$.

This reduction can be finished in a polynomial time.

We now show that the formula F is satisfiable iff the instance of the DCS has a cover set. If the formula is satisfiable, for every clause c_j, at least one of its literals is true. Picking the true literals from each clause yields a subset $D^{'}$ of D since each literal in the *3-CNF-SAT* corresponds to an element in D. Each $c_j \in A$ belongs to at least one member of $D^{'}$, which corresponds to one of its chosen literals. As x_i and $\overline{x}_i$ cannot both be true, the corresponding $d_{i,1}$ and $d_{i,2}$ in D cannot both be chosen into $D^{'}$, i.e., every two elements in

$D^{'}$ do not belong to the same $s \in S$. Therefore, $D^{'}$ is a cover set for A.

Conversely, suppose that the instance of the DCS has a cover set $D^{'}$. Since each element in $D^{'}$ corresponds to a literal in the *3-CNF-SAT*, we can assign true to these corresponding literals. Any literal and its complement are not both true because the corresponding elements in $D^{'}$ cannot belong to the same $s \in S$. Every clause is true because it belongs to at least one member of $D^{'}$, i.e., at least one of its literals is true. Therefore, the formula is satisfied.

Since the DCS is both NP and NP-hard, we conclude that the DCS is NP-complete. □

Theorem 2. *The MDCS is NP-complete.*

Proof: We first show that the MDCS ∈ NP. Given a solution $D_1, D_2, \ldots, D_K$ with weight $t_1, t_2, \ldots, t_K$, and a number p, the verification algorithm can verify whether $D_1, D_2, \ldots, D_K$ are cover sets in polynomial time as we have shown in the proof of Theorem 1. Checking $t_1 + t_2 + \ldots + t_K \geq p$ and all the members of s appear in $D_1, D_2, \ldots, D_K$ with a total weight of at most L for each $s \in S$ can also be done in a polynomial time. Therefore, the MDCS ∈ NP.

To prove that the decision version of the MDCS is NP-hard, we give the MDCS a polynomial time reduction from the DCS, which has been proved to be NP-complete in Theorem 1.

Given a DCS instance with a collection D^1 of a finite set A^1 and a collection S^1 of subsets of D^1, we construct an instance of the MDCS by setting $A = A^1$, $D = D^1$, $S = S^1$, $K = 1$, $L = 1$ and $p = 1$. If the instance of the DCS has a cover set $D^{'}$, we get a solution $D_1 = D^{'}$ with $t_1 = 1$ for the instance of the MDCS, and vice versa. This proves that the MDCS is NP-hard. As the MDCS ∈ NP, the MDCS is NP-complete. □

From the proof of Theorem 2, we can see that the DCS is a subclass of the MDCS where K, the number of cover sets, is restricted to 1.

IV. Optimization Formulation of MDCS

In this section, we first model the MDCS as a Mixed Integer Programming (MIP) problem [16]. Since the MDCS is NP-complete, it is unlikely to solve the MIP problem of the MDCS in polynomial time. Therefore, we relax the integrality restrictions in the MIP problem to get a Linear Programming (LP) problem which is used in the heuristic algorithms of the following sections.

Consider a directional sensor network with a set A of M targets, a set S of N sensors and a set D of directions. Each sensor $s_i \in S$ has W directions and an initial lifetime of L_i.

We organize the directions in D into K cover sets. The k^{th} cover set is denoted as D_k, with the work time t_k. A direction $d_{i,j}$ is allowed to participate into multiple cover sets. We set a boolean variable $x_{i,j,k}$ as

$$x_{i,j,k} = \begin{cases} 1 & \text{, if } d_{i,j} \in D_k, \\ 0 & \text{, otherwise.} \end{cases} \quad (1)$$

The MIP problem formulated for the MDCS is as follows:

$$\max\ t_1 + t_2 + \ldots + t_K \quad (2)$$

subject to

$$\sum_{k=1}^{K} \sum_{j=1}^{W} x_{i,j,k} \cdot t_k \leq L_i, \forall s_i \in S \quad (3)$$

$$\sum_{j=1}^{W} x_{i,j,k} \leq 1, \forall s_i \in S, k = 1 \ldots K \quad (4)$$

$$\sum_{\substack{a_m \in d_{i,j} \\ d_{i,j} \in D}} x_{i,j,k} \geq 1, \forall a_m \in A, k = 1 \ldots K \quad (5)$$

$$\text{where } x_{i,j,k} = \{0, 1\}, \text{ and } t_k \geq 0 \quad (6)$$

The objective function (2) maximizes the total work time of all the K cover sets. The constraint (3) shows the lifetime constraint for each sensor. The W directions of any sensor work across all the cover sets for no more than the initial lifetime of the sensor. The constraint (4) indicates the exclusivity among different directions of a single sensor, i.e., no more than one direction of the sensor can work in a cover set. The constraint (5) represents the coverage guarantee for each target. For each cover set, every target in A must be covered by at least one direction of this cover set. The constraint (6) shows the restrictions on the variables. The variable $x_{i,j,k}$ can be either 1 or 0, i.e., the direction $d_{i,j}$ works either in the k^{th} cover set or not.

As there exists $x_{i,j,k} \cdot t_k$ in constraint (3), the MIP problem is not linear. Let $t_{i,j,k} = x_{i,j,k} \cdot t_k$. The variable $t_{i,j,k}$ indicates the work time of $d_{i,j}$ in the cover set D_k. We get the following Linear Mixed Integer Programming (LMIP) problem with the objective function (2) and the following constraints:

$$\sum_{k=1}^{K} \sum_{j=1}^{W} t_{i,j,k} \leq L_i, \forall s_i \in S \quad (7)$$

$$\sum_{j=1}^{W} t_{i,j,k} \leq t_k, \forall s_i \in S,\ k = 1 \ldots K \quad (8)$$

$$\sum_{\substack{a_m \in d_{i,j} \\ d_{i,j} \in D}} t_{i,j,k} \geq t_k, \forall a_m \in A, k = 1 \ldots K \quad (9)$$

$$\text{where } t_{i,j,k} = 0 \text{ or } t_k, \text{ and } t_k \geq 0 \quad (10)$$

Since the MDCS is NP-complete, it is unlikely to solve the MIP or LMIP problem of the MDCS in polynomial time. We relax "$t_{i,j,k} = 0$ or t_k" to "$0 \leq t_{i,j,k} \leq t_k$" in the constraint (10) and obtain the variable constraint for the LP problem:

$$t_{i,j,k} \geq 0 \quad (11)$$

We use the constraint (11) for the LP problem instead of the constraint "$0 \leq t_{i,j,k} \leq t_k$" because the later can be deduced by the two constraints (8) and (11) together. Finally, we get the LP problem consisting of the objective function (2), the constraints (7), (8), (9) and (11). In the following sections, we describe several heuristic algorithms based on the LP problem.

V. Progressive and Prog-Resd Algorithms

In this section, we present a heuristic algorithm named *Progressive* based on the LP problem. Then, two important processes in this algorithm, which are also used in the *Feedback* algorithm, are specifically described. At last, we propose an enhanced algorithm named *Prog-Resd*.

A. Progressive Algorithm

In [10], an algorithm based on LP is proposed to get the maximal lifetime of an omni-directional sensor network. In this paper, we modify this algorithm as a basic solution to the MDCS. This algorithm is referred to as *Progressive* since in each iteration it computes several cover sets and their corresponding work time which is accumulated to the total network lifetime. Each iteration in the *Progressive* algorithm consists of the following steps.

First, we solve the LP problem and get the optimal solution of t_k, the work time of the k^{th} set of directions, and $t_{i,j,k}$, the work time of the direction $d_{i,j}$ in the k^{th} set of directions, for $i = 1 \dots N$, $j = 1 \dots W$ and $k = 1 \dots K$. We denote the k^{th} set of work directions as $D_k = \{d_{i,j} | \, t_{i,j,k} > 0, \forall d_{i,j} \in D\}$.

Note that more than one direction of a sensor may be in D_k, for $k = 1 \dots K$. For example, $t_{i,j,k} > 0$ and $t_{i,j',k} > 0$ indicate that the directions $d_{i,j}$ and $d_{i,j'}$ of the same sensor s_i work at the same time, although $t_{i,j,k}$ and $t_{i,j',k}$ may still satisfy all the constraints of the LP problem. If more than one direction of a sensor is in D_k, we say these directions *conflict* with each other and are *conflicting directions*. Otherwise, if only one direction of the sensor is in D_k, we say this direction is a *non-conflicting direction*. We need to remove conflicting directions in D_k to make it a cover set. We call this process as the *conflicting direction elimination process*. If this process succeeds, it returns the updated cover set D_k and the work time $t_{i,j,k}$ of any $d_{i,j} \in D_k$; otherwise, $D_k = \emptyset$. This process is unnecessary for omni-directional sensor networks because no conflicting direction exists in omni-directional sensor networks. For description convenience, we describe the detail of this process separately in Section V-B.

If the conflicting direction elimination process returns a cover set D_k, we also need to determine the work time for D_k. Although the work time of the directions in D_k may be variant, we determine an identical period of time such that all the targets in A can be covered by the directions in D_k. To save energy, only a subset of D_k can be selected. We call this process as the *direction selection process*. This process returns a cover set $D_k^* \subseteq D_k$ and the work time t_k^* of D_k^*. We describe the detail of this process separately in Section V-C.

After the direction selection process, the work time t_k^* of the cover set D_k^* is accumulated to the total network lifetime. Then the residual lifetime of any selected sensor s_i is updated, i.e., $L_i = L_i - t_k^*$, $\forall d_{i,j} \in D_k^*$. The constraint (7) in the LP problem is also updated.

The iterations are repeated until the lifetime computed in the current iteration is less than a small positive value of ε, which is given depending on the accuracy requirement of specific applications.

The *Progressive* algorithm is shown below:

Progressive Algorithm

1: $l_{net} = 0$ /* the lifetime of the network*/
2: **repeat**
3: Solve the LP problem and get each t_k and $t_{i,j,k}$
4: $D_k = \{d_{i,j} | \, t_{i,j,k} > 0, \forall d_{i,j} \in D\}$, for $k = 1 \dots K$
5: $l_{net}^{'} = l_{net}$
6: **for** $k = 1 \dots K$
7: Call the conflicting direction elimination process to make D_k a cover set.
8: **if** $D_k \neq \emptyset$
9: Call the direction selection process to select a cover set $D_k^* \subseteq D_k$ and get its work time t_k^*
10: $l_{net} = l_{net} + t_k^*$
11: **for** each $d_{i,j} \in D_k^*$
12: $L_i = L_i - t_k^*$
13: **until** $l_{net} - l_{net}^{'} < \varepsilon$
14: **return** l_{net}

B. Conflicting Direction Elimination Process

The conflicting direction elimination process exists only in directional sensor networks, since each sensor has exactly one direction in omni-directional sensor networks. At first, we have the set D_k of the work directions and the work time $t_{i,j,k}$ for any $d_{i,j} \in D_k$. Eliminating the conflicting directions in D_k to get a cover set is an instance of the DCS, which is NP-complete. Therefore, we provide a heuristic to eliminate the conflicting directions in D_k in this subsection. Initially, we set $D_k^{'} = D_k$, $A^{'} = A$, $D_k = \emptyset$. We repeat the following steps to select non-conflicting directions from $D_k^{'}$ into D_k.

In case that there exist non-conflicting directions in $D_k^{'}$, which do not conflict with any other direction in $D_k^{'}$, we find the direction d_{i^*,j^*} with the maximal work time among all of the non-conflicting directions in $D_k^{'}$. We denote U as the set of the targets in $A^{'}$ that are covered by d_{i^*,j^*}. Remove the targets in U from $A^{'}$. After the targets in U are removed from $A^{'}$, there are some directions in $D_k^{'}$ that covers no targets in the current $A^{'}$, including the direction d_{i^*,j^*}. We denote the set of these directions as V. Remove the directions in V from $D_k^{'}$. If a direction $d_{i,j}$ in V conflicts with the directions neither in D_k nor in $D_k^{'}$, we add $d_{i,j}$ to D_k. Remove $d_{i,j}$ from V and repeat to select a new direction from V into D_k until the remaining directions in V conflict with the directions either in D_k or $D_k^{'}$. We repeat to select non-conflicting directions from $D_k^{'}$ into D_k until no non-conflicting direction exists in $D_k^{'}$.

In case that no non-conflicting direction exists in $D_k^{'}$ and $D_k^{'} \neq \emptyset$, we find the direction d_{i^*,j^*} with the maximum work time in $D_k^{'}$. Remove the directions that conflict with d_{i^*,j^*} from $D_k^{'}$.

Repeat the above steps until $D_k^{'}$ is empty. If $A^{'}$ is empty, this process succeeds and we add the work time of the removed directions to the work time of the ones in D_k. Otherwise, this process fails and let $D_k = \emptyset$. Finally, return D_k and the work time $t_{i,j,k}$ for any $d_{i,j} \in D_k$.

The conflicting direction elimination process is shown below:

Conflicting-Direction-Elimination $(D_k, \{t_{i,j,k} | \, \forall d_{i,j} \in D_k\})$

1: $D_k^{'} = D_k$, $A^{'} = A$, $D_k = \emptyset$
2: **while** $D_k^{'} \neq \emptyset$
3: **while** there exist non-conflicting directions in $D_k^{'}$
4: Find d_{i^*,j^*} with the maximal work time among the non-conflicting directions in $D_k^{'}$

$U = \{a_m |\ a_m \in d_{i^*,j^*}, \forall a_m \in A^{'}\}$, $A^{'} = A^{'} - U$
$V = D_k^{'} - \{d_{i,j} |\ a_m \in d_{i,j}, \exists a_m \in A^{'}, \forall d_{i,j} \in D_k^{'}\}$, $D_k^{'} = D_k^{'} - V$
for each $d_{i,j} \in V$
if $d_{i,j}$ conflicts with the directions neither in D_k nor $D_k^{'}$
$D_k = D_k \cup \{d_{i,j}\}$, $V = V - \{d_{i,j}\}$
if $D_k^{'} \neq \emptyset$
Find d_{i^*,j^*} with the maximum work time in $D_k^{'}$
$D_k^{'} = D_k^{'} - \{d_{i^*,j} |\ j \neq j^*, \forall d_{i^*,j} \in D_k^{'}\}$
if $A^{'}$ is empty
for each $d_{i,j} \in D_k$
$t_{i,j,k} = t_{i,j,k} + \sum_{d_{i,j'} \in D - D_k} t_{i,j',k}$
else
$D_k = \emptyset$
return D_k and $\{t_{i,j,k} | \forall d_{i,j} \in D_k\}$

We give the following two examples to see how the conflicting direction elimination process works. Example 1 shows the case that a non-conflicting direction exists while Example 2 shows the case that no non-conflicting direction exists.

Example 1. Fig. 2(a) illustrates a directional sensor network with $A = \{a_1, \ldots, a_4\}$, $S = \{s_1, \ldots, s_5\}$ and $D = \{d_{1,1}, d_{1,2}, d_{1,3}, \ldots, d_{5,1}, d_{5,2}, d_{5,3}\}$. Assume $D_k = \{d_{1,3}, d_{2,1}, d_{2,3}, d_{3,1}, d_{3,3}, d_{4,1}, d_{4,3}, d_{5,1}\}$ and the work time of the corresponding directions in D_k is $\{0.8, 0.2, 0.8, 0.2, 0.8, 0.2, 0.8, 0.2\}$, i.e., $t_{1,3,k} = 0.8$, $t_{2,1,k} = 0.2$, $t_{2,3,k} = 0.8$ and so on. The elimination process goes as follows: Get $D_k^{'} = D_k, A^{'} = A, D_k = \emptyset$. At first, there are two non-conflicting directions $d_{1,3}$ with longer work time 0.8 and $d_{5,1}$ with work time 0.2, so $d_{1,3}$ is selected. Get $U = \{a_1\}$ and then remove a_1 from $A^{'}$. Get $V = \{d_{1,3}, d_{2,1}\}$, where $d_{1,3}$ is a non-conflicting direction and $d_{2,1}$ conflicts with $d_{2,3}$. Add $d_{1,3}$ to D_k and remove both $d_{1,3}$ and $d_{2,1}$ from $D_k^{'}$. Finally, we get $D_k = \{d_{1,3}, d_{2,3}, d_{3,3}, d_{4,3}\}$ with $\{0.8, 1.0, 1.0, 1.0\}$, the work time of the corresponding directions in D_k.

Example 2. Fig. 2(b) illustrates a directional sensor network with $A = \{a_1, \ldots, a_6\}$, $S = \{s_1, \ldots, s_6\}$ and $D = \{d_{1,1}, d_{1,2}, d_{1,3}, \ldots, d_{6,1}, d_{6,2}, d_{6,3}\}$. Assume $D_k = \{d_{1,1}, d_{1,3}, d_{2,1}, d_{2,3}, d_{3,1}, d_{3,3}, d_{4,1}, d_{4,3}, d_{5,1}, d_{5,3}, d_{6,1}, d_{6,3}\}$ and the work time of each direction in D_k is $\{0.2, 0.8, 0.2, 0.8, 0.2, 0.8, 0.2, 0.8, 0.2, 0.8, 0.2, 0.8\}$, i.e., $t_{1,1,k} = 0.2$, $t_{1,3,k} = 0.8$, $t_{2,1,k} = 0.2$ and so on. The elimination process goes as follows: Get $D_k^{'} = D_k, A^{'} = A, D_k = \emptyset$. At first, there is no non-conflicting direction in $D_k^{'}$, and we select $d_{1,3}$ with its work time 0.8. In $D_k^{'}$, $d_{1,1}$ conflicts with $d_{1,3}$, so $d_{1,1}$ is removed from $D_k^{'}$. The direction $d_{1,3}$ is a non-conflicting direction in $D_k^{'}$ after $d_{1,1}$ is removed. Then we can select $d_{1,3}$ and other directions into D_k just like the way in the Example 1. Finally, we get $D_k = \{d_{1,3}, d_{2,3}, d_{3,3}, d_{4,3}, d_{5,3}, d_{6,3}\}$ with $\{1.0, 1.0, 1.0, 1.0, 1.0, 1.0\}$, the work time of the corresponding directions in D_k.

C. Direction Selection Process

At first, we have the cover set D_k and the work time $t_{i,j,k}$ for any direction $d_{i,j}$ in D_k. For a target a_m, the maximal time for which it can be covered by the directions in D_k is $t_{a_m} = \max_{\substack{a_m \in d_{i,j} \\ d_{i,j} \in D_k}} t_{i,j,k}$. The maximal time for which all the targets in A can be covered by the directions in D_k is $t_k^* = \min_{a_m \in A} t_{a_m}$. Hence, $t_k^* = \min_{a_m \in A} \max_{\substack{a_m \in d_{i,j} \\ d_{i,j} \in D_k}} t_{i,j,k}$.

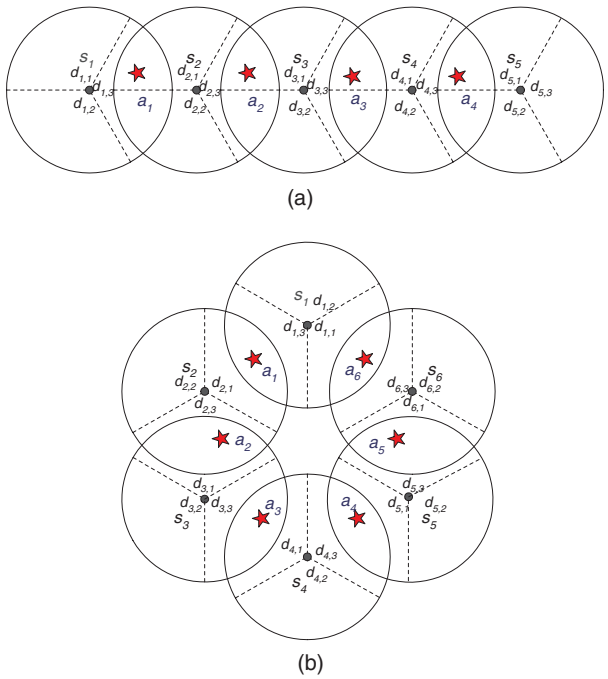

Fig. 2. Examples of conflicting direction elimination

A cover set $D_k^* \subseteq D_k$ is selected to save energy. A straightforward way is to select the direction $d_{i,j} \in D_k$ that satisfies $t_{i,j,k} > t_k^*$ and has the longest work time, to cover some uncovered targets each time. Repeat selecting another direction from D_k to D_k^* until all the targets are covered by the selected directions. Then, remove redundant directions in D_k^* because the targets covered by some directions formerly selected into D_k^* may be totally covered by the ones selected into D_k^* later, which causes redundancy. Finally, return the cover set $D_k^* \subseteq D_k$ and its work time t_k^*.

The direction selection process is shown below:

Direction-Selection $(D_k, t_k, \{t_{i,j,k} | \forall d_{i,j} \in D_k\})$

$t_k^* = \min_{a_m \in A} \max_{\substack{a_m \in d_{i,j} \\ d_{i,j} \in D_k}} t_{i,j,k}$
$D_k^* = \emptyset$, $A^{'} = A$
$D_k^{'} = \{d_{i,j} | t_{i,j,k} \geq t_k^*, \forall d_{i,j} \in D_k\}$
Sort the directions in $D_k^{'}$ according to the corresponding work time in non-increasing order
for $A^{'} \neq \emptyset$
Remove the direction $d_{i,j}$ from the head of $D_k^{'}$
if $\exists a_m \in A^{'}$, $a_m \in d_{i,j}$
$D_k^* = D_k^* \cup \{d_{i,j}\}$
$A^{'} = A^{'} - \{a_m | a_m \in d_{i,j}, \forall a_m \in A^{'}\}$
Remove redundant directions in D_k^*
return D_k^* and t_k^*

D. Prog-Resd Algorithm

Note that in the direction selection process in Section V-C, when selecting a cover set $D_k^* \subseteq D_k$ to save energy, the direction with the longest work time is selected each time. We propose an algorithm named *Prog-Resd* that takes into consideration the residual lifetime of sensors. This algorithm differentiates from the *Progressive* algorithm only in the direction selection process. In this process of the *Prog-Resd* algorithm, we get the maximal time t_k^* for which all the targets in A are covered by the directions in D_k. When selecting a cover set $D_k^* \subseteq D_k$, we select the direction $d_{i,j} \in D_k$ that satisfies $t_{i,j,k} > t_k^*$ and has the longest residual lifetime L_i to cover some uncovered targets each time. The remainder of this process is the same as the direction selection process in Section V-C.

VI. Feedback Algorithm

As we stated before, we aim to extend the network lifetime by activating a group of cover sets one after another in this paper. The number of the cover sets plays an important role when scheduling the cover sets in practice. Too many cover sets may be inefficient or impractical. Frequently switching sensors from one direction to another may not be easy for physical reasons. Furthermore, even if the state transition period, which is the time interval when one cover set is being put into sleep as well as another cover set being activated, is relatively short, too many cover sets mean too many state transition periods that lead to the occurrence of the following consequence with high probability: Some targets may not be covered during the state transition period. Therefore, an efficient algorithm should generate fewer cover sets with longer work time.

In this section, we propose an algorithm named *Feedback* that utilizes the results obtained from the previous iterations and finds a group of cover sets in the last iteration. This algorithm is more useful and practical because it generates no more than K cover sets totally. Although the cover sets generated in each iteration of the *Progressive* and the *Prog-Resd* algorithms are no more than K, the number of the total cover sets after all the iterations may be much larger than K.

In the *Feedback* algorithm, the LP problem formulated in Section IV, the conflicting direction elimination process proposed in Section V-B and the direction selection process proposed in Section V-C are also used. In each iteration of the *Feedback* algorithm, we only determine *one* cover set from the solution to the LP problem, and add the constraints that indicate this cover set to the LP problem in the next iteration. Then we solve the updated LP problem again to get the next cover set. The u^{th} iteration in the *Feedback* algorithm consists of the following steps.

At the first step, we solve the LP problem and get the optimal solution t_k and $t_{i,j,k}$, for $i = 1 \dots N$, $j = 1 \dots W$ and $k = 1 \dots K$. The set of work directions is denoted as $D_k = \{d_{i,j} | \, t_{i,j,k} > 0, \forall d_{i,j} \in D\}$, $k = 1 \dots K$. The former $u-1$ sets, D_1, D_2, $\dots$, D_{u-1}, are cover sets, and the later $K-u+1$ sets may not be cover sets. We set the collection of the later $K-u+1$ sets as $U_{nc} = \{D_k | \, k = u \dots K\}$ and the set of work time of the former $u-1$ cover sets as $V_c = \{t_k | \, k = 1 \dots u-1\}$.

At the second step, the set D_v in U_{nc} with the longest work time is selected. The conflicting directions in D_v are eliminated by the conflicting direction elimination process in Section V-B. If $D_v \neq \emptyset$, the elimination process succeeds and D_v is a cover set. Otherwise, another set in U_{nc} is tried. After the cover set D_v is found, a subset D_v^* of D_v is selected to save energy and its work time t_v^* is determined, using the direction selection process in Section V-C.

At the third step, if the cover set D_v^* with its work time t_v^* is successfully found at the second step, constraints are added to the LP problem to make the u^{th} set a cover set. A constraint $t_{i,j,u} = 0$ for each $d_{i,j} \notin D_v^*$ and a constraint $t_{i,j,u} = \delta$ for each $d_{i,j} \in D_v^*$ are added to the LP problem. The parameter δ is a quite small positive number.

The iteration consisting of the three steps above is repeated until all the K cover sets are found or no cover set can be found in the current iteration. Finally, the network lifetime is determined. In the case that K cover sets are found, we compute once again the LP problem to which we have added more constraints in the K^{th} iteration and get the work time t_k for each cover set. The network lifetime $l_{net} = \sum_{k=1}^{K} t_k$. In the case that less than K cover sets are found, the network lifetime $l_{net} = \sum_{t_k \in V_c} t_k$, where V_c is the set of the work time of all the cover sets in the last iteration.

The *Feedback* algorithm is shown below:

Feedback Algorithm

1: $u = 1$, $U_{nc} = \emptyset$, $V_c = \emptyset$
2: **while** $u \leq K$
3: Solve the LP problem and get each t_k and $t_{i,j,k}$
4: $D_k = \{d_{i,j} | \, t_{i,j,k} > 0, \forall d_{i,j} \in D\}$, $U_{nc} = \{D_k | \, k = u \dots K\}$, $V_c = \{t_k | \, k = 1 \dots u-1\}$
5: $Found =$ FALSE
6: **while** $Found ==$ FALSE
7: Select a D_v such that $t_v = \max_{D_k \in U_{nc}} t_k$
8: $U_{nc} = U_{nc} - D_v$
9: Call the conflicting direction elimination process to make D_v a cover set
10: **if** $D_v \neq \emptyset$
11: $Found =$ TRUE
12: Call the direction selection process to select a cover set $D_v^* \in D_v$ and get its work time t_v^*
13: **if** $Found ==$ TRUE
14: **for** each $d_{i,j} \in D - D_v^*$
15: Add $t_{i,j,u} = 0$ to the LP problem
16: **for** each $d_{i,j} \in D_v^*$
17: Add $t_{i,j,u} = \delta$ to the LP problem
18: $u = u + 1$
19: **else**
20: **break**
21: **if** $u > K$
22: Solve the LP problem and get each t_k
23: $l_{net} = \sum_{k=1}^{K} t_k$

24: **else**
25: $l_{net} = \sum_{t_k \in V_c} t_k$
26: **return** l_{net}

Here, we analyze the time complexity of the three algorithms. Consider a directional sensor network with M targets and N sensors, each of which has W directions. K is the maximal number of cover sets in each iteration. In each iteration of the three algorithms, *Progressive*, *Prog-Resd* and *Feedback*, the LP problem is solved once. The time complexity of the LP problem is $O(n^3)$ using Ye's algorithm [17], where n is the number of variables and $n = K + KNW$. The time complexity to get K cover sets in each iteration is $O(KN^3WM)$. As $KN^3WM \ll K^3N^3W^3$ in practice, the time complexity of each iteration is $O(K^3N^3W^3)$, which is mainly determined by the time complexity of the LP problem. The number of iterations in the *Feedback* algorithm is at most K, while the LP problem is solved for at most $K + 1$ times. Therefore, the total time complexity of the *Feedback* algorithm is $O(K^4N^3W^3)$. In the *Progressive* and *Prog-Resd* algorithms, it is complicated to determine the number of iterations because it depends on the many parameters, such as K and ε.

VII. Simulation Results

We evaluate the performance of the three algorithms *Progressive*, *Prog-Resd* and *Feedback* through simulations running on a computer with 3 GHz CPU and 1 GB memory. The optimization toolbox in Matlab is used to solve the LP problem. N sensors with sensing radius r and M targets are deployed uniformly in a region of $400m \times 400m$. Each sensor has W directions. The maximal number of cover sets in one iteration is equal to the number of sensors, i.e., $K = N$. For the *Progressive* algorithm and the *Prog-Resd* algorithm, we set $\varepsilon = 0.001$. For the *Feedback* algorithm, we set $\delta = 0.0001$. Each algorithm runs 10 times through random placement of sensors and targets.

1) Network Lifetime: In this subsection, the initial lifetime of each sensor is set as 1 and W is set as 3.

Fig. 3 shows the relationship between the network lifetime and the number of sensors when 10 targets are deployed and r is fixed at 100. The network lifetime increases almost linearly when the number of sensors increases. The *Prog-Resd* algorithm works a little better than the *Progressive* algorithm in this figure. The *Feedback* algorithm has the best performance compared to the other two algorithms. When the number of sensors is 80, the average network lifetime of the *Feedback* algorithm is 8.493, while they are 7.018 and 6.313 for the *Prog-Resd* algorithm and the *Progressive* algorithm respectively. The relationship between the network lifetime and the sensing radius is shown in Fig. 4 when 50 sensors and 10 targets are deployed. The network lifetime also increases almost linearly when the sensing radius increases.

We fix $N = 50$ and $r = 100$. Fig. 5 shows that the network lifetime drops as the number of targets increases. We can see that the network lifetime drops quickly when M varies from 1 to 2, and then drops relatively slowly when M varies from 5 up to 20.

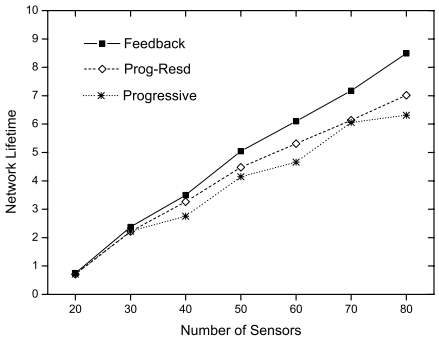

Fig. 3. Network lifetime vs. number of sensors N with $M = 10$, $r = 100$ and $W = 3$

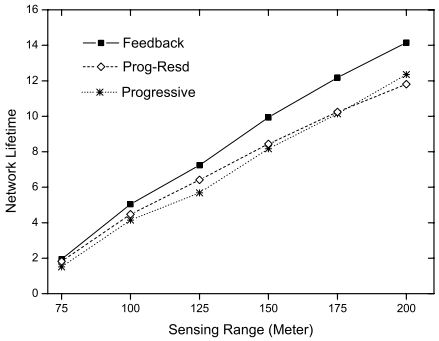

Fig. 4. Network lifetime vs. sensing radius r with $M = 10$, $N = 50$ and $W = 3$

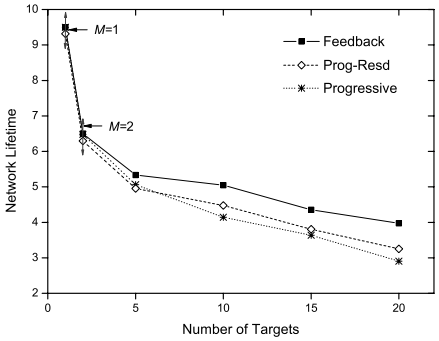

Fig. 5. Network lifetime vs. number of targets M with $N = 50$, $r = 100$ and $W = 3$

2) Number of Directions per Sensor: An omni-directional sensor network is a special case of directional sensor networks when $W = 1$. For simplicity, we assume the lifetime of a sensor with W directions is W here. Fig. 6 shows the growth of the network lifetime with the increase of W when 50 sensors and 10 targets are deployed. We can see from the figure that the network lifetime is almost linear to W.

3) Runtime: Fig. 7 shows the runtime for the three algorithms with 10 targets, $r = 100$ and $W = 3$. As the number of sensors increases, the runtime increases. The runtime of the *Prog-Resd* algorithm is slightly longer than the *Progressive* algorithm. The runtime of the *Feedback* algorithm is longer than the other two algorithms. We observe one fact that the *Feedback* algorithm runs mostly K iterations, while the *Prog-*

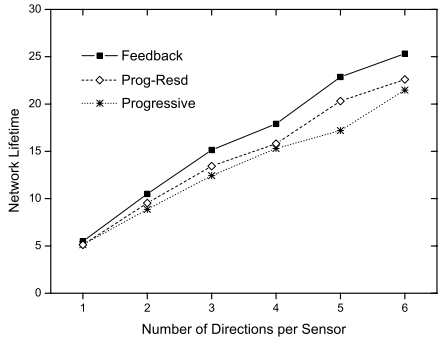

Fig. 6. Network lifetime vs. number of directions per sensor W with $M = 10$, $N = 50$ and $r = 100$

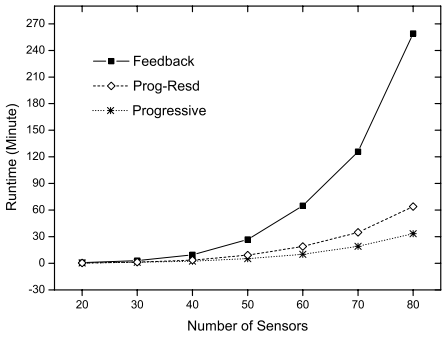

Fig. 7. Runtime vs. number of sensors N with $M = 10$, $r = 100$ and $W = 3$

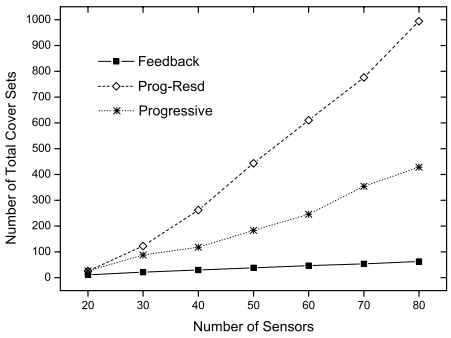

Fig. 8. Number of total cover sets vs. number of sensors N with $M = 10$, $r = 100$ and $W = 3$

Resd and the *Progressive* algorithms run much less than K iterations in most cases.

4) Number of Total Cover Sets: Fig. 8 shows the number of the total cover sets of each algorithm with 10 targets, $r = 100$ and $W = 3$. We can see from this figure that both the *Progressive* algorithm and the *Prog-Resd* algorithm generate much more cover sets than the *Feedback* algorithm. As we analyzed before, fewer cover sets with longer work time are more efficient and practical.

VIII. Conclusions and Future Work

Scheduling algorithms to save energy and prolong the network lifetime are always important for sensor networks. However, algorithms designed for omni-directional sensor networks may not be suitable for directional sensor networks. In this paper, we have studied the problem of multiple directional cover sets and proved that this problem is NP-complete. We have presented the *Progressive* algorithm based on LP, and then enhanced this algorithm to gain a longer network lifetime. Also, the *Feedback* algorithm using the previous results as a feedback has been described. Simulation results show that the *Feedback* algorithm gets a longer network lifetime and fewer cover sets that are more efficient in practice. As a future work, we plan to design distributed algorithms to prolong the network lifetime of a directional sensor network.

Acknowledgment

We would like to thank the anonymous reviewers for their valuable comments and suggestions. This work is supported in part by grants HKPU A-PG53, A-PH12, Z09M, PolyU-5236/06E, NSFC 60473092, NSFC 90612018, and the National Basic Research Program of China (973 Program) under Grant No. 2006CB303000.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *ACM Trans. on Multimedia Computing, Communications and Applications*, pp. 102–114, Aug 2002.

[2] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler, "An analysis of a large scale habitat monitoring application," in *ACM Conference on Embedded Networked Sensor Systems(SenSys)*, pp. 214–226, 2004.

[3] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin, and M. Srivastava, "Cyclops: In situ image sensing and interpretation in wireless sensor networks," in *SenSys*, pp. 192–204, 2005.

[4] W. Feng, E. Kaiser, W. C. Feng, and M. L. Baillif, "Panoptes: Scalable low-power video sensor networking technologies," *ACM Trans. on Multimedia Computing, Communications and Applications*, vol. 1, no. 2, pp. 151–167, May 2005.

[5] J. Djugash, S. Singh, G. Kantor, and W. Zhang, "Range-only slam for robots operating cooperatively with sensor networks," in *IEEE International Conference on Robotics and Automation*, 2006.

[6] F. Ye, G. Zhong, J. Cheng, S. Lu, and L. Zhang, "PEAS: a robust energy conserving protocol for long-lived sensor networks," in *IEEE International Conference on Network Protocols (ICNP)*, 2002.

[7] C. Gui and P. Mohapatra, "Power conservation and quality of surveillance in target tracking sensor networks," in *ACM MobiCom*, 2004.

[8] Y. Cai, M. Li, W. Shu, and M. Wu, "ACOS: A precise energy-aware coverage control protocol for wireless sensor networks," in *International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, 2005.

[9] H. Zhang and J. C. Hou, "Maintaining sensing coverage and connectivity in large sensor networks," *Ad Hoc & Sensor Wireless Networks, An International Journal*, 2005.

[10] M. Cardei, M. T. Thai, Y. Li, and W. Wu, "Energy-efficient target coverage in wireless sensor networks," in *IEEE INFOCOM*, 2005.

[11] H. Liu, P. Wan, C. Yi, X. Jia, S. Makki, and P. Niki, "Maximal lifetime scheduling in sensor surveillance networks," in *IEEE INFOCOM*, 2005.

[12] M. X. Cheng, L. Ruan, and W. Wu, "Achieving minimum coverage breach under bandwidth constraints in wireless sensor networks," in *IEEE INFOCOM*, 2005.

[13] H. Ma and Y. Liu, "On coverage problems of directional sensor networks," in *MSN*, 2005.

[14] J. Ai and A. A. Abouzeid, "Coverage by directional sensors in randomly deployed wireless sensor networks," *Journal of Combinatorial Optimization*, vol. 11, no. 1, pp. 21–41, Feb. 2006.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms, Second Edition*. MIT Press, 2001.

[16] L. A. Wolsey and G. L. Nemhauser, *Integer and Combinatorial Optimization*. Wiley, 1999.

[17] Y. Ye, "An $O(n^3L)$ potential reduction algorithm for linear programming," *Mathematical Programming*, vol. 50, pp. 239–258, 1991.