[1]Maximal Lifetime Scheduling in Sensor Surveillance Networks

Hai Liu[1], Pengjun Wan[2], Chih-Wei Yi[2], Xiaohua Jia[1], Sam Makki[3] and Niki Pissinou[4]

Dept of Computer Science

[1]City University of Hong Kong

[2]Illinois Institute of Technology

Dept of Electrical Engineering & Computer Science

[3]University of Toledo

Telecommunications & Information Technology Institute

[4]Florida International University

Email: {liuhai@cs.cityu.edu.hk, wan@cs.iit.edu, jia@cs.cityu.edu.hk, Kmakki@eng.utoledo.edu, pissinou@fiu.edu}

Abstract--**This paper addresses the maximal lifetime scheduling problem in sensor surveillance networks. Given a set of sensors and targets in a Euclidean plane, a sensor can watch only one target at a time, our task is to schedule sensors to watch targets, such that the lifetime of the surveillance system is maximized, where the lifetime is the duration that all targets are watched. We propose an optimal solution to find the target watching schedule for sensors that achieves the maximal lifetime. Our solution consists of three steps: 1) computing the maximal lifetime of the surveillance system and a workload matrix by using linear programming techniques; 2) decomposing the workload matrix into a sequence of schedule matrices that can achieve the maximal lifetime; 3) obtaining a target watching timetable for each sensor based on the schedule matrices. Simulations have been conducted to study the complexity of our proposed method and to compare with the performance of a greedy method.**

Keywords-- **Energy efficiency, lifetime, scheduling, sensor network, surveillance system.**

1. INTRODUCTIONS

A wireless sensor network consists of many low-cost and low-powered sensor devices (called sensor nodes) that collaborate with each other to gather, process, and communicate information using wireless communications [4]. Applications of sensor networks include military sensing, traffic surveillance, environment monitoring, building structures monitoring, and so on. One important characteristic of sensor networks is the stringent power budget of wireless sensor nodes, because those nodes are usually powered by batteries and it may not be possible to recharge or replace the batteries after they are deployed in hostile or hazardous environments [15]. The surveillance nature of sensor networks requires a long lifetime. Therefore, it is an important research issue to prolong the lifetime of sensor networks in surveillance services.

In this paper, we discuss a scheduling problem in sensor surveillance networks. Given a set of targets and sensors in an area, the sensors are used to watch (or monitor) the targets. A sensor can watch targets that are within its surveillance range, and a target can be inside several sensors' watching range. Suppose each sensor has a given energy reserve (in terms of the length of time it can operate correctly) and each sensor can watch at most one target at a time. The problem is to find a schedule for sensors to watch the targets, such that all targets should be watched by sensors at anytime and the lifetime of the surveillance is maximized. The lifetime is the duration up to the time when there exists one target that cannot be watched by any sensors due to the depletion of energy of the sensor nodes. By using this schedule, a sensor can switch off to save energy when it is not its turn to watch a target. We assume the positions of targets and sensors are given and are static. This information can be obtained via a distributed monitoring mechanism [10] or the scanning method [11].

Extensive research has been done on extending the lifetime of sensor networks. Authors in [12] studied the upper bounds on the lifetime of sensor networks used in data gathering in various scenarios. Both analytical results and extensive simulations showed that the derived upper bounds are tight for some scenarios and near-tight (about 95%) for the rest. The authors further proposed a technique to find the bounds of lifetime by partitioning the problem into the sub-problems for which the bounds are either already known or easy to derive. A differentiated surveillance service for various target areas in sensor networks was discussed in [15]. The proposed protocol was based on an energy-efficient sensing coverage protocol that makes full coverage to a certain geographic area. It is also guaranteed to achieve a certain degree of coverage for fault tolerance. Simulations

[1] This work is supported in part by Hong Kong Research Grant Council under grant No. CityU 1079/02E, NSF CCR-0311174, NSF 0123950, and NSF 9988336.

showed that a much longer network lifetime and a small communication overhead could be achieved.

Another important technique used to prolong the lifetime of sensor networks is the introduction of switch on/off modes for sensor nodes. Recent works on energy efficiency in three aspects, namely area coverage, request spreading and data aggregation, were surveyed in [8]. It pointed out that the best method for conserving energy is to turn off as many sensors as possible, at the same time, however, the system must maintain its functionality. A node scheduling scheme was developed in [3]. This scheme schedules the nodes to turn on or off without affecting the overall service provided. A node decides to turn off when it discovers that its neighbors can help it to monitor its monitoring area. The scheduling scheme works in a localized fashion where nodes make decisions based on its local information. Similar to [3], the work in [9] defined a criterion for sensor nodes to turn themselves off in surveillance systems. A node can turn itself off if its monitoring area is the smallest among all its neighbors and its neighbors will become responsible for that area. This process continues until the surveillance area of a node is smaller than a given threshold. A deployment of a wireless sensor network in the real world for habitat monitoring was discussed in [13]. A network consisting of 32 nodes was deployed on a small island to monitor the habitat environment. Several energy conservation methods were adopted, including the use of sleep mode, energy efficient communication protocols, and heterogeneous transmission power for different types of nodes.

The rest of the paper is organized as follows. Section 2 is the problem definition. Section 3 presents our solution that consists of three parts. Section 3.1 gives a linear programming formulation that is used to compute the maximal lifetime of the surveillance system. In section 3.2, we show that the maximal lifetime is achievable, and detailed algorithms for finding the schedule are presented. Section 3.3 discusses the final schedule timetable for sensors. Section 4 presents a numeric example solved by using our method and simulation results. We conclude our work in section 5.

2. System Model and Problem Statement

We consider a set of targets and a set of sensors that are used to watch targets and collect information. We first introduce the following notations:

S = the set of sensors.

T = the set of targets.

$n=|S|$ the number of sensors.

$m=|T|$ the number of targets.

$S(j)$ = the set of sensors that are able to watch target j, $j=1,\ldots,m$.

$T(i)$ = the set of targets that are within the surveillance range of sensor i, $i=1,\ldots,n$.

E_i = initial energy reserve of sensor i, $i=1,\ldots,n$.

Notice that $S(i)$ may overlap with $S(j)$ for $i\neq j$, and $T(i)$ may overlap with $T(j)$ for $i\neq j$. There are two requirements for sensors watching targets:

1) Each sensor can watch at most one target at a time.
2) Each target should be watched by one sensor at anytime.

The problem of our concern is, for given S and T, to find a schedule that meets the above two requirements for sensors watching targets, such that the lifetime of surveillance is maximized. The lifetime of surveillance is defined as the length of time until there exists a target j such that all sensors in $S(j)$ run out their energy.

3. Our Solutions

We tackle the problem in three steps. First, we compute the upper bound on the maximal lifetime of the system and a workload matrix of sensors. Second, we successfully decompose the workload matrix into a sequence of schedule matrices. Finally, we obtain a target watching timetable for each sensor.

3.1 Find Maximal Lifetime

We use linear programming (LP) technique to find the maximum lifetime of the system. Let L denote the lifetime of the surveillance system, and x_{ij} be the variable denoting the total time sensor i watching target j, where $i\in S$, $j\in T$. The problem of finding the maximum lifetime for sensors watching targets can be formulated as the following:

Objective: Max L

$$\text{s.t.} \sum_{i\in S(j)} x_{ij} = L \quad \forall j \in T, \tag{1}$$

$$\sum_{j\in T(i)} x_{ij} \le Min\{L, E_i\} \ \forall i \in S. \tag{2}$$

Equation (1) specifies that for each target j in T, the total time that sensors watch it is equal to the lifetime of the system. That is, each target should be watched throughout the lifetime.

Inequality (2) implies that for each sensor i in S, the total working time neither exceeds the lifetime of the system, nor exceeds its battery's lifetime.

The above formulation is a typical LP formulation, where x_{ij}, $1\le i\le n$ and $1\le j\le m$, are real number variables and the objective is to maximize L. The optimal results of x_{ij} and L can be computed in polynomial time.

However, L, obtained from computing the above LP formation, is the upper bound on the lifetime, and each x_{ij} specifies only the total time that sensor i should watch target j in order to achieve this upper bound L. Now we have two questions:

1) Is this upper bound of lifetime L achievable? If yes, then
2) How to schedule sensors to watch targets, such that each value of x_{ij}, $1\le i\le n$ and $1\le j\le m$, can be actually met?

In answering question 2), we need to find a schedule for each sensor that specifies from what time up to what time that this sensor should watch which target.

The values of x_{ij}, $1\le i\le n$ and $1\le j\le m$, obtained from the LP, can be represented as a matrix:

$$X_{n\times m}=\begin{bmatrix} x_{11}x_{12}\ldots x_{1m} \\ x_{21}x_{22}\ldots x_{2m} \\ \ldots\ldots \\ x_{n1}x_{n2}\ldots x_{nm} \end{bmatrix}_{n\times m}$$

We call matrix $X_{n\times m}$ ***workload*** matrix, for it specifies the total length of time that a sensor should watch a target. There are two important features about this workload matrix:

1) the sum of all elements in each column is equal to L (from eq. (1) in the LP formulation).
2) the sum of all elements in each row is less than or equal to L (from ineq. (2) in the LP formulation).

In the next step, we need to find the detailed schedule for sensors to watch targets based on the workload matrix.

3.2 Decompose Workload Matrix

The lifetime of the surveillance system can be divided into of a sequence of sessions. In each session, a set of sensors are scheduled to watch their corresponding targets; and in the next session, another set of sensors are scheduled to work (some sensors may work continuously for multiple sessions). Suppose a sensor will not switch to watch another target within a session. Thus, the schedule of sensors during a session can be represented as a matrix. In this matrix, there is only one positive number in each column, representing each target should be watched by one sensor at a time; and at most one positive number in each row, representing each sensor can watch at most one target at a time and there is no switching to watch other targets in a session. Furthermore, all the non-zero elements in this matrix have the same value, which is the time duration of this session. Now, our task becomes to decompose the workload matrix into a sequence of session schedule matrices, represented as:

$$\begin{bmatrix} x_{11}x_{12}\ldots x_{1m} \\ x_{21}x_{22}\ldots x_{2m} \\ \ldots\ldots \\ x_{n1}x_{n2}\ldots x_{nm} \end{bmatrix}_{n\times m}=\begin{bmatrix} 0z_1 0\ldots 0 \\ z_1 00\ldots 0 \\ \ldots\ldots \\ 00z_1\ldots 0 \end{bmatrix}+\begin{bmatrix} z_2 00\ldots 0 \\ 00 0\ldots z_2 \\ \ldots\ldots \\ 0z_2 0\ldots 0 \end{bmatrix}+\ldots+\begin{bmatrix} 000\ldots z_t \\ 000\ldots 0 \\ \ldots\ldots \\ 0z_t 0\ldots 0 \end{bmatrix},$$

where z_i, i=1,2,…,t, is the length of time of session i, and t the total number of sessions. We call this sequence of session schedule matrices the ***schedule*** matrices. Considering the schedule matrix of session i, all elements in it are either "0" or z_i; each column has exactly one non-zero element, and each row has at most one non-zero element (it could be all "0", indicating the sensor is idle in this session).

The next, we discuss how to decompose the workload matrix into a sequence of schedule matrices. We first consider a simple special case of n=m, i.e., the number of targets is equal to the number of sensors in the system. Then, we extend the result to the general case of n>m.

3.2.1 A Special Case n=m

We consider the case n=m. Let R_i and C_j denote the sum of row i and the sum of column j in the workload matrix, respectively. According to eq. (1) and ineq. (2) of the LP formation, we have:

$$C_j = L, j=1,2,\ldots,m. \quad (3)$$
$$R_i \le L, i=1,2,\ldots,n. \quad (4)$$

Furthermore, since $\sum_{i=1}^{n} R_i = \sum_{j=1}^{m} C_j = m\times L$ and n=m, we have:

$$\sum_{i=1}^{n} R_i = n\times L. \quad (5)$$

Combining (4) and (5), we have:

$$R_i = L, i=1,2,\ldots,n. \quad (6)$$

(3) and (6) imply that for the workload matrix the sum of each column is the same as the sum of each row, all equal to L.

We divide the workload matrix $X_{n\times m}$ by L and denote the new matrix by $Y_{n\times n}$. That is, $y_{ij} = x_{ij}/L$, for i,j=1,2,…,n. For matrix $Y_{n\times n}$, we have:

$$y_{ij} \ge 0 \text{ and } \sum_{i=1}^{n} y_{ij} = \sum_{j=1}^{n} y_{ij} = 1, \text{ for } i,j = 1,2,\ldots,n. \quad (7)$$

From (7), we know matrix $Y_{n\times n}$ is a *Doubly Stochastic Matrix* [1, 2].

Theorem 1. Matrix $Y_{n\times n}$ can be decomposed as:

$$Y_{n\times n} = c_1P_1 + c_2P_2 + \ldots + c_tP_t \quad (8)$$

where each P_i, $1\le i\le t$, is a permutation matrix*; and c_1, c_2,…, c_t, are positive real numbers and: $c_1+c_2+\ldots+c_t=1$.

*(Permutation matrix is a square matrix that has only "0" and "1" elements, and each row and each column has exactly one "1" element.)

Proof. It is proved by following the Theorem 5.4 in [1]. □

Theorem 2. The number of permutation matrices decomposed in (8) is bounded by $t\le(n-1)^2+1$.

Proof. The proof can be done by following the Theorem 3 in [14]. □

Therefore, when n=m, workload matrix $X_{n\times m}$ can be decomposed into a sequence of schedule matrices:

$$X_{n\times m}=L\times Y_{n\times n}= c_1L\times P_1+c_2 L\times P_2+\ldots+c_t L\times P_t. \quad (9)$$

Furthermore, the total number of sessions decomposed is bounded. Therefore, the optimal lifetime L is achievable in the case of n=m. We will give an efficient decomposition algorithm in section 3.2.3.

3.2.2 General Case n>m

When n>m, matrix $X_{n\times m}$ is no longer a square matrix. The idea of our method is to "fill" matrix $X_{n\times m}$ with some dummy columns to make it a doubly stochastic matrix of order n.

Let $Z_{n\times(n-m)}$ be the dummy matrix, which has $(n-m)$ columns. By appending the columns of the dummy matrix to the right hand side of $X_{n\times m}$, the resulting matrix, denoted by $W_{n\times n}$ is in the form as:

$$W_{n\times n}=\begin{bmatrix} x_{11}x_{12}\ldots x_{1m} & z_{11}z_{12}\ldots z_{1n-m} \\ x_{21}x_{22}\ldots x_{2m} & z_{21}z_{22}\ldots z_{2n-m} \\ \ldots\ldots & \ldots\ldots \\ x_{n1}x_{n2}\ldots x_{nm} & z_{n1}z_{n2}\ldots z_{nn-m} \end{bmatrix}_{n\times n}.$$

To make matrix $W_{n\times n}$ having the feature of (3) and (6), i.e., the sum of each column is equal to the sum of each row, the dummy matrix $Z_{n\times(n-m)}$ should satisfy the following conditions:

1) $R_i^{'} = \sum_{j=1}^{n-m} z_{ij} = L - R_i$, for $\forall i = 1,2,...,n$. (10)

2) $C_j^{'} = \sum_{i=1}^{n} z_{ij} = L$, for $\forall j = 1,2,...,n-m$. (11)

We propose a simple algorithm to compute the dummy matrix $Z_{n\times(n-m)}$. The algorithm starts to assign values to the elements of $Z_{n\times(n-m)}$ from its top-left corner. Let R_i^- and C_j^- record the sum of the remaining undetermined elements of row i and column j, respectively, for i=1,2,..,n and j=1,2,...,n–m. Initially, $R_i^- \leftarrow (L-R_i)$ and $C_j^- \leftarrow L$, where R_i and L are computed from matrix $X_{n\times m}$. The strategy of the algorithm is to assign the remaining sum of the row (or column), as much as possible, to an element without violating conditions (10) and (11), and assign the rest elements of the row (or column) to 0. Then, we move down to the next undetermined element from the top-left of the matrix. For example, we start with z_{11}. Now R_1^- is $(L-R_1)$ and C_1^- is L, i.e., $R_1^- < C_1^-$. Thus, we can assign R_1^- to z_{11}, and assign 0 to the rest of elements of row 1 (so condition (10) is met). Then, C_1^- should be updated to $(C_1^- - z_{11})$, because the remaining sum of column 1 now becomes $(C_1^- - z_{11})$ and this value is used to ensure that condition (11) will be met during the process. Suppose we now come to element z_{ij} (i.e., elements of z_{kl}, for k=1,...,i–1 and l=1,...,j–1, are already determined so far). We compare R_i^- with C_j^-. There are three cases:

1) $C_j^- > R_i^-$: it means z_{ij} can use up the remaining value the sum of row i, i.e., R_i^-. Thus, $z_{ij} \leftarrow R_i^-$ and the rest elements of this row should be assigned to 0. So, all elements of row i have been assigned and condition (10) is met for row i.
2) $R_i^- > C_j^-$: it means z_{ij} can use up the remaining value the sum of column j, i.e., C_j^-. Thus, $z_{ij} \leftarrow C_j^-$ and the rest elements of this column should be assigned to 0, i.e., z_{kj}=0, k=2,3,...,n. By doing so, all elements of column j have been assigned and condition (11) is met for column j.
3) $R_i^- = C_j^-$: we can determine elements in both row i and column j by $z_{ij} \leftarrow R_i^-$ and setting the rest elements in row i and in column j to 0. It is easy to see that condition (10) is met for row i and condition (11) is met for column j.

After determining each row (or column), we need to update C_j^- (or R_i^-), before moving to the next row (or column). Each step, we can determine the elements in one row (or column). This process is repeated until all elements in $Z_{n\times(n-m)}$ are determined. The details of the algorithm are given below.

FillMatrix Algorithm
Input: workload matrix $X_{n\times m}$.
Output: dummy matrix $Z_{n\times(n-m)}$.
Begin
  $R_i^- = L - R_i$, for i = 1 to n;
  $C_j^- = L$, for j = 1 to $n-m$;
  i=1; j=1;
  while ($i \le n$) && ($j \le n-m$) **do**
    if $C_j^- > R_i^-$ **then**
      //determine elements in row i.
      $z_{ij} = R_i^-$;
      z_{ik} = 0, for $k = j+1$ to $n-m$;
      // set the rest of row i to 0.
      $C_j^- = C_j^- - z_{ij}$;
      $i = i+1$;
    else if $\sum_{i=1}^{n} z_{i1} = z_{11} = C_1^- = C_1^{'} > C_j^-$
      //determine elements in column j.
      $z_{ij} = C_j^-$;
      z_{kj} = 0, for $k = i+1$ to n;
      // set the rest of column j to 0.
      $R_i^- = R_i^- - z_{ij}$;
      $j = j+1$;
    else
      //determine elements in both row i and //column j.
      $z_{ij} = R_i^-$;
      z_{ik} = 0, for $k = j+1$ to $n-m$;
      z_{kj} = 0, for $k = i+1$ to n;
      $i = i+1$; $j = j+1$;
    endwhile
End

Theorem 3. For a given workload matrix $X_{n\times m}$, *FillMatrix* Algorithm can compute $Z_{n\times(n-m)}$, such that the square matrix $[X_{n\times m} Z_{n\times(n-m)}]/L$ is a doubly stochastic matrix of order n.

Proof. At the beginning of the *FillMatrix* Algorithm, row sums and column sums of the dummy matrix are initialized, and then the dummy matrix is worked out step by step to satisfy conditions (10) and (11). So we can prove a general case: given row sums $R_i^{'}$ and column sums $C_j^{'}$ of a matrix $Z_{n\times m}$, i=1,2,...,n, j=1,2,...,m, the proposed algorithm can compute all elements z_{ij} that satisfy conditions (10) and (11). We use the induction method to prove the theorem.

1) When n=1, m=1, according to the *FillMatrix* algorithm, since $C_1 = R_1^-$, we have $z_{11} = R_1^- = C_1 = R_1^{'} = C_1^{'}$. The conditions (10) and (11) are both met.
2) We assume when $n \le p-1$, $m \le q-1$, the proposed algorithm can compute $Z_{n\times m}$, such that the conditions (10) and (11) are both met.

3) When $n=p$, $m=q$, according the algorithm, we first compare C_1^- with R_1^-, there are three cases.

a) If $C_1^- = R_1$, then set $z_{11}=R_1$, $z_{1k}=0$, $k=2,3,\ldots,m$ and $z_{k1}=0$, $k=2,3,\ldots,n$. For the row 1 and column 1 where z_{ij} have been determined, we have $\sum_{j=1}^{m} z_{1j} = z_{11} = R_1 = R_1'$ and $\sum_{i=1}^{n} z_{i1} = z_{11} = C_1^- = C_1'$. So the conditions (10) and (11) are both met in row 1 and column 1. The remaining undetermined elements z_{ij}, $i=2,3,\ldots,n$, $j=2,3,\ldots,m$, are in the matrix $Z_{(p-1)\times(q-1)}$. According to assumption 2), the remaining matrix $Z_{(p-1)\times(q-1)}$ can be correctly worked out.

b) If $C_1^- > R_1^-$, then set $z_{11}= R_1^-$, $z_{1k}=0$, $k=2,3,\ldots,m$ and $C_1^- = C_1^- - R_1^-$. For the row 1 where z_{ij} have been determined, we have $\sum_{j=1}^{m} z_{1j} = z_{11} = R_1^- = R_1'$, condition (10) is met. For the column 1 which is updated, we have $C_1^- + z_{11} = C_1'$, it does not violate condition (11). The remaining undetermined elements z_{ij}, $i=2,3,\ldots,n$, $j=1,2,3,\ldots,m$, are in the matrix $Z_{(p-1)\times q}$. We continue run the algorithm to compute the remaining elements in $Z_{(p-1)\times q}$ that satisfies the conditions (10) and (11). Note that C_1^- monotonously decreases after each round of assignment and $\sum_{i=2}^{n} R_i^- = \sum_{j=1}^{m} C_j^- > C_1^-$. There must exist $R_l^- \geq C_1^-$ in round l, we set $z_{l1}= C_1^-$, $z_{lk}=0$, $k=l+1,l+2,\ldots,n$ and $R_l^- = R_l^- - C_1^-$. Then the remaining matrix is $Z_{(p-l+1)\times(q-1)}$. According to assumption 2), the remaining matrix $Z_{(p-l+1)\times(q-1)}$ can be correctly worked out.

c) If $R_1^- > C_1^-$, similar to b), we can prove this case.

4) The proof of cases $n=p$, $m=q-1$ and $n=p-1$, $m=q$ are similar to 3).

Combining 1), 2), 3) with 4), the proposed algorithm can correctly compute all elements in the matrix $Z_{n\times m}$ such that the conditions (10) and (11) are both met.

Theorem 3 is proved. □

Theorem 4. The time complexity of the *FillMatrix* Algorithm is $O(n^2)$.

Proof. It is not difficult to see that the time complexity of the proposed algorithm is $O(n^2)$. Theorem proved. □

Given a workload matrix $X_{n\times m}$, using the proposed algorithm, we can fill the matrix to make it a square matrix $W_{n\times n}$ and $W_{n\times n}$ satisfies conditions (3) and (6). According to the theorems discussed in section 3.2.1, $W_{n\times n}$ can be decomposed as (9):

$$W_{n\times n} = c_1 L\times P_1 + c_2 L\times P_2 + \ldots + c_t L\times P_t.$$

We simply denote $c_i L$ as c_i, $i=1,2,\ldots,t$, because they are constants anyway. Thus,

$$W_{n\times n} = c_1\times P_1 + c_2\times P_2 + \ldots + c_t\times P_t. \quad (12)$$

Let P_i' denote the matrix which contains the first m columns in P_i (i.e., the information for the m valid targets by dropping the $n-m$ dummy columns), $i=1,2,\ldots,t$. We have

$$X_{n\times m} = c_1\times P_1' + c_2\times P_2' + \ldots + c_t\times P_t'. \quad (13)$$

Since P_i is a permutation matrix and P_i' contains the first m columns of P_i, there is exactly one positive number in each column and at most one positive number in each row in P_i'. That is, the matrices $c_i\times P_i'$, $i=1,2,\ldots,t$, are the schedule matrices. In session i, sensors are scheduled to watch their respective targets according to the position of "1" elements in P_i' for the period of c_i time. By following this schedule, the optimal lifetime L of the surveillance system can be achieved.

The above discussions conclude that a workload matrix is decomposable to a sequence of schedule matrices such that the optimal lifetime can be achieved. In the next section, we propose an efficient algorithm that decomposes the workload matrix.

3.2.3. Algorithm for Decomposing Workload Matrix

In this section, we study the details of decomposing workload matrix. The basic idea of the algorithm is to represent the filled workload matrix as a bipartite graph where one side are sensors and the other are targets, and thus the problem of decomposing the filled workload matrix is transformed into the problem of finding perfect matchings in a bipartite graph.

Notice that the workload matrix is already filled with dummy columns as discussed in section 3.2.2. The bipartite graph consists of two set of nodes $S=(s_1, s_2, \ldots, s_n)$ and $T=(t_1, t_2, \ldots, t_m)$, $n=m$, representing sensors and targets respectively. For each non-zero element x_{ij} in the workload matrix, there is an edge from s_i to t_j and the weight of the edge is x_{ij}. The decomposing process is as follows. We compute a perfect matching in the bipartite graph, which has exactly n edges. Let c_i be the smallest weight of the n edges. Deduct c_i from the weight of the n edges in the perfect matching and remove the edge whose weight becomes zero. This operation is repeated until there is no perfect matching can be found in the bipartite graph.

Notice that each perfect matching corresponds to a decomposed schedule matrix P_i in (12), where all elements of this matrix is either 0 or c_i (the weight found in round i) and there is only one non-zero elements in each column and each row. By removing the $(n-m)$ dummy columns in P_i, it becomes a valid schedule matrix.

Because we try to decompose the matrix by using the technique of finding perfect matchings, the questions we have now are:

1) Does it guarantee that there exists a perfect matching in every round of the decomposition process?
2) Can this perfect matching method *exactly* decompose the workload matrix? That is, is it possible that the last round

of the perfect matching will exactly remove all the remaining edges in the bipartite graph?

Theorem 5 and theorem 6 (will appear later) give answers to the above questions, respectively.

Theorem 5. For any square matrix W of nonnegative real numbers, if all row sums and column sums are same, there exists a perfect matching on the corresponding bipartite graph.

Proof. Let L be the sum of all elements in W matrix, and A denote matrix $A=1/L \times W$. It is obvious to see that A is a doubly stochastic matrix. We prove the theorem by contradictory.

If there does not exist perfect matching in the corresponding bipartite graph of A, there does not exist n positive entries with no two of the positive entries on a line (column or row) in A. According to the König theorem [6, 7], we could cover all of the positive entries in the matrix with e rows and f columns, such that $e + f < n$. But, since all of the line sums of A equal to 1, it follows $n \le e + f < n$. This contradicts to the assumption.

Theorem 5 is proved. □

Since in each round i, we deduct c_i from the weight of the n edges in the perfect matching, it is equal to deduct schedule matrix $c_i \times P_i$ from the workload matrix. That is, the row sums still equal the column sums in the workload matrix after this deduction. According to theorem 5, we can guarantee that there exists a perfect matching in every round of the decomposition process.

The next, we propose a simple recursive algorithm for finding a perfect matching in a bipartite graph. Let M denote a set of edges of a perfect matching. We use (s_i, t_j) to denote an edge from S to T and (t_j, s_i) denote an edge from T to S. There is no direction of edges in the graph, but this notation helps to describe the algorithm. The algorithm starts from any edge in the graph. Each time, it tries to find an *M-path* (called augment matching path). An *M-path* is a path in the bipartite graph. It starts with an S node that is not in M and end with a T node that is also not in M, and any edge in the *M-path* from S to T should not be in M and any edge from T to S should be in M. We can see that there are always one more non M-edges than the M-edges in an *M-path* (an M-edge is an edge in M). Thus, by replacing M-edges in the *M-path* by the non M-edges, the number of edges in M is incremented by 1. We keep on finding this *M-path* and increasing the size of M, until a perfect matching is found. For clarity of notation, in the algorithm, "$s_i \in M$" simply means s_i is an end-node of an edge in M and "$s_i \notin M$" means s_i is not an end-node of an edge in M. The detailed algorithm is given as follows.

PerfectMatching Algorithm

Input: a bipartite graph $G=(S \cup T, E)$.

Output: a perfect matching M.

Begin

```
Pick any edge from E and add to M;
while there exists a s_i∈S but s_i∉M do
    // pick up an unmatched node
    M-path = ∅;
    if Find-M-path(s_i) then // an M-path is found
        Remove M-edges in M-path from M and add in
        non M-edges to M;
endwhile
Output the perfect matching M;
End
Find-M-path(s_i) {
    //recursive procedure to find an M-path
    for t_j∈T(s_i) and (s_i, t_j)∉M do
        // try a non M-edge from S to T
        M-path = M-path + (s_i, t_j);
        // grow M-path from S to T
        if t_j∉M then
            return true; // an M-path is found
        else
            for s_k∈S(t_j) and (t_j, s_k)∈M do
                // try an M-edge from T to S
                M-path = M-path + (t_j, s_k),
                // grow M-path from T to S
                if Find-M-path(s_k) then
                // recursive call to find a M-path
                    return true; // an M-path is found
            endfor
            return false;
    endfor
    return false;
}
```

Integrating together with *FillMatrix* Algorithm and *PerfectMatching* Algorithm, we have the algorithm of decomposing the workload matrix as follows.

DecomposeMatrix Algorithm

Input: the workload matrix $X_{n \times m}$.

Output: a sequence of schedule matrices.

Begin

```
if n>m then
    Run FillMatrix Algorithm to obtain a square
    matrix W_{n×n}=X_{n×m};
Construct a bipartite graph G from W_{n×n};
while there exists edges in G do
    Run PerfectMatching Algorithm on G to find a
    perfect matching M;
    Record c_i×P_i;  // c_i: smallest weight in M and
                     // P_i: permutation matrix of M
    Deduct c_i from the weight of edges in M and
    remove edges with weight 0;
endwhile
Output W_{n×n}= c_1P_1 + c_2P_2 +...+ c_iP_i;
```

End

The following theorem claims the correctness of the *DecomposeMatrix* Algorithm.

Theorem 6. The workload matrix can be exactly decomposed into a sequence of schedule matrix by the *DecomposeMatrix* algorithm in $O(|E| \times n^3)$ time, where $|E|$ is the total number of non-zero elements in the filled workload matrix.

Proof. Each time when a perfect matching is found, supposing the corresponding schedule matrix is $c_i \times P_i$, the

workload matrix is subtracted by $c_i \times P_i$. The remaining matrix still satisfies the conditions that its row sum is equal to its column sum. According to theorem 5, a perfect matching can still be found in the graph for the remaining matrix. Therefore, the workload matrix can be decomposed step by step, until finally there is a perfect matching that makes elements in the remaining matrix all "0", after the schedule matrix of the matching is subtracted from the remaining workload matrix. The workload matrix is thus, exactly decomposed by the algorithm.

Furthermore, since each time of finding a perfect matching, at least one edge in the bipartite graph is removed. Therefore, it takes at most $|E|$ number of runs of the perfect matching algorithm, where $|E|$ is the total number of non-zero elements in the filled workload matrix. Since we use depth-first search in the *PerfectMatching* Algorithm, according to [5, 6], it takes $O(n^3)$ to find a perfect matching in each round. Therefore, it totally costs $O(|E| \times n^3)$ time to find all schedule matrices.

Theorem 6 is proved. □

3.3. Obtain Schedule Timetable

We have obtained a sequence of schedule matrices by decomposing the workload matrix. Each schedule matrix specifies sensors watching targets for the same period of time (called a session). In fact, there is no need for all sensors to start watching their corresponding targets at the same time, and switch synchronously to other targets (or switch off) at the end of a session. Each sensor's schedule can be independent from the others. That is, sensors can switch on/off and switch to watch other targets asynchronously from each other. To come up with the target-watching timetable for sensor i, we simply take the i-th row of all the schedule matrices, and combine the time of the consecutive sessions that it watches the same target (in this case there is no need for sensor i to switch). Finally, we have an independent timetable for each sensor.

Since global clock synchronization is achievable in sensor networks by using some localized method [16] or time synchronization scheme [17], sensors can cooperate correctly according to the timetable to achieve the maximal network lifetime.

4. Experiments and Simulations

4.1. A Numeric Example

We randomly place 6 sensors (in clear color in Fig. 2) and 3 targets (in grey in Fig. 1) in a 50×50 two-dimensional free-space region. For simplicity, the surveillance range of all sensors is set to 20 (our solution can work for any system with non-uniform surveillance range). Fig. 1 shows the surveillance relationship between sensors and targets, with an edge between a sensor and a target if and only if the target is within the surveillance range of the sensor. The initial energy reserves of sensors, in terms of hours, are random number generated in the range of [0, 50] with the mean at 25, as shown in Tab. 1.

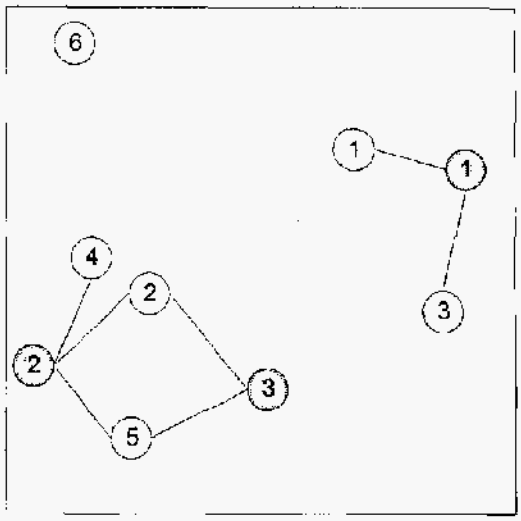

Fig. 1. An example system with 6 sensors and 3 targets.

Tab. 1. The initial energy of 6 sensors (hr.).

Sensors	1	2	3
E_i	15.6926	34.2627	24.8717
Sensors	4	5	6
E_i	21.7847	46.6865	34.5310

We follow the three steps in our method to find the timetables for the sensors.

First, we use the linear programming, described in section 3.1, to compute the maximum lifetime L and the workload matrix that achieves L:

$$L = 40.5643 \text{ hr.},$$

$$X_{6\times3} = \begin{bmatrix} 15.6926 & 0 & 0 \\ 0 & 10.2454 & 18.7199 \\ 24.8717 & 0 & 0 \\ 0 & 17.9125 & 0 \\ 0 & 12.4064 & 21.8444 \\ 0 & 0 & 0 \end{bmatrix}.$$

In the workload matrix, we can see target 1 is only watched by sensors 1 and 3 for 15.6926 hr., 24.8717 hr., respectively. The total time for target 1 to be watched is 40.5643 hr., which is the lifetime of the surveillance system.

Second, we run the *FillMatrix* Algorithm, proposed in section 3.2.2, to append a dummy matrix to the workload matrix to make it a square matrix $W_{6\times6}$, where the sum of each column and the sum of each row are all equal to L:

$$W_{6\times6} = \begin{bmatrix} 15.6926 & 0 & 0 & 24.8717 & 0 & 0 \\ 0 & 10.2454 & 18.7199 & 11.5990 & 0 & 0 \\ 24.8717 & 0 & 0 & 4.0936 & 11.5990 & 0 \\ 0 & 17.9125 & 0 & 0 & 22.6518 & 0 \\ 0 & 12.4064 & 21.8444 & 0 & 6.3135 & 0 \\ 0 & 0 & 0 & 0 & 0 & 40.5643 \end{bmatrix}.$$

Then we run the *DecomposeMatrix* Algorithm, proposed in section 3.2.3, to decompose $W_{6\times6}$ into a sequence of schedule matrices $c_1 \times P_1$, $c_2 \times P_2$, ..., and $c_5 \times P_5$ (i.e., the decomposition terminates at round 5), such that

$$W_{6\times6} = c_1P_1 + c_2P_2 + \ldots + c_5P_5.$$

By removing the dummy columns of the schedule matrices, we have:

$$X_{6\times 4}=\begin{bmatrix}4.0936 & 0 & 0 & 0\\ 0 & 4.0936 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 4.0936 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}+\begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 6.1518 & 0 & 0\\ 6.1518 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 6.1518 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}+\begin{bmatrix}11.5990 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 11.5990 & 0 & 0\\ 0 & 0 & 11.5990 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}+\begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 6.3135 & 0\\ 6.3135 & 0 & 0 & 0\\ 0 & 6.3135 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}+\begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 12.4064 & 0\\ 12.4064 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 12.4064 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}.$$

Finally, we obtain target watch timetables for sensors based on the above schedule matrix. The timetable for the 6 sensors is shown in Tab. 2.

Tab. 2. The schedule timetable for 6 sensors

Sensors	Watching Duty (time duration and watching targets)			
1	0~4.0936 Target 1	4.0936~28.9653 Turn off		28.8953~40.5643 Target 1
2	0~10.2454 Target 2	10.2454~28.9653 Target 3		28.8953~40.5643 Turn off
3	0~4.0936 Turn off	4.0936~28.9653 Target 1		28.8953~40.5643 Turn off
4	0~10.2454 Turn off	10.2454~16.5589 Target 2	16.5589~28.8953 Turn off	28.8953~40.5643 Target 2
5	0~10.2454 Target 3	10.2454~16.5589 Turn off	16.5589~28.8953 Target 2	28.8953~40.5643 Target 3
6	0~40.5643 Turn off			

It is easy to see that the timetable in Tab. 2 satisfies the surveillance conditions that each sensor can watch at most one target at a time and each target is watched by a sensor at anytime.

4.2. Simulations

We conduct some simulations to study the complexity of our proposed solution and compare its performance with a greedy method.

The simulations are conducted in a 50×50 two-dimensional free-space region. Sensors and targets are randomly distributed inside the region. Again, the surveillance range of all sensors is set 20 (except the simulations for Fig. 3(a)). The initial energy reserves of sensors are the random numbers in the range of [0, 50], with the mean value of 25 hours. The results presented in the figures are the means of 100 separate runs.

A. Growth of decomposition steps is linear

According to Theorem 2, we know the number of steps for decomposing the workload matrix, denoted by t, is bounded by $t\le(n-1)^2+1$. In the simulations, we found that t is linear to the size of system.

Fig. 2(a) and Fig. 2(b) show the increase of t versus the change of N (number of sensors) and M (number of targets), respectively, when one of the two variables is fixed. From the figures, we can see a strong linear relationship between t and N (or M). This result tells us that the actual number of steps for decomposing the matrix is linear to the size of system in real runs.

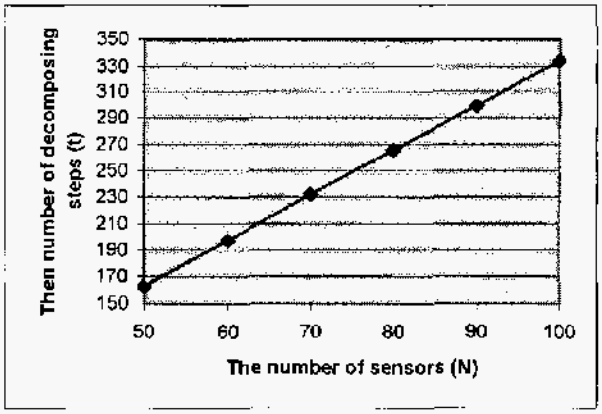

Fig. 2(a). t versus N when M=10.

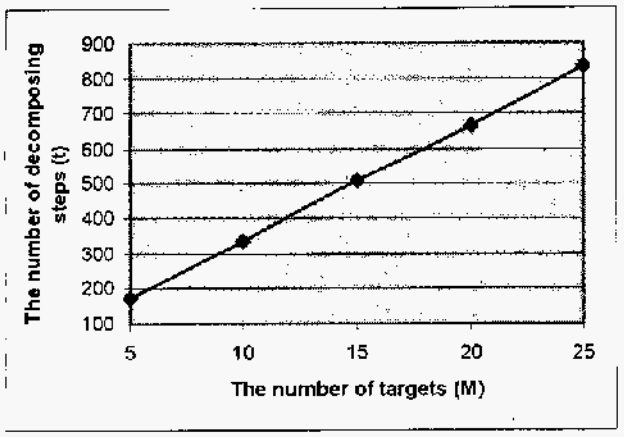

Fig. 2(b). t versus M when N=100.

B. Comparison with a greedy method

A greedy algorithm is proposed to compare the performance with our optimal solution. The basic idea of the greed method is to allocate sensors to targets in such a way that each sensor is allocated to watch one target in its lifetime and the total working time of the sensors allocated to targets are balanced as much as possible. It first assigns the sensors that have only one target in their surveillance range to their respective targets. Then, the remaining sensors are assigned to the targets such that the total time for targets being watched is as balanced as possible among all targets.

We set N=100 and M=10. Fig. 3(a) shows the lifetime versus the change of surveillance range of sensors. From Fig. 3(a), we can see that when the surveillance range is small, two curves are very close. This is because with a small surveillance range, sensors usually have got only one target within its range. There is hardly any room that our optimization method can take advantages. As the surveillance range becomes larger, more sensors are able to cover multiple targets, which gives our method more room to schedule the sensors properly to achieve the maximum lifetime. That is why the performance gap between the two methods becomes more significant as the increase of the surveillance ranges.

Fig. 3(b) shows the lifetime versus the number of sensors placed in the same region. This simulation shows how the lifetime is affected by the density of sensors. Fig. 3(b) exhibits the similar trend as in Fig. 3(a). As more sensors deployed in the same region, the density becomes higher. A target can be watched by more sensors and there is a higher chance for a target to be in the watching range of multiple sensors. Thus, our optimal algorithm can take more advantages by optimizing the schedule and the performance becomes more significant than the greedy method in this kind of situations.

From Fig. 3(a) and Fig. 3(b), we can conclude that our optimal algorithm has significantly better performance in the situation where sensors have larger coverage range or senses are densely deployed.

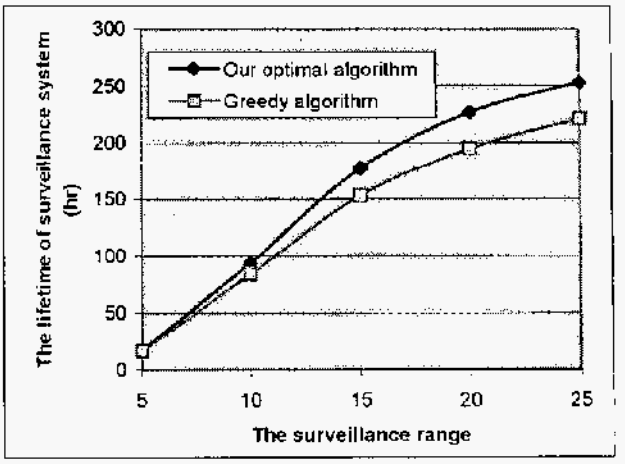

Fig. 3(a). Lifetime versus surveillance range.

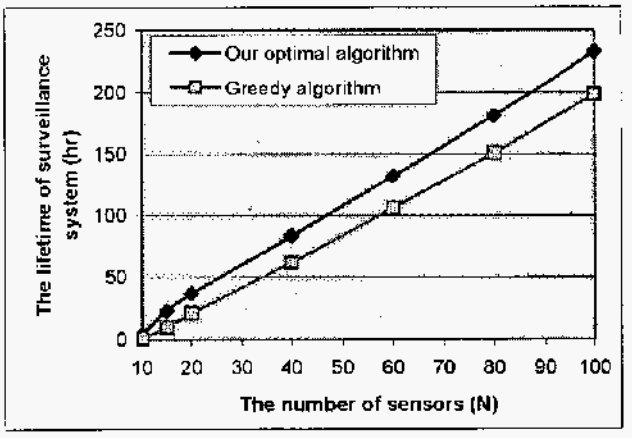

Fig. 3(b). Lifetime versus N when M=10.

5. Conclusions

This paper addressed the maximal lifetime scheduling problem in sensor surveillance networks.

Our solution consists of three steps: 1) compute the maximum lifetime of the system and the workload matrix by using linear programming method; 2) decompose the workload matrix into a sequence of schedule matrices by using perfect matching method. This decomposition can preserve the maximum lifetime; 3) obtain target watching timetable for sensors. It is not difficult to see that our solution is the optimum in the sense that it can find the schedules for sensors watching targets that achieve the maximum lifetime. Simulations have been conducted to show that the steps of decomposition is linear to the size of system and our method can take more advantages in the situation that senses are densely deployed or sensors have larger coverage ranges.

Acknowledgment

We would like to thank Professor Dingzhu Du and Professor Xiaotie Deng for pointing us towards relevant results on decomposing of doubly stochastic matrices.

REFERENCES

[1] Herbert John Ryser, *Combinational Mathematics*, The Mathematical Association of America, pp58-59, 1963.

[2] Richard A. Brualdi and Herbert J. Ryser, *Combinatorial Matrix Theory*, Cambridge University Press, pp9-10, 1991.

[3] D. Tian and N. D. Georganas, "A Coverage-Preserving Node Scheduling Scheme for Large Wireless Sensor Networks" In *First ACM International Workshop on Wireless Sensor Networks and Applications*, pp32-41, 2002

[4] C.-Y. Chong and S. P. Kumar, "Sensor Networks: Evolution, Opportunities, and Challenges", *Proc. of the IEEE*, pp1247-1256, Vol 91, No. 8, Aug. 2003.

[5] Ronald Gould, *Graph Theory*, The Benjamin/Cummings Publishing Company, INC, pp198-209, 1988.

[6] Douglas B. West, *Introduction to Graph Theory*, Prentice Hall, INC, pp109-111, 1996.

[7] S. Axler, F. W. Gehring and K. A. Ribet, *Graph Theory*, Second Edition, Springer, 2000.

[8] Jean Carle and David Simplot-Ryl, "Energy-Efficient Area Monitoring for Sensor Networks", *IEEE Computer*, pp40-46, Vol. 37 No. 2, Feb. 2004.

[9] Linnyer B. Ruiz, Luiz Filipe Menezes Vieira, Marcos Augusto Menezes Vieira *et al*, "Scheduling Nodes in Wireless Sensor Networks: a Voronoi Approach", *Proceedings of 28th IEEE Conference on Local Computer Networks (LCN2003)*, October 2003, pages 423-429, Bonn/Konigswinter, Germany.

[10] Chih-fan Hsin and Mingyan Liu, "A Distributed Monitoring Mechanism for Wireless Sensor Networks", *International Conference on Mobile Computing and Networking Proceedings of the ACM Workshop on Wireless Security*, Atlanta, USA, pp57-66, 2002.

[11] Y. Zhao, R. Govindan and D. Estrin, "Residual Energy Scans for Monitoring Wireless Sensor Networks", *IEEE Wireless Communications and Networking Conference*, pp356-362, 2002.

[12] M. Bhardwaj, T. Garnett and A. Chandrakasan, "Upper Bounds on the Lifetime of Sensor Networks", In *IEEE International Conference on Communications*, pp785-790, 2001.

[13] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler and John Anderson, "Wireless Sensor Networks for Habitat Monitoring", In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pp88-97, Atlanta, USA, September 2002.

[14] T. Inukai, "An Efficient SS/TDMA Time Slot Assignment Algorithm", *IEEE Trans. Commun.*, vol. COM-27, pp1449-1455, 1979.

[15] Ting Yan, Tian He and John A. Stankovic, "Differentiated Surveillance for Sensor Networks", *Proceedings of the First International Conference on Embedded Networked Sensor Systems*, Los Angeles, USA, pp51-62, 2003.

[16] Qun Li, Daniela Rus, "Global Clock Synchronization in Sensor Networks", *IEEE INFOCOM* 2004.

[17] K. Römer, "Time Synchronization in Ad Hoc Networks", in *Proc. of ACM Mobihoc*, Long Beach, CA, 2001.