Energy Efficient Target-Oriented Scheduling in Directional Sensor Networks

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

Abstract—Unlike convectional omnidirectional sensors that always have an omniangle of sensing range, *directional sensors* may have a limited angle of sensing range due to the 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 (MDCS) problem* of organizing the directions of sensors into a group of nondisjoint 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 several algorithms for the MDCS. Simulation results are presented to demonstrate the performance of these algorithms.

Index Terms—Coverage, energy conservation, mixed integer programming, scheduling, sensor networks.

1 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 is always based on the assumption of *omnidirectional sensors* that have an omniangle of sensing range. However, sensors may have a limited angle of sensing range due to the technical constraints or cost considerations, which are denoted by *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.

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 that 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 overlap 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 omnidirectional 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 (DCS) problem*, is proved to be NP-complete in this paper.

- *Y. Cai is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. E-mail: caiyanli@gmail.com.*
- *W. Lou is with the Department of Computing, The Hong Kong Polytechnic University, PQ 705, Hong Hum, Kowloon, Hong Kong. E-mail: csweilou@comp.polyu.edu.hk.*
- *M. Li is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Dongchuan Rd 800, Shanghai, China. E-mail: li-ml@cs.sjtu.edu.cn.*
- *X.-Y. Li is with the Department of Computer Science, Illinois Institute of Technology, 10, West 31st Street, Chicago, IL 60616. E-mail: xli@cs.iit.edu.*

Manuscript received 30 Apr. 2007; revised 10 July 2008; accepted 28 Jan. 2009; published online 26 Feb. 2009.
Recommended for acceptance by S. Nikoletseas.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-04-0147.
Digital Object Identifier no. 10.1109/TC.2009.40.

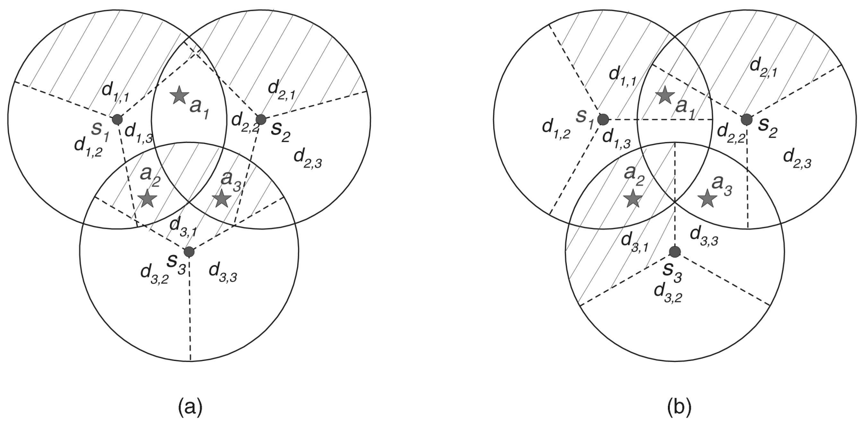


Fig. 1. Simple directional sensor networks.

Fig. 1a 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. 1a. 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 nonrechargeable. Also, the batteries of the sensors are hard to replace due to the hostile or inaccessible environments in many scenarios. We assume that each sensor is nonrechargeable 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.

The objective of this paper 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 nondisjoint subsets, each of which is a cover set, and allocate the work time for each cover set. Note that nondisjoint 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 nondisjoint cover sets and allocating the work time for each of them to maximize the network lifetime as *multiple directional cover sets (MDCS) problem*.

In this paper, we formally define the DCS and the MDCS and prove that both problems are NP-complete. We model the MDCS as an optimization problem [6]. To solve the MDCS, we first consider the solutions to the DCS, which is a subclass of the MDCS, where the number of cover sets is restricted to 1. The main contributions of this paper are as follows. We design several algorithms to meet different application requirements for the MDCS. First, we present a heuristic algorithm named *Progressive*, which is based on the optimization problem. Second, we propose an algorithm called *Feedback* that uses the results obtained in previous iterations as a feedback to the next iteration. This algorithm gets a longer network lifetime and fewer cover sets, which are more efficient and practical. Third, we describe an algorithm named *MDCS-Greedy* that is not based on the optimization problem, which has much shorter runtime. Finally, a distributed algorithm called *MDCS-Dist* is presented.

The rest of the paper is organized as follows: Section 2 briefly surveys the related works in the literature. In Section 3, the DCS and the MDCS are formally defined and proved to be NP-complete. In Section 4, we formulate the MDCS as an optimization problem. In Section 5, we describe and evaluate the solutions to the DCS. In Section 6, we present the solutions to the MDCS. In Section 7, we present the simulation results for the MDCS. The paper is concluded in Section 8.

2 Related Work

A number of scheduling algorithms have been proposed to prolong the network lifetime for omnidirectional sensor networks. Sleeping protocols, such as RIS [7], [8], PEAS [9], and PECAS [8], 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. In [10], [11], [12], both area coverage and communication connectivity are considered in the scheduling algorithms for omnidirectional sensor networks. If the communication radius is at least twice of the sensing radius, complete area coverage of a convex region implies communication connectivity among the active sensors [10], [11].

When a set of targets is deployed to be monitored by omnidirectional sensor networks, scheduling problems are studied in [13], [14], [15]. Liu et al. [13] assume that a sensor can watch only one target at a time and build a target watching timetable for each sensor to maximize the network lifetime. Cheng et al. [14] organize 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 the authors of [13] and [14], Cardei et al. [15] aim to extend the lifetime of an omnidirectional sensor network by organizing the sensors into nondisjoint subsets, where each target must be covered by at least one sensor in each subset. This problem is proved to be NP-complete in [15], although finding a subset of omnidirectional sensors to cover all the targets can be done in a polynomial time. Note that the problem discussed in [15] is a special case of the MDCS, where a sensor has only one direction.

Some efforts have recently been devoted to the research of the directional sensor networks. Ma and Liu [16] provide a directional sensor model, where each sensor is fixed to one direction and analyzes the probability of full area coverage. In [17], a similar directional sensor model is proposed, where a sensor is allowed to work in several directions. The objective 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 nondisjoint cover sets in each of which the directions cover all the targets so as to maximize the network lifetime.

3 Problem Statement

In this section, we first define the notations, and then 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.

3.1 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 mth target, $1 \leq m \leq M$.
- s_i: the ith sensor, $1 \leq i \leq N$.
- $d_{i,j}$: the jth direction of the ith 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 $s_i = \{d_{i,j} | \, j = 1 \dots W\}$. Hence, if $a_m \in d_{i,j}, a_m$ is covered by $d_{i,j}$.
- A: the set of targets. $A = \{a_1, a_2, \dots, a_M\}$.
- S: the set of sensors. $S = \{s_1, s_2, \dots, s_N\}$.
- D: the set of the directions of all the sensors. $D = \{d_{i,j} | \, i = 1 \dots N, j = 1 \dots W\}$.
- 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.

For simplicity, we assume that 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.

3.2 Simple Examples of the MDCS

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. 1a, we can get the following cover sets: $D_1 = \{d_{1,3}, d_{3,1}\}$ with the work time of 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. 1a, 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. 1b, 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.

3.3 Problem Definition

To prove the NP-completeness of the DCS and 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 collection S of subsets 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. *DCS Problem: Given a collection D of subsets of a finite set A and a collection S of subsets of D, find a cover set for A.*

Definition 3. *MDCS Problem: Given a collection D of subsets of a finite set A and a collection S of subsets of D, find a family of K cover sets $D_1, D_2, \dots, D_K \subseteq D$ for A, with nonnegative weights $t_1, t_2, \dots, t_K$, such that $t_1 + t_2 + \dots + t_K$ 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.

3.4 NP-Completeness

In this section, we first prove the DCS to be NP-complete by reduction from the 3-Conjunctive Normal Form Satisfiability (*3-CNF-SAT*) problem [18]. 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 collection S of subsets 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 collection S of subsets of D, determine if there exists a family of K cover sets $D_1, D_2, \dots, D_K \subseteq D$ for A, with nonnegative weights $t_1, t_2, \dots, t_K$, such that $t_1 + t_2 + \dots + t_K \geq p$, 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 DCS $\in$ NP. Suppose that a set D' is given as a certificate. The verification algorithm first affirms $D' \subseteq D$ and then checks that if each element in A belongs to at least one member of D'. Finally, 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, DCS $\in$ NP.

To prove that the decision version of the DCS is NP-hard, we show a polynomial time reduction from

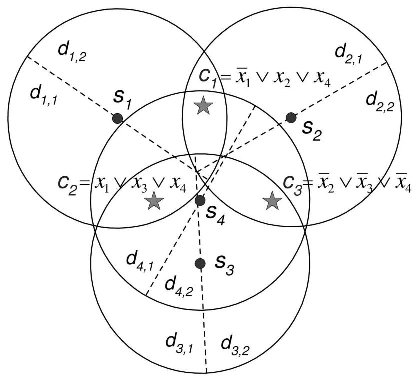

$d_{1,2}$
$d_{1,1}$
s_1
$c_1 = \overline{x}_1 \vee x_2 \vee x_4$
$d_{2,1}$
$d_{2,2}$
s_2
$c_2 = x_1 \vee x_3 \vee x_4$
$c_3 = \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4$
s_4
$d_{4,1}$
s_3
$d_{4,2}$
$d_{3,1}$
$d_{3,2}$

Fig. 2. An example of the reduction from the *3-CNF-SAT* problem to the DCS. The formula in the *3-CNF-SAT* problem is $F = c_1 \wedge c_2 \wedge c_3$, where $c_1 = (\overline{x}_1 \vee x_2 \vee x_4), c_2 = (x_1 \vee x_3 \vee x_4)$, and $c_3 = (\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4)$. In the instance of the DCS, there are three targets c_1, c_2, and c_3 and four sensors s_1, s_2, s_3, and s_4, each of which has two directions. Direction $d_{1,1}$ of s_1 corresponds to $x_1, d_{1,2}$ of s_1 corresponds to $\overline{x}_1$, and so on. A satisfying assignment of F is $x_1 = 1, x_2 = 1, x_3 = 0$, and $x_4 = 0$. The corresponding directions of this assignment $d_{1,1}, d_{2,1}, d_{3,2}$, and $d_{4,2}$ form a cover set for the targets c_1, c_2, and c_3.

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. An example of the reduction is illustrated in Fig. 2.

We now show that the formula F is satisfiable if and only if 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* problem 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* problem, 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 $\text{MDCS} \in \text{NP}$. 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 $t_1 + t_2 + \cdots + t_K \geq p$ 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, $\text{MDCS} \in \text{NP}$.

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 $\text{MDCS} \in \text{NP}$, the MDCS is NP-complete. □

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

4 Optimization Formulation of the MDCS

In this section, we first model the MDCS as a Mixed Integer Programming (MIP) problem [6]. 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 kth cover set is denoted byD_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

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

The MIP problem formulated for the MDCS is as follows:

$$\max t_1 + t_2 + \cdots + t_K \tag{2}$$

subject to

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

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

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

$$\text{where } x_{i,j,k} = \{0, \; 1\} \text{ and } t_k \geq 0. \tag{6}$$

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 kth 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:

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

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

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

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

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:

$$t_{i,j,k} \geq 0. \tag{11}$$

We use the constraint (11) for the LP problem instead of the constraint "$0 \leq t_{i,j,k} \leq t_k$" because the latter 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 first consider the solutions to the DCS, which is a subclass of the MDCS, and then describe several heuristic algorithms to the MDCS based on the LP problem and the solutions to the DCS.

5 Solutions to the DCS

As stated before, the DCS is a subclass of the MDCS, where the number of cover sets K is restricted to 1. To solve the MDCS, we first consider the solutions to the DCS. In this section, we first present a search algorithm named *DCS-Search* to the DCS. Although we attempt to speed up the search process in this algorithm, it may still take too long runtime for some large-scale directional sensor networks. Based on the *DCS-Search* algorithm, we propose a greedy algorithm named *DCS-Greedy*, which has much shorter runtime while maintaining high possibility to find a cover set. In Section 6, the *DCS-Greedy* algorithm is applied in several solutions to the MDCS.

5.1 DCS-Search Algorithm

In this section, we propose a search algorithm called *DCS-Search*. Given a directional sensor network with a set A of M targets, a set S of N sensors, and a set D of directions. We define a tuple $G = (D_G, A_G)$, where A_G is the set of targets and $A_G = A$ initially, and D_G is the set of directions that cover at least one target in A_G, i.e., $D_G = \{d_{i,j} | \ a_m \in d_{i,j}, \exists a_m \in A, \forall d_{i,j} \in D\}$. If more than one direction of a sensor is in D_G, we say that these directions *conflict* with each other and are *conflicting directions*. Otherwise, if only one direction of the sensor is in D_G, we say that this direction is a *nonconflicting direction*. For example, the directions $d_{i,j}$ and $d_{i,j'}$ of the same sensor s_i conflict with each other if they are both in D_G. We need to select a set of nonconflicting directions from D_G to be a cover set. We denote D_s as such a selected set of directions. We also define a stack R_s to store the states of the search process.

In the *DCS-Search* algorithm, we consider the following cases, Case 1, Case 2, and Case 3, to speed up the search process when selecting directions from D_G to D_s. For each case, we specify a *pivot policy* to pick a direction among the candidate directions in D_G that satisfy this case. The pivot policy we use here is to find a direction to cover the target that can be covered by minimal number of directions. Other pivot policies can also be adopted according to the specific application requirements. In Sections 6.1.1 and 6.3, some other pivot policies are used, including selecting a direction of the sensor that has the longest residual lifetime.

Case 1. *Each target in A_G is covered by at least one direction in D_G and there exist nonconflicting directions in D_G.*

We handle this case as the following to select nonconflicting directions into D_s. Pick a nonconflicting direction d_{i^*,j^*} in D_G using the pivot policy. We denote U as the set of targets in A_G that is covered by d_{i^*,j^*}. Remove the targets in U from A_G. After the targets in U are removed from A_G, there are some directions in D_G that cover no targets in the current A_G, including the direction d_{i^*,j^*}. We denote the set of these directions by V. Remove the directions in V from D_G. If a direction $d_{i,j}$ in V conflicts with the directions neither in D_s nor in D_G, we add $d_{i,j}$ to D_s. Remove $d_{i,j}$ from V and repeat to select a new direction from V into D_s until the remaining directions in V conflict with the directions either in D_s or D_G.

Case 2. *Each target in A_G is covered by at least one direction in D_G and no nonconflicting direction exists in D_G.*

We handle this case as the following to select a direction and remove its conflicting directions from D_G. Apply the pivot policy to select a direction d_{i^*,j^*}. Record the current state of the search process, denoted by $R = (G, D_s, s_{i^*}, D_{i^*})$, where $D_{i^*} = \{d_{i^*,j^*}\}$. Push R into R_s. Remove the directions that conflict with d_{i^*,j^*} from D_G.

Case 3. *There exist some targets in A_G that are not covered by any direction in D_G.*

We handle this case as the following to backtrack. Pop the previous state of the search process $R = (G, D_s, s_{i^*}, D_{i^*})$

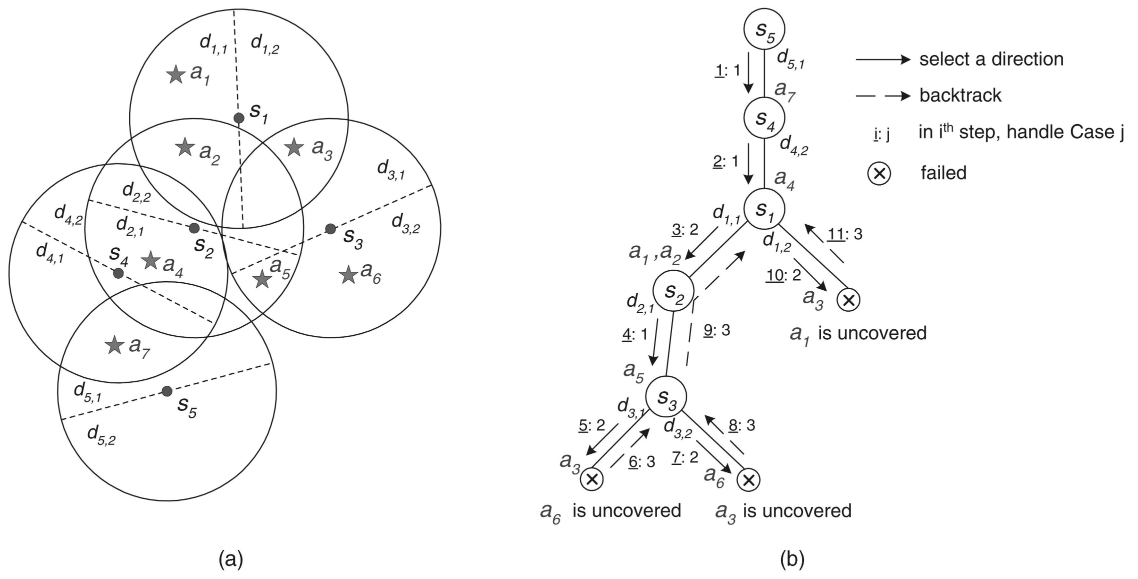


Fig. 3. An example of the *DCS-Search* algorithm.

from R_s and restore it. Try another unselected direction $d_{i^*,j'}$ of sensor s_{i^*}, i.e., $d_{i^*,j'} \in D_G$ and $d_{i^*,j'} \notin D_{i^*}$. Update D_{i^*} as $D_{i^*} \bigcup \{d_{i^*,j'}\}$ and push R back to R_s. If such a direction does not exist, backtrack again. The backtracking process is a recursive process.

The search process of the *DCS-Search* algorithm works as follows. First, while Case 1 is satisfiable, repeat to use the pivot policy to select nonconflicting directions into D_s. Second, if Case 2 is satisfiable, use the pivot policy to select one direction and remove its conflicting directions. Third, if neither Case 1 nor Case 2 is satisfiable, handle Case 3 to call the backtracking process. If the backtracking process fails, the whole search process ends and returns an empty set; otherwise, repeat the above steps until A_G is empty and return D_s as a cover set, which we will prove in Theorem 3. An example of the search process is illustrated in Fig. 3.

Theorem 3 shows that if there exists a cover set in a directional sensor network, the *DCS-Search* algorithm can succeed to find a cover set.

Theorem 3. *Consider a directional sensor network with a set A of M targets, a set S of N sensors, and a set D of directions. Suppose that there exists a cover set, which is a subset of D, covering all the targets in A. The DCS-Search algorithm returns a cover set D_s for A.*

Proof. The proof of this theorem is shown in Appendix A. □

We give the following example to illustrate how the search process works.

Example. Fig. 3a shows a directional sensor network of seven targets $a_1, a_2, \ldots, a_7$ and five sensors $s_1, s_2, \ldots, s_5$, each of which has two directions. Fig. 3b shows the search process. In Fig. 3b, a direction $d_{i,j}$ is the direction selected in the corresponding step, and the targets next to it are the targets removed from A_G in this step. At first, $d_{5,1}$ is a nonconflicting direction and Case 1 is satisfiable. Select $d_{5,1}$ into D_s. After removing a_7 from A_G and $d_{4,1}$ from D_G, $d_{4,2}$ becomes a nonconflicting direction and we handle Case 1 again. Then, Case 2 is satisfiable, $d_{1,1}$ is selected, and we record the current state of the search process. Remove $d_{1,2}$ that conflicts with $d_{1,1}$ from A_G. Handling Case 2 results in $d_{1,1}$ as a nonconflicting direction. Repeat to handle Case 1 until s_3 is reached and Case 2 is satisfiable. Select $d_{3,1}$ into D_s and record the current state of the search process. Removing $d_{3,2}$ results in a_6 uncovered. Backtracking to try the other direction $d_{3,2}$ of s_3 results in a_3 uncovered. Thus, backtrack to try the other direction $d_{1,2}$ of s_1. Selecting $d_{1,2}$ into D_s results in a_1 uncovered. At last, no backtracking is available. The whole process fails and no cover set is found. From this example, we can see that we do not need to backtrack to try every sensor even though a sensor initially has conflicting directions, such as s_2 and s_4.

The *DCS-Search* algorithm and the backtracking process are shown below.

DCS-Search Algorithm

```
1:  $A_G = A, D_G = \{d_{i,j}|\ a_m \in d_{i,j}, \exists a_m \in A, \forall d_{i,j} \in D\}$,
    $G = (D_G, A_G), D_s = \emptyset, R_s = \emptyset$
2:  while $A_G \neq \emptyset$
3:     while Case 1 is satisfiable
4:        Pick a non-conflicting direction $d_{i^*,j^*}$ in $D_G$ using
          the pivot policy
5:        $U = \{a_m|\ a_m \in d_{i^*,j^*}, \forall a_m \in A_G\}, A_G = A_G - U$
6:        $V = D_G - \{d_{i,j}|\ a_m \in d_{i,j}, \exists a_m \in A_G, \forall d_{i,j} \in D_G\}$,
          $D_G = D_G - V$
7:        for each $d_{i,j} \in V$
8:           if $d_{i,j}$ conflicts with the directions neither in $D_s$
             nor $D_G$
9:              $D_s = D_s \cup \{d_{i,j}\}$
10:    if $A_G \neq \emptyset$
11:       if Case 2 is satisfiable
12:       Pick $d_{i^*,j^*}$ in $D_G$ using the pivot policy
13:       Record the current state of the search process
          $R = (G, D_s, s_{i^*}, D_{i^*})$, where $D_{i^*} = \{d_{i^*,j^*}\}$, and push
        $R$ into $R_s$
```

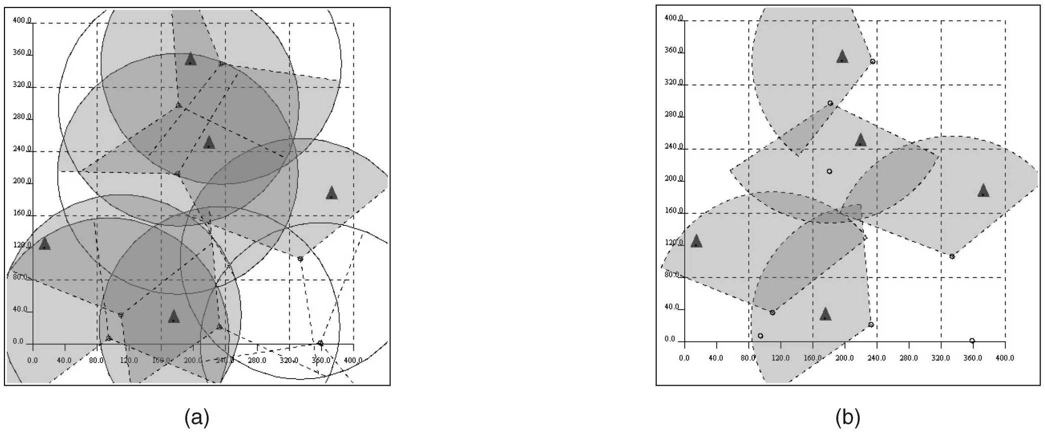


Fig. 4. Snapshots of simulations to find a cover set. The small solid triangles are targets. (a) A randomly generated directional sensor network. (b) A cover set for all the targets.

```
14:     $D_G = D_G - \{d_{i^*,j} | \ j \neq j^*, \forall d_{i^*,j} \in D_G\}$
15:  else if Backtracking-Process ($R_s$) == FALSE
16:     $D_s = \emptyset$
17:     break
18: return $D_s$
```

Backtracking-Process (R_s)

```
1: $Succeeded =$ TRUE
2: if $R_s \neq \emptyset$
3:    Pop $R = (G, D_s, s_{i^*}, D_{i^*})$ from $R_s$ and restore it
4:    if $\exists d_{i^*,j} \in D_G$ and $d_{i^*,j} \notin D_{i^*}$
5:       Pick an unselected direction $d_{i^*,j'} \in D_G$ using the
         pivot policy
6:       Set $D_{i^*} = D_{i^*} \bigcup \{d_{i^*,j'}\}$ and push updated $R$ back
         to $R_s$.
7:       $D_G = D_G - \{d_{i^*,j} | \ j \neq j', \forall d_{i^*,j} \in D_G\}$
8:    else
9:       Backtracking-Process ($R_s$)
10: else
11:    $Succeeded =$ FALSE
12: return $Succeeded$
```

5.2 DCS-Greedy Algorithm

Although we attempt to speed up the search process by reducing the times of backtracking in the *DCS-Search* algorithm, it may still take too long runtime for some large-scale directional sensor networks. In this section, we propose a greedy algorithm named *DCS-Greedy* based on the *DCS-Search* algorithm, which is suitable for large-scale directional sensor networks.

As the backtracking process in the *DCS-Search* algorithm may take most of the runtime, it is not allowed in the *DCS-Greedy* algorithm. Therefore, the *DCS-Greedy* algorithm deals with the following two cases:

Case 1. *There exist nonconflicting directions in* D_G.

Case 2. *No nonconflicting direction exists in* D_G *and* $D_G \neq \emptyset$.

We handle these two cases just as we deal with Cases 1 and 2 in the *DCS-Search* algorithm. Note that the pivot policy used in this algorithm is the same as the one in the *DCS-Search* algorithm, i.e., among all the candidate directions, it is to select one direction to cover the target that can be covered by minimal number of directions. The search process of this algorithm is almost the same as the *DCS-Search* algorithm except that there is no backtracking process. The *DCS-Greedy* algorithm works as follows. First, we construct $G = (D_G, A_G)$. Then repeat to handle Cases 1 and 2 and get a set of directions D_s. When A_G is empty, the search process succeeds to find a cover set D_s. When D_G is empty but A_G is not empty, the search process fails to find a cover set and returns an empty set.

The *DCS-Greedy* algorithm is shown below.

DCS-Greedy algorithm

```
1: $A_G = A, D_G = \{d_{i,j} | \ a_m \in d_{i,j}, \exists a_m \in A, \forall d_{i,j} \in D\}$,
   $G = (D_G, A_G), D_s = \emptyset$
2: while $D_G \neq \emptyset$
3:   while Case 1 is satisfiable
4:      Handle this case just as handling Case 1 in the
        DCS-Search algorithm
5:   if $D_G \neq \emptyset$
6:      if Case 2 is satisfiable
7:         Handle this case just as handling Case 2 in the
           DCS-Search algorithm
8: if $A_G \neq \emptyset$
9:    $D_s = \emptyset$
10: return $D_s$
```

5.3 Simulation Results

We evaluate the performance of the *DCS-Search* and *DCS-Greedy* algorithms through simulations running on a computer with 3 GHz CPU and 1 GB memory. N sensors with the sensing radius r and M targets are deployed uniformly in a region of $400 \text{ m} \times 400 \text{ m}$. Each sensor has W directions. We randomly generate 1,000 deployments of sensors and targets, and average the result on every deployment for each algorithm.

Fig. 4 shows the snapshots of the simulations. Fig. 4a illustrates a randomly generated directional sensor network when $M = 5, N = 8, r = 150$, and $W = 3$. Fig. 4b shows a cover set, where only five shadowed sectors are the work directions of the sensors.

Fig. 5 shows the relationship between the runtime and the number of directions per sensor W when $M = 40, N = 40,$

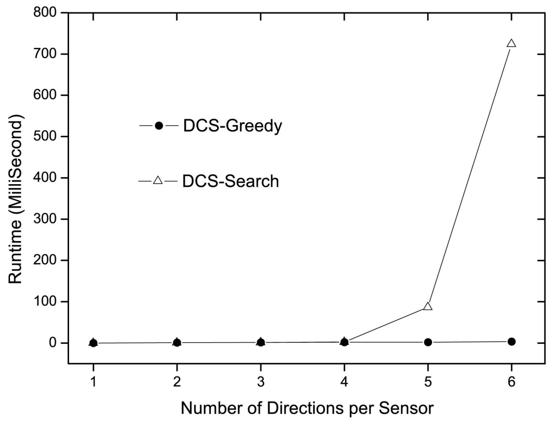


Fig. 5. Runtime versus number of directions per sensor W with $M = 40, N = 40$, and $r = 100$.

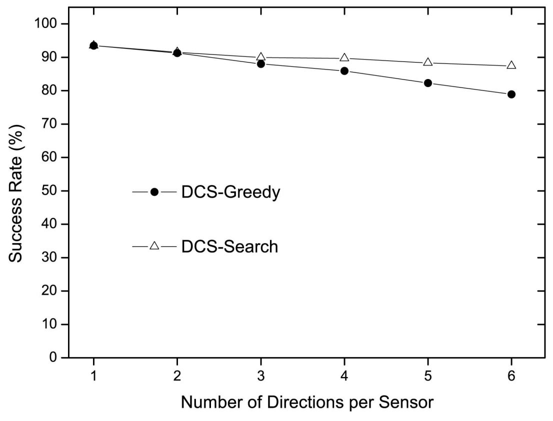


Fig. 6. Success rate versus number of directions per sensor W with $M = 40, N = 40$, and $r = 100$.

and $r = 100$. In the figure, we can see that the runtime of the *DCS-Search* algorithm grows exponentially as W increases. However, the runtime of the *DCS-Greedy* algorithm is much shorter. When $W = 6$, the average runtime of the *DCS-Search* algorithm is 724.041 millisecond, while the average runtime of the *DCS-Greedy* algorithm is only 3.298 millisecond.

Fig. 6 shows the relationship between the success rate and the number of directions per sensor W when $M = 40$, $N = 40$, and $r = 100$. The success rate is the ratio of the number of samples, where a cover set is successfully found to the total number of samples. For both algorithms, the success rate drops when W increases. The success rate of the *DCS-Greedy* algorithm decreases a little faster than the *DCS-Search* algorithm. However, the *DCS-Greedy* algorithm still maintains relatively high success rate even when $W = 6$.

Simulation results show that the runtime of the *DCS-Search* algorithm grows exponentially as W increases, while the runtime of the *DCS-Greedy* algorithm is not sensitive to W. The success rate of the *DCS-Greedy* algorithm still maintains relatively high when W increases. Therefore, in the following section, we propose several solutions to the MDCS based on the *DCS-Greedy* algorithm.

6 Solutions to the MDCS

In this section, we propose several algorithms for the MDCS. First, we present a heuristic algorithm named *Progressive* based on the LP problem as a basic solution to the MDCS. Second, we propose an algorithm called *Feedback* that gets a longer network lifetime and fewer cover sets, which are more efficient and practical. Third, we describe an algorithm named *MDCS-Greedy* without LP, which has shorter runtime. Finally, a distributed algorithm called *MDCS-Dist* is presented.

6.1 Progressive Algorithm

In [15], an algorithm based on LP is proposed to get the maximal lifetime of an omnidirectional 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 that 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 kth set of directions, and $t_{i,j,k}$, the work time of the direction $d_{i,j}$ in the kth set of directions for $i = 1 \dots N, j = 1 \dots W$, and $k = 1 \dots K$. We denote the kth set of work directions by $D_k = \{d_{i,j} | \; t_{i,j,k} > 0, \forall d_{i,j} \in D\}$.

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. 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, $D_k = \emptyset$. For description convenience, we describe the detail of this process separately in Section 6.1.1.

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 6.1.2.

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 $k = 1 \ldots K$
 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 $l_{net}$
```

6.1.1 Conflicting Direction Elimination Process

First, we have the set D_k 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. In this section, we describe how to eliminate the conflicting directions in D_k based on the *DCS-Greedy* algorithm in Section 5.2.

In this process, we first construct the tuple $G = (D_G, A_G)$, where $A_G = A$ and D_G is the set of directions in D_k that covers at least one target in A. Differentiated with the *DCS-Greedy* algorithm, the pivot policy here is to select the direction d_{i^*,j^*} with the maximal work time into D_s among all the candidate directions in D_G. The two cases of this process are the same as the ones in the *DCS-Greedy* algorithm.

Repeat to handle Cases 1 and 2 and get a set of nonconflicting directions D_s as in the *DCS-Greedy* algorithm. If the elimination process succeeds, we add the work time of the removed directions to the work time of directions in D_s, i.e., $\forall d_{i,j} \in D_s, t_{i,j,k} = t_{i,j,k} + \sum_{d_{i,j'} \in D_k - D_s} t_{i,j',k}$. Otherwise, we set $D_s = \emptyset$. Finally, return D_s as the updated D_k with the work time $t_{i,j,k}$ for any $d_{i,j} \in D_s$.

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: $A_G = A, D_G = \{d_{i,j} | a_m \in d_{i,j}, \exists a_m \in A, \forall d_{i,j} \in D_k\}$,
   $G = (D_G, A_G), D_s = \emptyset$
2: Repeat to handle Case 1 and Case 2 to select a set of
   nonconflicting directions $D_s$ until $D_G$ is empty as in the
   DCS-Greedy algorithm
3: if $A_G$ is empty
4:     for each $d_{i,j} \in D_s$
5:         $t_{i,j,k} = t_{i,j,k} + \sum_{d_{i,j'} \in D_k - D_s} t_{i,j',k}$
6: else
7:     $D_s = \emptyset$
8: return $D_s$ and $\{t_{i,j,k} | \forall d_{i,j} \in D_s\}$
```

6.1.2 Direction Selection Process

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 $t_{a_m} = \max_{a_m \in d_{i,j} d_{i,j} \in D_k} t_{i,j,k}$. 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, $t^*_k = \min_{a_m \in A} \max_{a_m \in d_{i,j} d_{i,j} \in D_k} t_{i,j,k}$.

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, we remove redundant directions in D^*_k, since the targets covered by some directions formerly selected into D^*_k may be totally covered by the ones selected into D^*_k later. We employ a simple strategy here. Get a direction from D^*_k and check whether it is redundant or not. Remove it from D^*_k if it is redundant. Get another and check it until all the directions in D^*_k have been checked. 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 $(D_k, t_k, \{t_{i,j,k} | \forall d_{i,j} \in D_k\})$
 1: $t^*_k = \min_{a_m \in A} \max_{a_m \in d_{i,j} d_{i,j} \in D_k} t_{i,j,k}$
 2: $D^*_k = \emptyset, A' = A$
 3: $D'_k = \{d_{i,j} | t_{i,j,k} \geq t^*_k, \forall d_{i,j} \in D_k\}$
 4: Sort the directions in $D'_k$ according to the corresponding
    work time in nonincreasing order
 5: while $A' \neq \emptyset$
 6:     Remove the direction $d_{i,j}$ from the head of $D'_k$
 7:     if $\exists a_m \in A', a_m \in d_{i,j}$
 8:         $D^*_k = D^*_k \cup \{d_{i,j}\}$
 9:         $A' = A' - \{a_m | a_m \in d_{i,j}, \forall a_m \in A'\}$
10: Remove redundant directions in $D^*_k$
11: return $D^*_k$ and $t^*_k$
```

6.2 Feedback Algorithm

As 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* algorithm 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 4, the conflicting direction elimination process proposed in Section 6.1.1 and the direction selection process proposed in Section 6.1.2 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 uth 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 by $D_k = \{d_{i,j} | \ t_{i,j,k} > 0, \forall d_{i,j} \in D\}, k = 1 \dots K$. The former $u-1$ sets $D_1, D_2, \dots, D_{u-1}$ are cover sets and the latter $K-u+1$ sets may not be cover sets. We set the collection of the latter $K-u+1$ 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 6.1.1. 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 6.1.2.

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 uth set a cover set. For each $d_{i,j} \notin D_v^*$, a constraint $t_{i,j,u} = 0$ that indicates $d_{i,j}$ does not work in the uth cover set, and for each $d_{i,j} \in D_v^*$, a constraint $t_{i,j,u} = \min(\delta, t_v^*)$ that indicates $d_{i,j}$ works in the uth cover set are added to the LP problem, where δ is a quite small positive number. Instead of immediately determining the final work time of a cover set, we use the parameter δ to indicate whether a direction works in the cover set or not. The final work time of all the cover sets is computed at the end of the algorithm.

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 Kth 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 $t_{i,j,u} = \min(\delta, t_v^*)$ to the LP problem
18:    $u = u + 1$
19:  else
20:    break
21: if $u == K + 1$
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 $l_{net}$
```

6.3 MDCS-Greedy Algorithm

In the *Progressive* and *Feedback* algorithms, the LP problem is solved once in each iteration, which may result in heavy computation overhead and long runtime. In this section, we propose an algorithm called *MDCS-Greedy* without LP to find multiple cover sets. In each iteration of this algorithm, we compute at most *one* cover set. We set the work time of each cover set as a fixed value Δt, which is determined according to the application requirements on both the network lifetime and the number of cover sets. Larger Δt may result in shorter network lifetime, while smaller Δt may result in more cover sets, which we will discuss specifically in Section 7. An iteration of the *MDCS-Greedy* algorithm consists of the following steps.

First, we find the sensors whose residual lifetimes are no less than Δt. The directions of these sensors are selected into a set D'.

Second, implement the conflicting direction elimination process to eliminate the conflicting directions in D'. The pivot policy in this process takes into consideration the residual lifetime of each sensor and works as follows. First, we find the uncovered target a_{m^*} that can be covered by minimal number of directions. Then, we find the sensor s_{i^*} with the longest residual lifetime among all the candidates whose directions can cover a_{m^*}. Finally, the direction d_{i^*,j^*} of s_{i^*} that can cover a_{m^*} is selected. If this elimination process succeeds, it returns the updated cover set D'; otherwise, D' is set empty and the *MDCS-Greedy* algorithm exits.

Finally, when the cover set D' is found, implement the direction selection process to select a subset D^* of D' so as to save energy. When selecting a cover set $D^* \subseteq D'$, we select the direction $d_{i,j} \in D'$ of sensor s_i that has the longest residual lifetime L_i to cover some uncovered targets each time. Then, the work time Δt is assigned to D^* and accumulated to the total network lifetime. Moreover, the residual lifetime of any selected sensor s_i is updated, i.e., $L_i = L_i - \Delta t, \forall d_{i,j} \in D^*$.

The iteration consisting the above steps is repeated until it fails to find a cover set in the current iteration.

The *MDCS-Greedy* algorithm is shown below.

MDCS-Greedy Algorithm

```
1: $l_{net} = 0$
2: $Found =$ TRUE
3: repeat
4:   $D' = \{d_{i,j} | \ L_i \geq \Delta t, \forall d_{i,j} \in D\}$
```

```
5:  Call the conflicting direction elimination process to
    make $D'$ a cover set.
6:  if $D' \neq \emptyset$
7:      Call the direction selection process to select a cover
        set $D^* \subseteq D'$
8:      Assign the work time $\Delta t$ to $D^*$
9:      $l_{net} = l_{net} + \Delta t$
10:     for each $d_{i,j} \in D^*$
11:         $L_i = L_i - \Delta t$
12: else
13:     $Found =$ FALSE
14: until $Found ==$ FALSE
15: return $l_{net}$
```

6.4 Distributed Algorithms

In this section, we present a distributed algorithm called *MDCS-Dist* based on the centralized algorithms, where a sensor only cooperates with its neighbors in its communication range. In the *MDCS-Dist* algorithm, sensors work in rounds. A round is equivalent to a cover set when all the targets are covered in this round. Each round lasts for a period of Δt, which is the same as the one in the *MDCS-Greedy* algorithm. There is a scheduling stage prior to each round. In the scheduling stage, a sensor probes the states of its neighbors and decides its work direction.

First, a sensor broadcasts a message to its neighbors including each target that it can cover. Each sensor s_i assigns a priority p_m to each target a_m that it can cover locally. The fewer times a target can be covered by its neighbors, the higher priority the target is assigned to. A sensor tends to work in the direction that covers the uncovered target with the highest priority in this round. This strategy is similar to the pivot policy used in the centralized algorithms, which is to find a direction to cover the target that can be covered by minimal number of directions, except that this strategy is to find a target locally.

Second, each sensor s_i initializes a timer uniformly distributed in $[0, T_p]$ and goes to sleep. When the timer decreases to zero, s_i wakes up, marks itself PREWORK, broadcasts a probing message, and waits for a period for its neighbors' replies. On receiving the probing message, any neighbor $s_{i'}$ that is active but not in the PREWORK state responds to s_i with a message indicating its work direction. After receiving the neighbors' replies, s_i decides whether to sleep. If it finds out that itself does not cover any uncovered target, it sleeps. Otherwise, it erases the PREWORK mark, switches to the direction that covers the uncovered target a_{m^*} with the highest priority p_{m^*}, and broadcasts a message to its neighbors indicating its work direction.

Third, when a neighbor of s_i, say $s_{i'}$, which is not in the PREWORK state, receives the message from s_i indicating its work direction, it checks whether it becomes redundant, i.e., all the targets covered by its work direction have been covered by the sensors recently waked up. If it finds out that it has become redundant, it notifies its neighbors and sleeps in the round. This check strategy is similar to the one of removing redundant directions in the centralized algorithms in Section 6.1.2.

In this paper, we will compare the performance of the *MDCS-Dist* algorithm with another distributed algorithm named *SNCS*, proposed by Ai and Abouzeid [17], where sensors also work in rounds. The objective of [17] is to cover maximal targets with minimal sensors. The *SNCS* algorithm works as follows. At the beginning of the scheduling stage, each sensor is active. It assigns itself a priority that is equal to its residual energy and randomly picks a direction as its work direction. Each sensor s_i broadcasts a message including the priority, location, and work direction to its neighbors. Upon receiving a message, s_i calculates the number of *acquired* targets for each of its directions. A target is *acquired* to a sensor if the target is not covered by any higher priority neighbor of this sensor. If there are acquired targets, s_i chooses the direction that has the maximum number of acquired targets as its current work direction and broadcasts a message to inform its neighbors about its new work direction. Otherwise, s_i activates a transition timer. The timer is off if a new message arrives and there are acquired targets for s_i. If the timer remains on for longer than a duration T_w, s_i sleeps in this round.

The *MDCS-Dist* algorithm is shown below.

MDCS-Dist Algorithm

```
    /* scheduling stage prior to one round */
1:  $s_i$ broadcasts a message to its neighbors including each
    target that it can cover
2:  $s_i$ assigns a priority $p_m$ to each target $a_m$ that it can
    cover locally. The fewer times a target can be covered
    by the directions of its neighbors, the higher priority the
    target is assigned to.
3:  $s_i$ initializes a decreasing timer uniformly distributed in
    $[0, T_p]$ and goes to sleep
4:  if the timer $\leq 0$
5:      $s_i$ wakes up, marks itself PREWORK and broadcasts
        a probing message
6:      for each $s_{i'} \in N_i$ that receives the probing message
7:          if $s_{i'}$ is active but not in the PREWORK state
8:              $s_{i'}$ responds to $s_i$ to indicate its work direction
9:      if $\nexists a_m$ that $a_m$ is uncovered and can be covered by $s_i$
10:         $s_i$ goes to sleep
11:     else
12:         $s_i$ erases the PREWORK mark, switches to the
            direction that covers the uncovered target $a_{m^*}$
            with the highest priority $p_{m^*}$ and notifies its
            neighbors
13:     for each $s_{i'} \in N_i$ that receives the message from
        $s_i$ indicating its work direction
14:         if $s_{i'}$ is active but not in the PREWORK state
15:             if $\forall a_m \in d_{i',j}$ that $d_{i',j}$ is the work direction and
                $a_m$ is covered
16:                 $s_{i'}$ notifies its neighbors and goes to sleep
```

The time complexity of the algorithms for the MDCS is shown in Table 1. In each iteration of the *Progressive* and

TABLE 1
Time Complexity of the Algorithms for the MDCS

Progressive	*Feedback*	*MDCS-Greedy*	*MDCS-Dist*
$O(K^3N^4W^3)$	$O(K^4N^3W^3)$	$O(N^3WM)$	$O(N^2WM)$

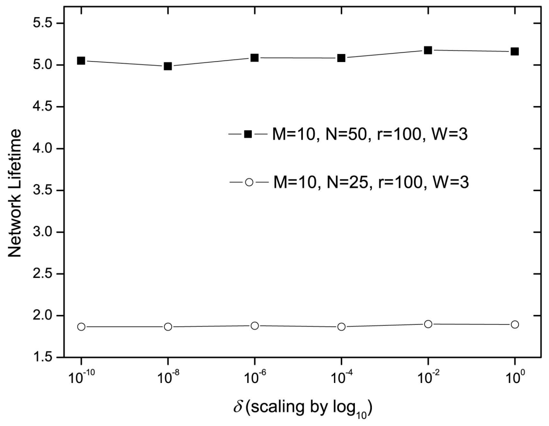


Fig. 7. Network lifetime versus δ in the $Feedback$ algorithm.

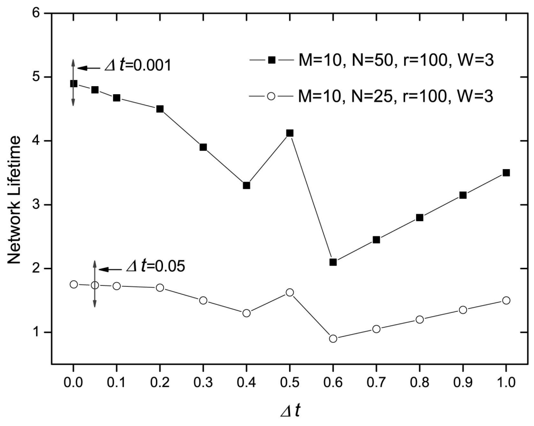


Fig. 8. Network lifetime versus Δt in the *MDCS-Greedy* algorithm.

Feedback algorithms, the LP problem is solved once. The time complexity of the LP problem is $O(n^3)$ using Ye's algorithm [19], where n is the number of variables and $n = K + KNW$. The time complexity of each iteration is $O(K^3N^3W^3)$, which is mainly determined by the time complexity of the LP problem. In the *Progressive* algorithm, the number of iterations is at most N/ε, where N is the longest possible network lifetime when there is only one direction in any cover set. Therefore, the time complexity of this algorithm is $O(K^3N^4W^3)$ by assuming that ε is a constant. The time complexity of the *Feedback* algorithm is $O(K^4N^3W^3)$ since the LP problem is solved for at most $K + 1$ times. In the *MDCS-Greedy* algorithm, the time complexity to get a cover set in each iteration is $O(N^2WM)$ and the number of iterations is at most $N/\Delta t$. Therefore, the time complexity of this algorithm is $O(N^3WM)$ when Δt is fixed. In the *MDCS-Dist* and *SNCS* algorithms, the time complexity of each round is $O(NWM)$ and the time complexity is $O(N^2WM)$.

7 Simulation Results of the MDCS

In this section, we evaluate the performance of the *Progressive*, *Feedback*, *MDCS-Greedy*, *MDCS-Dist*, and *SNCS* algorithms through simulations with the same configurations as in Section 5.3. The optimization toolbox in Matlab is used to solve the LP problem. For the *Progressive* and *Feedback* algorithms, the maximal number of cover sets in one iteration is equal to the number of sensors, i.e., $K = N$. For the *Progressive* algorithm, we set $\varepsilon = 0.001$. For the *MDCS-Dist* and *SNCS* algorithms, we assume that the communication radius is twice of the sensing radius. We randomly generate 10 deployments of sensors and targets, and average the result on every deployment for each algorithm.

7.1 Parameters Tuning

The initial lifetime of each sensor is set as 1 in this section.

7.1.1 δ in the Feedback Algorithm

Fig. 7 shows the network lifetime of the *Feedback* algorithm when δ varies from 10^{-10} to 1. The upper curve shows the network lifetime when 10 targets and 50 sensors are deployed, r is fixed at 100, and W is set as 3. The lower curve shows the network lifetime when 25 sensors are deployed. We can see from the two curves that the value of δ does not affect the network lifetime much. We observe in the figure that the algorithm works slightly better when $\delta = 0.01$. In the following simulations, we set $\delta = 0.01$ for the *Feedback* algorithm.

7.1.2 Δt in the MDCS-Greedy Algorithm

Fig. 8 shows the network lifetime of the *MDCS-Greedy* algorithm when Δt, the work time of each cover set, varies from 0.001 to 1. When Δt increases from 0.001 to 0.1, the network lifetime drops slightly. When Δt becomes greater than 0.1, the network lifetime becomes unstable and drops dramatically. The reason is that, when the initial lifetime of each sensor is divisible by Δt, the sensor has a high probability to use up all of its lifetime. Note that when Δt is too small, it may result in too many cover sets totally. Therefore, in the following simulations, we set $\Delta t = 0.05$ for the *MDCS-Greedy*, *MDCS-Dist*, and *SNCS* algorithms.

7.2 Communication and Computation Overhead

In this section, we first provide an energy model for a sensor node. Then we compute the communication and computation overhead for all the algorithms based on this energy model.

For a sensor node, the energy is mainly consumed by three components: the processor, transceiver, and sensor such as an ultrasonic sensor on the node. Note that we differentiate the terms *sensor* and *node* in this section. The power consumptions of different components depend on the different work modes. For the processor or the sensor, it can work in either the active or sleep mode. For the transceiver, it can work in one of the four modes: transmit, receive, idle, or sleep. According to Crossbow Mica2 motes [20], we set up the energy consumption levels of different components, as shown in Table 2. The transmission data rate is set as 19.2 Kbps. We assume that when a node works in a cover set, the processor, transceiver, and sensor are in the active, idle, and active mode, respectively. Assume that the initial energy of each sensor node is 2,000 J. The total expected work time (second) of a node T_0 is calculated as 312,012 s. The network lifetime (unit) is computed as the ratio of the time duration (second) before at least one target is not covered, i.e., a cover set is not found, to T_0.

For the *MDCS-Dist* and *SNCS* algorithms, we assume that all the messages are 16 bytes since each message sent

TABLE 2
Power Consumption Levels

Component	Mode	Power
Processor	Active	$21.6mW$
	Sleep	$40.5\mu W$
Transceiver	Transmit	$67.5mW$
	Receive	$25mW$
	Idle	$22.5mW$
	Sleep	$25\mu W$
Sensor	Active	$20mW$
	Sleep	$25\mu W$

by a node contains only the information about the node itself. Both T_p in the *MDCS-Dist* algorithm and T_w in the *SNCS* algorithm with $M = 10, N = 50, W = 3$, and $r = 100$ are set as 5 s, when a node can exchange messages with its neighbors to decide whether to sleep. Note that T_p and T_w are proportional to the average number of the neighbors. When a node has more neighbors, it may need more time to exchange messages with its neighbors. The number of neighbors of a node is proportional to the number of nodes in the network, the communication range of a node, and is inversely proportional to the deployment area of the network. Fig. 9 shows the network lifetime of the two algorithms with and without considering the communication and computation overhead. The network lifetime of the *MDCS-Dist* algorithm drops about 3 percent when considering the overhead, while the network lifetime of the *SNCS* algorithm drops about 5 percent. The *MDCS-Dist* algorithm has lower overhead since each node sleeps for a random period at the beginning of each round and goes to sleep immediately when it finds out that it does not cover any uncovered target.

For the *Progressive*, *Feedback*, and *MDCS-Greedy* algorithms, the cover sets are computed in the sink. We assume that there is a scheduling stage before the network starts to monitor all the targets. In the scheduling state, the information of all the cover sets is broadcasted to the nodes. Therefore, both the size of each message and the time duration when a transceiver is in the idle mode are proportional to the number of nodes and the number of total cover sets. Fig. 10 shows the network lifetime of the

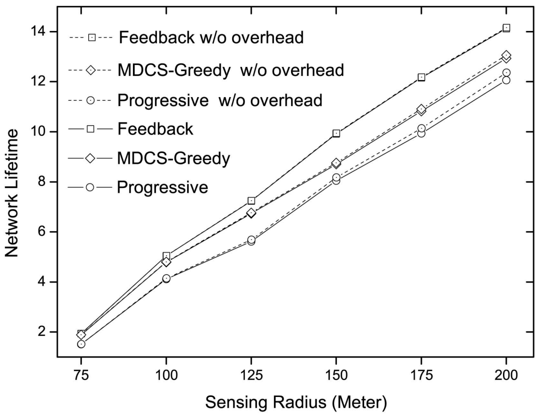


Fig. 10. Affect of communication and computation overhead to network lifetime of the *Progressive*, *Feedback*, and *MDCS-Greedy* algorithms with $M = 10, N = 50$, and $W = 3$.

three algorithms with and without considering the overhead. The network lifetime of the *Progressive* algorithm drops about 2 percent when considering the overhead and the network lifetime of the *MDCS-Greedy* algorithm drops about 1 percent, while the network lifetime of the *Feedback* algorithm drops about 0.2 percent, which is almost 10 times less than the other algorithms. The *Feedback* algorithm has the lowest overhead because it has the least number of total cover sets as shown in Fig. 15.

In the following simulations, the communication and computation overhead is considered for all the algorithms.

7.3 Algorithm Comparison

7.3.1 Network Lifetime

Fig. 11 shows the relationship between the network lifetime and the sensing radius r when $M = 10, N = 50$, and $W = 3$. We can see that, for all algorithms, the network lifetime increases when the sensing radius r increases. Among all the algorithms, the *Feedback* algorithm has the best performance, followed by the *MDCS-Greedy* algorithm. The *MDCS-Dist* algorithm has about the same performance as the *Progressive* algorithm, even when r increases. The network lifetime of the *SNCS* algorithm increases relatively slowly when r increases to greater than 125, i.e., the network becomes more complicated. The *MDCS-Greedy* algorithm works better than

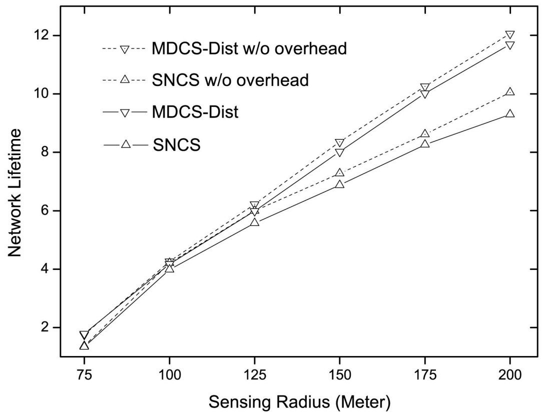


Fig. 9. Affect of communication and computation overhead to network lifetime of the *MDCS-Dist* and *SNCS* algorithms with $M = 10, N = 50$, and $W = 3$.

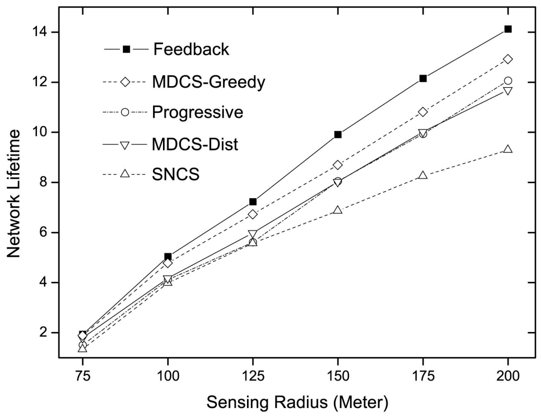


Fig. 11. Network lifetime versus sensing radius r with $M = 10, N = 50$, and $W = 3$.

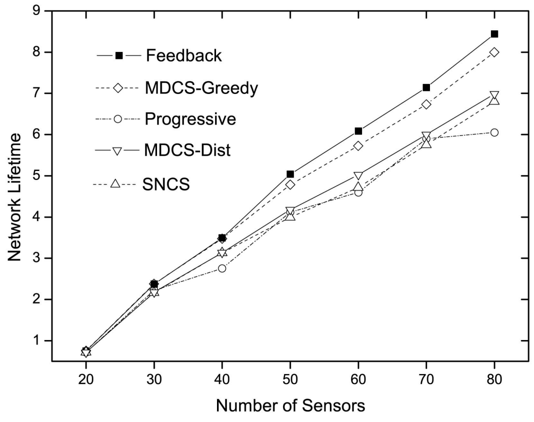


Fig. 12. Network lifetime versus number of sensors N with $M = 10, r = 100$, and $W = 3$.

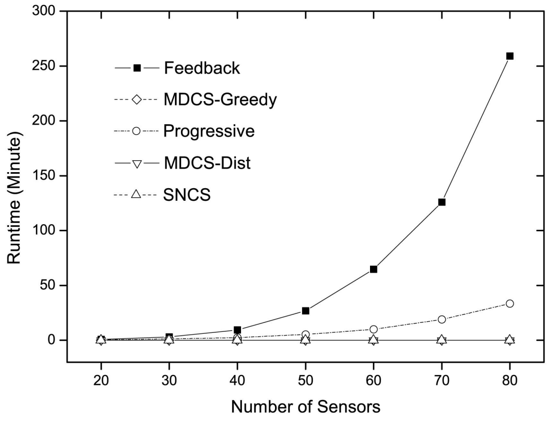


Fig. 14. Runtime versus number of sensors N with $M = 10, r = 100$, and $W = 3$.

the *Progressive* algorithm because of the following reason. In each iteration of the *Progressive* algorithm, multiple cover sets and their work time are determined, while the residual lifetimes of the sensors are not taken into consideration. However, in the *MDCS-Greedy* algorithm, this factor has been considered both in the conflicting direction elimination process and in the direction selection process.

Fig. 12 shows the relationship between the network lifetime and the number of sensors when 10 targets are deployed, r is fixed at 100, and W is set as 3.

Fig. 13 shows that the network lifetime drops as the number of targets increases when $N = 50, r = 100$, and $W = 3$. We can see that the network lifetime drops significantly when M varies from 1 to 2 and then drops relatively slowly when M varies from 5 up to 20.

7.3.2 Runtime

Fig. 14 shows the runtime for the algorithms when $M = 10, r = 100$, and $W = 3$. As the number of sensors increases, the runtime increases. The runtime of the *Feedback* algorithm is longer than the *Progressive* algorithm. The *MDCS-Greedy*, *MDCS-Dist*, and *SNCS* algorithms have much shorter runtime than the other two algorithms.

7.3.3 Number of Total Cover Sets

Fig. 15 shows the number of the total cover sets of each algorithm when $M = 10, r = 100$, and $W = 3$. We can see that the *Feedback* algorithm generates the least cover sets. As stated before, fewer cover sets with longer work time are more efficient and practical.

7.4 Discussion

As shown by the simulations, the *Feedback* algorithm has the longest network lifetime, the fewest cover sets, and the lowest communication overhead. However, it has a longer runtime. It is feasible that the sink, a central processing base station, can collect the information needed from the sensors and run the algorithm. The sink then transfers the result back to sensors. The *MDCS-Dist* and *SNCS* algorithms have shorter network lifetime and higher communication overhead, while they are able to detect a node that has died unexpectedly in the scheduling stage prior to a round.

8 Conclusions and Future Work

Scheduling algorithms to save energy and prolong the network lifetime are always important for sensor networks. However, algorithms designed for omnidirectional sensor

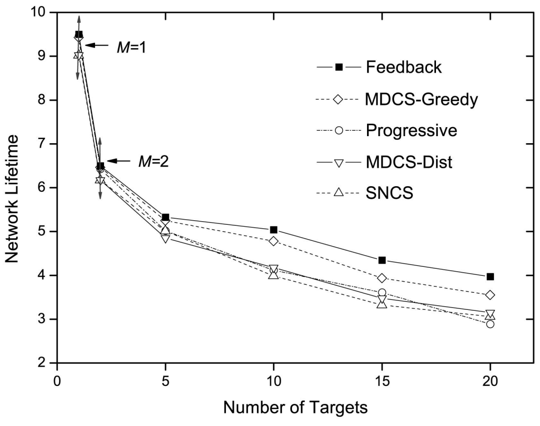


Fig. 13. Network lifetime versus number of targets M with $N = 50, r = 100$, and $W = 3$.

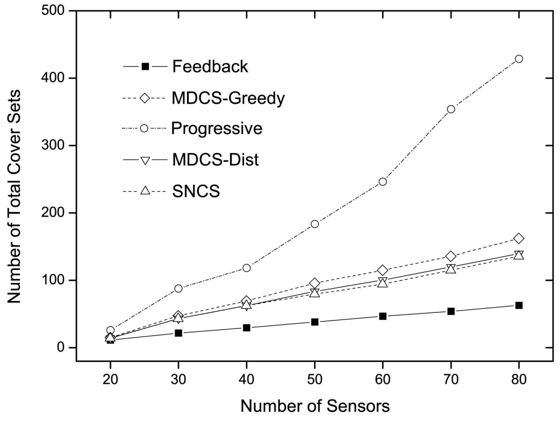


Fig. 15. Number of total cover sets versus number of sensors N with $M = 10, r = 100$, and $W = 3$.

networks may not be suitable for directional sensor networks. In this paper, we have studied the problem of finding a directional cover set, the DCS, and the problem of finding multiple directional cover sets to extend the network lifetime, the MDCS, and proved that both problems are NP-complete. We have proposed several centralized algorithms and distributed algorithms. As future work, we plan to find out how the proposed algorithms approximate the optimal solution for the MDCS and test the proposed algorithms in the real environments.

Appendix A
Proof Outline of Theorem 3

By the *DCS-Search* algorithm, we first get a tuple $G = (D_G, A_G)$ from D and A. To assist this proof, we give a definition as follows. Given a tuple $G = (D_G, A_G)$, if there exists a cover set, which is a subset of D_G, covering all the targets in A_G, we say that G is *feasible*; otherwise, we say that G is *infeasible*. According to this definition, G is feasible at the beginning of the search process.

When G is feasible and Case 1 is satisfiable, we will prove by contradiction that after handling Case 1, the newly updated G, denoted by G', is still feasible. Assume that G' is infeasible. According to the process of handling Case 1, $G' = (D_G - V, A_G - U)$. Recall that U is the set of targets in A_G that is covered by d_{i^*,j^*}. V is the set of directions in D_G that covers no target in the updated A_G from which the targets in U have been removed. The only way to make G' feasible is to add more directions to cover the targets in $A_G - U$. We add V to G' and get a tuple $G'' = (D_G, A_G - U)$. Because the directions in V make no contribution to the coverage of the targets in $A_G - U, G''$ is still infeasible. Then, we add U to G'' and get the tuple (D_G, A_G), which is G. Because there does not exist a cover set in D_G to cover all the targets in $A_G - U$, which is a subset of A_G, G is infeasible. This is contradictory to the condition that G is feasible. Therefore, after handling Case 1, the updated G is still feasible.

Now we consider Cases 2 and 3. Suppose that in certain step of the search process, G is feasible and Case 2 is satisfiable. When handling Case 2, we select a direction d_{i^*,j^*} of sensor s_i^*. If selecting this direction results in that G is infeasible in the following step, backtracking to try another direction of s_i^* is allowed by handling Case 3. Since G is feasible, it implies that there exists at least one direction of s_i^* that can be selected into the final D_s and selecting this direction results in that G is still feasible in the following step. Hence, handling Case 2 together with Case 3 can finally result in that G is feasible in the following step.

By repeatedly handling Cases 1, 2, and 3, we ultimately get a set of selected directions D_s. Now we prove that D_s is a cover set for A. Recall that when targets are removed from A_G, nonconflicting directions that only cover these targets are selected into D_s in Case 1. When the search process completes, i.e., $A_G = \emptyset, D_s$ is a set of directions that covers all the targets in A, which do not conflict with each other. Therefore, D_s is a cover set for A.

Acknowledgments

This work is supported in part by Hong Kong RGC grants PolyU-5236/06E, PolyU-5232/07E, PolyU-5243/08E, Hong Kong PolyU ICRG grant A-PH12, NSFC 60473092, NSFC 90612018, NSFC 60828003, NSF CNS-0832120, National Basic Research Program of China under Grant No. 2006CB303000, and National High Technology Research and Development Program of China under Grant No. 2007AA01Z180.

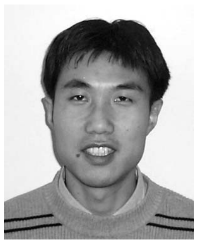
References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks," *ACM Trans. Multimedia Computing, Comm. and Applications,* vol. 40, no. 8, 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," *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys),* 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," *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys),* 2005.
[4] W. Feng, E. Kaiser, W.C. Feng, and M.L. Baillif, "Panoptes: Scalable Low-Power Video Sensor Networking Technologies," *ACM Trans. Multimedia Computing, Comm. 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," *Proc. IEEE Int'l Conf. Robotics and Automation,* 2006.
[6] L.A. Wolsey and G.L. Nemhauser, *Integer and Combinatorial Optimization.* Wiley, 1999.
[7] S. Kumar, T.H. Lai, and J. Balogh, "On K-Coverage in a Mostly Sleeping Sensor Network," *Proc. ACM MobiCom,* 2004.
[8] C. Gui and P. Mohapatra, "Power Conservation and Quality of Surveillance in Target Tracking Sensor Networks," *Proc. ACM MobiCom,* 2004.
[9] F. Ye, G. Zhong, J. Cheng, S. Lu, and L. Zhang, "PEAS: A Robust Energy Conserving Protocol for Long-Lived Sensor Networks," *Proc. IEEE Int'l Conf. Network Protocols (ICNP),* 2002.
[10] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated Coverage and Connectivity Configuration in Wireless Sensor Networks," *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys),* 2003.
[11] H. Zhang and J.C. Hou, "Maintaining Sensing Coverage and Connectivity in Large Sensor Networks," *Ad Hoc & Sensor Wireless Networks, an Int'l J.,* vol. 1, pp. 89-124, 2005.
[12] Y. Zou and K. Chakrabarty, "A Distributed Coverage- and Connectivity-Centric Technique for Selecting Active Nodes in Wireless Sensor Networks," *IEEE Trans. Computers,* vol. 54, no. 8, pp. 978-991, Aug. 2005.
[13] H. Liu, P. Wan, C. Yi, X. Jia, S. Makki, and P. Niki, "Maximal Lifetime Scheduling in Sensor Surveillance Networks," *Proc. IEEE INFOCOM,* 2005.
[14] M.X. Cheng, L. Ruan, and W. Wu, "Achieving Minimum Coverage Breach under Bandwidth Constraints in Wireless Sensor Networks," *Proc. IEEE INFOCOM,* 2005.
[15] M. Cardei, M.T. Thai, Y. Li, and W. Wu, "Energy-Efficient Target Coverage in Wireless Sensor Networks," *Proc. IEEE INFOCOM,* 2005.
[16] H. Ma and Y. Liu, "On Coverage Problems of Directional Sensor Networks," *Proc. Int'l Conf. Mobile Ad-Hoc and Sensor Networks (MSN),* 2005.
[17] J. Ai and A.A. Abouzeid, "Coverage by Directional Sensors in Randomly Deployed Wireless Sensor Networks," *J. Combinatorial Optimization,* vol. 11, no. 1, pp. 21-41, Feb. 2006.
[18] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms,* second ed. MIT Press, 2001.
[19] Y. Ye, "An $O(n^3L)$ Potential Reduction Algorithm for Linear Programming," *Math. Programming,* vol. 50, pp. 239-258, 1991.
[20] MICA2 Datasheet, http://www.xbow.com/products/Product_ pdf_files/Wireless_pdf/MICA2_Datasheet.pdf, 2007.

Yanli Cai received the bachelor degree in computer science from the Sun Yat-sen University, China, in 2003, and the PhD degree in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2009. During her PhD program, she was a research assistant in The Hong Kong Polytechnic University from February 2006 to July 2007, an intern in Microsoft Research Asia from July 2007 to May 2008, and an intern in Microsoft, Redmond, Washington, from October 2008 to December 2008.

Wei Lou received the BE degree in electrical engineering from Tsinghua University, China, in 1995, the ME degree in telecommunications from Beijing University of Posts and Telecommunications, China, in 1998, and the PhD degree in computer engineering from Florida Atlantic University in 2004. He is currently an assistant professor in the Department of Computing, The Hong Kong Polytechnic University, HKSAR, China. His current research interests are in the areas of mobile ad hoc and sensor networks, peer-to-peer networks, mobile computing, and computer networks. He has worked intensively on designing, analyzing, and evaluating practical algorithms with the theoretical basis as well as building prototype systems. His research work is supported by several Hong Kong GRF grants and Hong Kong Polytechnic University ICRG grants.

Minglu Li received the graduate degree from the School of Electronic Technology, University of Information Engineering in 1985 and the PhD degree in computer software from Shanghai Jiao Tong University (SJTU) in 1996. He is a full professor and the vice chair of the Department of Computer Science and Engineering and the director of Grid Computing Center of SJTU. Currently, his research interests include grid computing, services computing, and sensor networks. He has presided over 20 projects supported by the National Natural Science Foundation, National Key Technology R&D Program, 863 Program, 973 Program, and Science and Technology Commission of Shanghai Municipality (STCSM). He has published more than 100 papers in academic journals and international conferences. He is also a member of the Expert Committee of the ChinaGrid Program of Ministry of Education, a principal scientist of ShanghaiGrid, which is a grand project of STCSM, a member of the Executive Committee of the ChinaGrid Forum, and the Executive Committee of the Technical Committee on Services Computing of the IEEE Computer Society.



Xiang-Yang Li received the bachelor degrees in computer science and business management from Tsinghua University, P.R. China, in 1995, the MS and PhD degrees in computer science from the University of Illinois at Urbana-Champaign in 2000 and 2001, respectively. He has been an associate professor since 2006 and an assistant professor of computer science at the Illinois Institute of Technology from 2000 to 2006. From May 2007 to August 2008, he was a visiting professor of Microsoft Research Asia. His research interests span wireless ad hoc and sensor networks, noncooperative computing, computational geometry, optical networks, and cryptography. He has been a guest editor of special issues of *ACM Mobile Networks and Applications* and the *IEEE Journal on Selected Areas in Communications*.