

# A Distributed Algorithmic Framework for Coverage Problems in Wireless Sensor Networks \*

Akshaye Dhawan and Sushil K. Prasad  
Department of Computer Science  
Georgia State University  
Atlanta, Ga 30303  
akshaye@cs.gsu.edu, sprasad@gsu.edu

## Abstract

*One of the key challenges in Wireless Sensor Networks (WSNs) is that of extending the lifetime of the network while meeting some coverage requirements. In this paper we present a distributed algorithmic framework to enable sensors to determine their sleep-sense cycles based on specific coverage goals. The framework is based on our earlier work on the target coverage problem. We give a general version of the framework that can be used to solve network/graph problems for which melding compatible neighboring local solutions directly yields globally feasible solutions. We also apply this framework to several variations of the coverage problem, namely, target coverage, area coverage and  $k$ -coverage problems, to demonstrate its general applicability. Each sensor constructs minimal cover sets for its local coverage objective. The framework entails each sensor prioritizing these local cover sets and then negotiating with its neighbors for satisfying mutual constraints. We introduce a dependency graph model that can capture the interdependencies among the cover sets. Detailed simulations are carried out to further demonstrate the resulting performance improvements and effectiveness of the framework.*

## 1. Introduction

Wireless sensor networks(WSNs) have attracted a lot of recent research interest due to their applicability in security, monitoring, disaster relief and environmental applications. WSNs consist of a number of low-cost sensors scattered in a geographical area of interest and connected by a wireless RF interface. Sensors gather information about the monitored area and send this information to gateway nodes [1].

\*We will maintain additional information on this framework at <http://www.cs.gsu.edu/~dimos/framework.html>

In order to keep their cost low, the sensors are equipped with limited energy and computational resources. The energy supply is typically in the form of a battery and once the battery is exhausted, the sensor is considered to be dead. To ensure that the area or targets of interest can be covered, sensors are usually deployed in large numbers by randomly dropping them in this region. Such a deployment scheme gives rise to an overlap in the monitoring regions of individual sensors.

A simplistic approach to meet the coverage objective would be to turn on all sensors after deployment. But this needlessly reduces the *lifetime* of the network since the overlap between monitoring regions implies that not all sensors need to be on at the same time. In order to extend the lifetime of a sensor network while maintaining coverage, a minimal subset of the deployed sensors are kept active while the other sensors can sleep. Through some form of scheduling, this active subset changes over time until there are no more such subsets available to satisfy the coverage goal. In using such a scheme to extend the lifetime, the problem is two fold. First, we need to select these minimal subsets of sensors. Then there is the problem of *scheduling* them wherein, we need to determine how long to use a given set and which set to use next. For an arbitrarily large network, there are exponential number of possible subsets making the problem intractable and it has been shown to be NP-complete in [5, 9].

Existing centralized and distributed heuristics for arriving at a lifetime extending schedule are discussed in Section 6. Centralized solutions like those in [5, 13] are based on assuming that the entire network structure is known at one node (typically the gateway node), which then computes the schedule for the network. The schedule is computed using *linear programming* based algorithms. Like any centralized scheme, it suffers from the problems of scalability, single point of failure and lack of robustness. The later is particularly relevant in the context of sensor networks since sensor

nodes are deployed in hostile environments and are liable to failure. Existing distributed solutions in [2, 4, 14] work by having a sensor exchange information with its neighbors (limited to  $k$ -hops). These algorithms use information like targets covered and battery available at each sensor to greedily decide which sensors remain on. This happens in rounds so the set of active sensors is periodically reshuffled. The problem with these algorithms is that they use simple greedy criteria to make their decision and do not efficiently take into account the problem structure.

**Our Contributions:** This paper generalizes the concepts introduced in [12]. We realize that the scheme of creating local solutions and modeling their dependencies applies to various other problems besides target coverage. In general, for certain graph and network problems where solving the problem locally yields a globally feasible solution, our framework can be used (See Section 2). In this paper we use the solution in [12] to develop a framework and show its application to the area coverage and  $k$ -coverage problems besides showing how the target coverage solution fits into this generalized framework. The key points of the framework include construction of local solutions, modeling the dependency between the local solutions using a *dependency* graph, prioritizing the interdependent local solutions using a *priority* function that can utilize the dependency graph and negotiating with neighbors to arrive at a mutually satisfactory local solution.

In [12] we focused on the target coverage problem and presented distributed algorithms for scheduling the sensors to extend the lifetime. We used the idea of constructing *local* cover sets consisting of sensors that can cover local targets. Since certain cover sets are *better* than others, we presented a *Lifetime Dependency Graph* model that enabled us to use some properties of this graph in order to prioritize these covers. We also showed how other existing algorithms like [2] [4] can be modeled by variations in the prioritizing scheme. We carried out simulations to show improvements of 10-20% over LBP [2] and similar performance to DEEPS [4] which is a 2-hop algorithm. A 2-hop version of our algorithm outperformed DEEPS [4]. The remainder of this paper is organized as follows. In section 2 we introduce the generic framework, its steps and the problems to which it applies. In section 3 we develop a coverage-problem specific framework. Section 4 discusses the solution of the three different coverage problems area, target and  $k$ -coverage within the developed framework for coverage problems. In Section 5 we evaluate our algorithms against those of [2] [4]. Section 6 surveys previous work on the lifetime for coverage problem. Finally, we conclude in Section 7.

## 2 The Framework

Our framework applies to a class of graph and network problems wherein the locally compatible solutions can be melded in a distributed fashion to yield a globally feasible solution. For most sensor coverage problems, local covers when combined together yield global covers. On the other hand, local spanning subtrees do not seem to be easily meldable distributively to construct a global spanning tree.

For such class of problems, even if it is intractable to find globally-optimal solutions, it may be tractable to find locally-optimal solutions, since the problem size is much smaller. For example, all possible covers of local targets of a sensor can be efficiently found for bounded sensing range. These local solutions often are interdependent, i.e., using one may impact a subsequent use of others. For example, in sensor coverage problems, using one cover set may reduce the lifetime of those covers which share sensors with the first. This is problematic if a series of solutions are needed.

Our overall framework is a two phase distributed meta-algorithm. The first phase is the setup phase for each node to construct prioritized local solutions. The second phase is the rounds of negotiation phase during which each node chooses its best local solution compatible with its neighbors. Employing the framework entails the following: (i) verifying the applicability of the framework, (ii) modeling the state space of local solutions and their interdependencies using a dependency graph structure, (iii) heuristically modeling the priority of local solutions based on the properties of the dependency graph structure, and (iv) determining the logistics of negotiating with neighbors to settle on mutually-compatible and high-priority local solutions.

In this section we provide an overview of the principles of the generic framework. The details as applied to the coverage problem are given in Section 3. Hence, in the discussion that follows, we describe the four phases in the order in which they are used.

**1. Applicability of the framework:** The problem being solved must have the property that compatible local solutions of neighbors when combined give a globally feasible solution. For example, for the target coverage problem, if for each sensor, all local targets have been covered, this implies that all targets have been covered globally also. Hence, solving the problem locally for every node gives a globally valid solution. This step also establishes criteria for checking mutual compatibility of local solutions.

For an example of a problem that *cannot* be solved using this framework, consider constructing a spanning tree. Here, the local solution would be a spanning tree connecting a sensor to its neighborhood. However, these local trees when combined, may have cycles and do not imply that a global tree is formed. Of course, melding these subtree edges in a reduction-tree fashion rejecting those edges

which cause a cycle will yield a global spanning tree [7]. However, that needs much more communication than what can be efficiently done distributively. Hence this property of all local solutions yield a globally feasible solution is imperative for our framework.

**2. Modeling the local solutions and their interdependencies:** This step starts with modeling the local solutions for the given problem. In some cases it may be possible to compute all solutions, whereas for others we would need a representative sample to model the state space. For many problems, these local solutions are not independent of each other. For example, in the target coverage problem, for a given sensor, there can be a number of different subsets of neighbors that cover the local targets being considered. Since these sets are not disjoint, using one set drains the lifetime of another set. To account for this, the framework envisions a graph model to capture these dependencies among the local solutions. Note the specifics of this graph would depend on the problem being considered. We call this a *Dependency Graph* (See Section 3.2). This is a key contribution of our framework. Instead of a simple greedy heuristic to choose the best solution, we consider a solution with relation to its impact on other possible solutions and use this as a criterion to assess a solution's quality.

**3. Prioritize the local solutions:** Each node needs to decide which of the possible local solutions to use. Based on the dependency structure of these local solutions and other problem specific metrics, a *priority* function can be defined that measures the quality of a local solution. Including the dependency information of the graph into this heuristic priority function gives us a way to account for the problem structure.

**4. Negotiate with neighbors for mutually satisfying solutions:** This step involves deciding the details of communication related logistics for the 2-phase algorithmic framework consisting of setup and negotiation phases. The setup phase is usually a round of information exchange with 1- or 2-hop neighbors followed by construction of the local solutions and the dependency graphs structure, and calculation of the priorities of the local solutions. In the negotiation phase, a node communicates with its neighbors and based on both its preference and those of its neighbors, picks a solution to use. Again, the nature of this step would be problem dependent and we explore this with respect to the target coverage problem in the next section. Through its negotiation phase, a node attempts to locally satisfy the twin goals of feasibility and local optimality. Also note that the problem may require several rounds of negotiation to arrive at a mutually acceptable solution.

### 3 Developing the Framework for Coverage Problems

In this section we define the steps of our Framework presented in Section 2 as applied to different coverage problems. The specifics for each of the three coverage problems - area, target and  $k$ -coverage are defined in the next section. We begin with some definitions. Section 3.2 then lays out the framework.

#### 3.1 Definitions

*The Network Graph:* A common representation of a sensor network is that of using a graph to model the connectivity of the network. The network can be represented using a graph  $G = (V, E)$  where,  $V = \{s_1, s_2, \dots, s_n\}$  is the set of sensors and an edge  $e = (s_i, s_j) \in E$ , if and only if sensor  $s_i$  is in communication range of sensor  $s_j$ . In a network with fixed communication ranges for each node, this model becomes a *Unit Disk Graph* (UDG).

$b(s)$ : the battery available at sensor  $s$ .

*Cover C:* In general, by a cover set we are referring to a minimal subset of sensors that meets some coverage objective. The objective here depends on the coverage problem being solved.

$lt(C) = \min_{s \in C} b(s)$ , the maximum lifetime of a cover  $C$ . The sensor  $s$  with minimum battery is known as the *bottleneck* sensor of the cover  $C$ .

#### 3.2 The Framework for Coverage Problems

##### 3.2.1 Applicability of Framework

For coverage problems, if every sensor ensures that its local coverage objective (local targets/local area) is satisfied then this implies that the global coverage objective has also been met. Also for the coverage problem, local solutions are always compatible with each other since a sensor's local solution does not invalidate that of another sensor.

##### 3.2.2 Modeling the local solutions and their Dependencies - The Lifetime Dependency (LD) Graph

We approach this problem by focusing on the local neighborhood of a sensor. Each sensor constructs all possible local covers that satisfy its local coverage objective. In the local 1-hop neighborhood, the number of local covers (where a cover could be for area or targets, see Section 3.1) is usually small. This allows individual sensors to distributively construct local minimal cover sets. A sensor can construct its local covers by considering one-hop neighbors it

can communicate to while trying to meet its coverage objectives. For a better decision, it can also consider all neighbors up to two hops and their targets at a slightly increased communication cost.

When two cover sets have some sensors in common, they have some *dependency* on each other because using one cover set now drains the battery of the sensors it shares with the other cover set. This is important because when we pick cover sets to use in a schedule, we should take into account this influence that they have on future cover sets in the schedule. To account for this we define the lifetime dependency graph as follows.

Every node in the LD graph represents a local cover and an edge between two nodes implies that the two covers share some sensors in common. The weight of an edge is given by the life of the *weakest* sensor in this set of common sensors that the edge represents. Finally, we define the *degree* of a cover  $C$  as the sum of the weights of all edges incident on that node. More formally, the local lifetime dependency graph can be defined as a weighted graph,  $G' = (V', E')$  where,  $V' = \{C_1, C_2, \dots, C_k\}$  and each member  $C_i \in V'$  represents a local cover meeting the local coverage objective, and an edge  $e' = (C_i, C_j) \in E'$ , if and only if  $C_i \cap C_j \neq \emptyset$ . We also define weights on the nodes and the edges as follows:

- $w(e')$ : Let  $e' = (C_i, C_j)$ , and  $I = C_i \cap C_j$ . Then,  $w(e') = \min_{s \in I} b(s)$ .
- $d(C) = \sum_{e' \in E' \text{ and incident to } C} w(e')$ , the weight or *degree* of a cover  $C$ .

The reasoning behind this definition of the edge weight comes from considering a simple two node LD Graph (see Fig. 1) with two covers  $C_1$  and  $C_2$  sharing an edge  $e$ . The lifetime of the graph is given by,

$$L \leq \min(lt(C_1) + lt(C_2), w(e))$$

This indicates that for

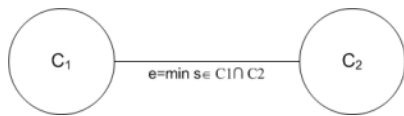


Figure 1. A Simple 2-node LD Graph

the two node graph, the lifetime is bounded by either the sum of the individual lifetimes of the two covers  $C_1$  and  $C_2$  or by the lifetime of the smallest common sensor in  $C_1 \cap C_2$  (given by  $w(e)$ ). Note that if the two sets  $C_1$  and  $C_2$  were disjoint then the lifetime of the graph would have been  $lt(C_1) + lt(C_2)$ . The fact that they share some nodes in common implies that using one cover set also reduces the lifetime of the sensor it has in common with the other cover set. Hence, for the non-disjoint case, the bound on the graph could also be the smallest common sensor between these covers.

Similarly, the reasoning behind the definition of the *degree* of a cover  $C$  is that by summing the weights of all the edges incident on the cover  $C$ , we are getting a measure of the *impact* it would have on *all* other covers with which it shares an edge.

### 3.2.3 Prioritize the local solutions

Initially, each sensor  $s$  communicates with its neighbors and exchanges information on available battery  $b(s)$  and the region (area or targets) it can cover. We will discuss the specific message exchanges in Section 4. Based on this information, sensor  $s$  can compute all the local covers for its local objective. Each sensor then constructs a local LD graph  $G' = (V', E')$  for these covers, and calculates the degree  $d(C)$  of each cover  $C \in V'$  in  $G'$ .

A *priority function* can be defined to prioritize the local covers. We base the priority of cover  $C$  on its degree  $d(C)$  in the lifetime dependency (LD) graph. A lower degree is better since this corresponds to a smaller impact on other covers. If the degree is the same for two or more covers, ties are broken by using (i) the cover with a longer lifetime, (ii) the cover with fewer sensors remaining to be turned on (See the next step), (iii) the cover with the smaller sensor id.

### 3.2.4 Negotiate solutions

The algorithm for a sensor to negotiate its solutions with that of its neighbors operates in rounds. Each round consists of two phases. Phase 1 is the setup phase as described above. In Phase 2, a sensor arrives at an on-off decision based on the messages it receives. The operation in rounds is very similar to other distributed algorithms.

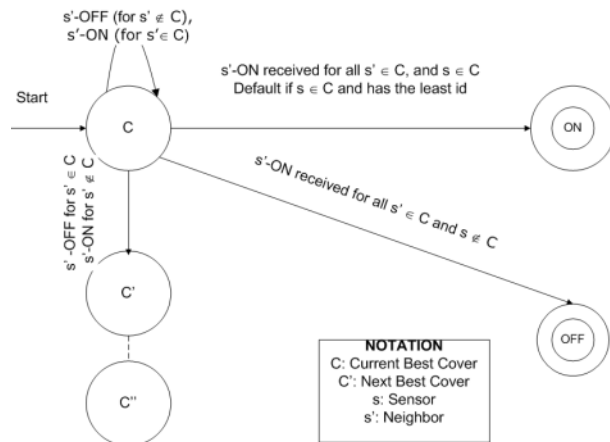


Figure 2. The state transitions to decide the On-Off Status

After calculating the priority function, each sensor now

has an ordering of its local cover sets in terms of preference. The goal is to try and satisfy the highest priority cover. However, a cover comprises of multiple sensors and if one of these switches off, this cover cannot be satisfied. Hence, each sensor now uses the automaton in Fig.2 to decide whether it can switch off or if it needs to remain on.

The automaton starts with every sensor  $s$  in its highest priority cover  $C$ . The sensor  $s$  keeps trying to satisfy this cover  $C$  and eventually if the cover  $C$  is satisfied, then  $s$  switches on if  $s \in C$  or,  $s$  switches off if  $s \notin C$ . Note that a cover  $C$  is considered satisfied if every sensor  $s \in C$  has switched on. If a cover  $C$  cannot be satisfied, then the sensor  $s$  transitions to its next best priority cover  $C'$ ,  $C''$  and so on, until a cover is satisfied.

The transitions of the automaton are outlined below. - Continue with the best cover  $C$ : Sensor  $s$  continues with its current best cover  $C$  if its neighbor  $s' \notin C$  goes off (since  $s'$  does not affect  $C$ ) or if neighbor  $s' \in C$  becomes on (thus improving chances for  $C$ ). -To on/sense status: If all the neighboring sensors in cover  $C$  except  $s$  become on, and  $s \in C$ ,  $s$  switches itself on satisfying the cover  $C$  for its neighborhood, and sends its on-status to its neighbors. -To off/sleep status: If all the neighboring sensors in cover  $C$  become on thus satisfying  $C$ , and  $s \notin C$ ,  $s$  switches itself off, and sends its off-status to its neighbors. -Transition to the next best cover  $C'$ : Sensor  $s$  transitions to the next best priority cover  $C'$ , if (i)  $C$  becomes infeasible because a neighboring sensor  $s' \in C$  has turned off, or (ii) priority of  $C$  is now lower because a sensor  $s' \notin C$  has turned on causing another cover  $C'$ , with same degree and lifetime as  $C$ , with fewer sensors remaining to be turned on.

Note that the automata for negotiation is simple because for coverage problems, we are not negotiating for mutual compatibility/feasibility. The only concern of the negotiation phase here is that of local optimality.

## 4 Applying the Coverage Framework

In this section we extend our framework to show how it can be applied to the Target, Area, and  $k$ -coverage problems.

### 4.1 Target Coverage

In the target coverage problem, we are given a set of targets  $T = \{t_1, t_2, \dots, t_m\}$  that are scattered around a region  $R$ . These targets are considered to be stationary. The objective of the problem is to monitor all targets in  $T$  while maximizing the lifetime of the network. In addition to the previous definitions, we define:  $T(s)$ : The set of targets that sensor  $s$  can sense,  $N(s, k)$ : The set of neighbors of sensor  $s$  at no more than  $k$  hops from  $s$ . This set is *closed* i.e. it includes the sensor  $s$ .

Using the framework defined in Section 3, the phases of the algorithm can be specified as follows. Note that we consider a 1-hop neighbor set i.e.  $N(s, 1)$ . This can easily be extended to more hops at a larger communication cost.

*Compute, Weight and prioritize local solutions:* Each sensor  $s$  communicates with each of its neighbor  $s' \in N(s, 1)$  exchanging locations, battery levels  $b(s)$  and  $b(s')$ , and the targets covered  $T(s)$  and  $T(s')$ . Then it finds all the local covers using the sensors in  $N(s, 1)$  for the target set being considered. The latter can be solely  $T(s)$  or could also include  $T(s')$  for all  $s' \in N(s, 1)$ . It then constructs the local LD graph  $G = (V, E)$  over those covers, and calculates the degree  $d(C)$  of each cover  $C \in V$  in the graph  $G$ .

*Negotiate solutions:* This round essentially remains the same with each sensor using the automata of Fig.2. Each cover set  $C$  in the automata is a set of sensors that can cover the target set being considered.

*Message and Time Complexity:* If the maximum degree  $\Delta$  is assumed to be a constant, the message complexity is also a constant. If  $\tau$  is the maximum number of targets in any sensors neighborhood, it can be shown that the time complexity is given by  $O(\Delta^\tau)$ . See [12] for more details.

### 4.2 Area Coverage

For the area coverage problem, we are given a region  $R$  and the goal is to have this region completely covered at all times by active sensors.

In order to formulate the area coverage problem in our framework, consider any individual sensor  $s$ . The area covered by this sensor can be represented as a disk with the sensor at the center. The coverage objective of this sensor is to ensure that the area within its coverage disk is completely covered at all times. Now, certain parts of this disk are covered by different sensors in  $N(s, 1)$ . Hence, a cover set is any set of sensors in  $s' \in N(s, 1)$  that together cover this entire area.

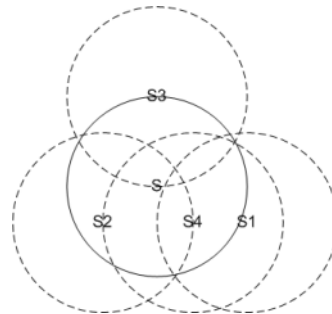


Figure 3. Example for area coverage

An example is shown in Figure 3. For the sensor  $s$ , the following set covers are possible for its area,

$\{s\}$ ,  $\{s_1, s_2, s_3\}$  and  $\{s_2, s_3, s_4\}$ . In order to compute what part of its area is covered by its neighbors, each sensor  $s$  exchanges its location and its battery information with its neighbors in  $N(s, 1)$ . To determine what part of its area is covered by its neighbors, we use the method described in [14] to determine sponsored coverage. Once sponsoring information has been determined, a sensor can compute all cover sets for its area. The weighting, prioritizing and negotiating phases of the framework then follow as before.

Alternatively, [13] defines a field as a set of points that are covered by the same set of sensors. They then discretize the area into a grid. Once points have been grouped into fields, all that is needed is to ensure that all fields are covered. Hence, each field corresponds to a *virtual target* and the problem can effectively be reduced to that of target coverage. We experiment with both the field based method and the direct formulation described above in Section 5. A different approach is taken in [3], where the authors use the idea of covering all the faces of a graph. This avoids having to define the granularity of a grid.

### 4.3 $k$ -Coverage

The  $k$ -Coverage problem can be defined for either target or area coverage. Here, we discuss it in the context of target coverage. Let  $T = \{t_1, t_2, \dots, t_m\}$  be the set of targets scattered around a region  $R$ . The goal is to ensure that for every  $t \in T$  at least  $k$  sensors cover  $t$  at all times. Note that  $k \leq \delta$ , where  $\delta$  is the minimum number of sensors covering any target  $t \in T$ . See [15] [11] [18] for more details.

The extension of our framework to the  $k$ -Coverage problem is straightforward. All we need to do is to ensure that every local cover  $C$  constructed during the initial setup phase is a  $k$ -Cover. To construct local covers that are  $k$ -covering, each sensor picks  $k$  neighbors for every target in  $T(s)$ . By imposing this restriction on the local covers, we can ensure that globally, every target  $t$  is  $k$ -Covered.

## 5 Simulation Results

In this Section, we evaluate the performance of our coverage algorithms as compared with LBP [2] and DEEPS [4]. The simulations were programmed using C++. A static network of sensors is used. For the target coverage problem, the targets are considered to be static also. For the area coverage problem, we use both our direct formulation and the concept of *fields* (Section 4.2) to transform the area coverage problem into the target coverage problem. Then the same algorithms are applied with these *virtual targets*. The  $k$ -coverage problem is also simulated with respect to target coverage. However, since LBP and DEEPS are not extended to solve this problem, we cannot compare our results to theirs for  $k$ -coverage.

We consider sensors scattered randomly in a 100m x 100m area. It is also assumed that the communication range of each sensor is twice the sensing range. We consider two different energy models. In the *linear* model, the energy required to sense a target at a distance  $d$  is a function of  $d$ . In the *quadratic* model, the energy needed to sense the same target is a function of  $d^2$ . For the target coverage prob-

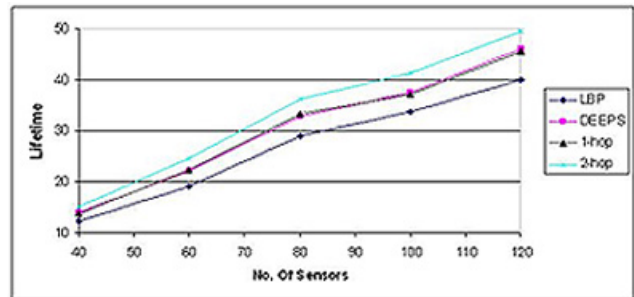


Figure 4. Lifetime with 25 Targets, Linear Energy Model

lem, we consider 25 and 50 targets, randomly deployed in the area. This is the same as considered in [2] and [4]. We vary the number of sensors in steps of 20. The results for 25 targets using a linear energy model are shown in Fig.4. The same experiments are also carried out using the quadratic energy model. A snapshot of the results are shown in Fig.5. As can be seen from these results, the basic 1-hop heuristic outperforms LBP and is very similar to DEEPS (which uses 2-hop information) in performance.

We also consider a 2-hop version of our heuristic. Here, the set of neighbors is defined as  $N(s, 2)$ . Also, for each sensor the target set  $T(s)$ , is defined as all the targets of its one-hop neighbors and itself. If we compare the 2-hop version of our heuristic against DEEPS, it is superior. In order

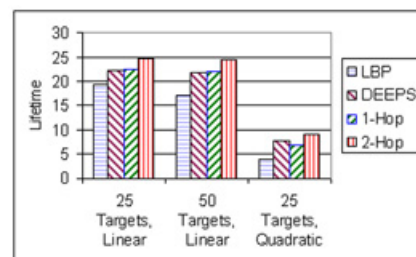
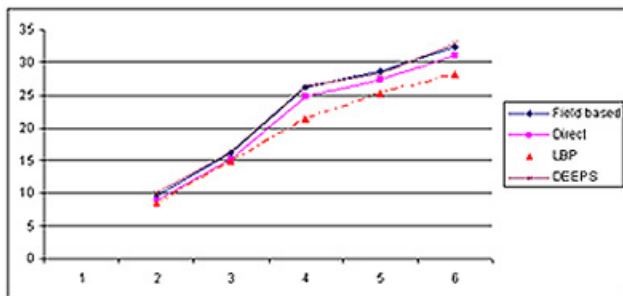


Figure 5. Lifetime with 60 sensors, energy model and number of targets varying

to compute all covers, each sensor maintains a list of all local targets and the neighbors that cover them. This is stored in a matrix where the rows represent neighbor sensors in

$N(s, 1)$  and the columns targets in  $T(s)$ . This matrix is bounded since, by considering only one or two hop neighbors, we have restricted its size. Iterating over the different row combinations that cover all columns of this matrix gives us all possible covers. Since the number of local targets is limited, the number of columns of this matrix are also limited.

For the area coverage problem, once again sensors are scattered randomly in a 100m x 100m area. However, now the objective is to continuously monitor the area. We apply the same decomposition of a given graph into fields for the field based formulation. A virtual target corresponding to each field is considered. By covering each of these virtual targets, we ensure that all the fields and hence, the whole area is covered. We also use a direct formulation as explained in Section 4.2. The results are shown in Figure 6. Once again, a similar trend to the target coverage problem is observed with the 1-hop version outperforming LBP and being very similar to DEEPS. The field based decomposition yields slightly better results because of the loss in sponsored area calculation of the direct method as explained in [14].



**Figure 6. Area coverage with Number of Sensors varying, Linear Energy Model**

## 6 Related Work

In this Section we look briefly at existing approaches to maximizing the lifetime. [6] gives a more detailed survey on the various coverage problems and the scheduling mechanisms they use. We end this section by focusing on two algorithms, LBP [2] and DEEPS [4], since we use them for comparisons against our algorithms in Section 5.

[13] considers the area coverage problem and introduces the notion of *fields* (See Section 4.2 for more details). In [14], the authors give a distributed and localized algorithm that works in rounds, with a scheduling phase followed by a sense phase. Nodes check to see if the area that they cover can be *sponsored* by their neighbors. A probing based distributed algorithm is given in [17]. [16] divides the network into a grid and then selects one active sensor per grid square.

In [3], both centralized and distributed algorithms are provided for working with non-disjoint cover sets. The authors provide a clear problem statement of the Sensor Network Lifetime Problem (SNLP). Their centralized solution is formulated as a packing Linear Program (LP). Using the  $(1 + \epsilon)$  Garg-Könemann approximation algorithm [10], they provide a  $(1 + \epsilon)(1 + 2l/n)$  approximation of the SNLP problem. A similar problem is solved for sensors with *adjustable* ranges in [8].

To solve the target coverage problem, where a set of targets must be monitored by at least one sensor at any point of time, [5] considers the disjoint cover set approach. They prove that the problem is NP-complete and reduce it to the maximum flow problem. Modelling their solution as a Mixed Integer Program shows an improvement over [13].

[15] was the first work to consider the  $k$ -coverage problem. [11] also addresses the  $k$ -coverage problem from the perspective of choosing enough sensors to ensure coverage. Authors consider different deployments with sensors given a probability of being active and obtain bounds for deployment. [18] solves the problem of picking minimum size connected  $k$ -covers.

Now, we look at the two protocols that we compare our heuristics against. The load balancing protocol (LBP) [2] is a simple 1-hop protocol which works by attempting to balance the load between sensors. Sensors can be in one of three states sense/on, sleep/off or vulnerable/undecided. Initially all sensors are vulnerable and broadcast their battery levels along with information on which targets they cover. Based on this, a sensor decides to switch to off state if its targets are covered by a higher energy sensor in either on or vulnerable state. On the other hand, it remains on if it is the sole sensor covering a target. This is an extension of the work in [3]. LBP is simplistic and attempts to share the load evenly between sensors instead of balancing the energy for sensors covering a specific target.

The other protocol we consider is DEEPS [4]. The maximum duration that a target can be covered is the sum of the batteries of all its nearby sensors that can cover it and is known as the life of a target. The main intuition behind DEEPS is to try to minimize the energy consumption rate around those targets with smaller lives. A sensor thus has several targets with varying lives. A target is defined as a *sink* if it is the shortest-life target for at least one sensor covering that target. Otherwise, it is a *hill*. To guard against leaving a target uncovered during a shuffle, each target is assigned an in-charge sensor. For each sink, its in-charge sensor is the one with the largest battery for which this is the shortest-life target. For a hill target, its in-charge is that neighboring sensor whose shortest-life target has the longest life. An in-charge sensor does not switch off unless its targets are covered by someone. Apart from this, the rules are identical as those in LBP protocol. DEEPS relies

on two-hop information to make these decisions.

In [12], we describe how [2] and [4] can be modeled using our target coverage solution (Section 4.1).

## 7 Conclusion

In this paper, we present a general algorithmic framework for solving certain graph and network problems. The framework applies to problems that exhibit the property of local compatible solutions yielding globally feasible solutions when combined. We utilize the framework to develop heuristics for solving the different coverage problems for sensor networks. Experimental evaluation of these heuristics shows performance gains when compared to existing approaches in [2] [4].

The value of our framework lies in the definition of a *Dependency Graph* to capture the interactions between local solutions. This enables us to not only model these interactions but also factor them into the process of picking good solutions. While previous algorithms are based on greedily picking solutions, our heuristics can use this graph to gain an insight into the problem structure. This is combined with a negotiating phase, where each node communicates with its neighbors to determine which solution to use.

Overall, our framework presents a new way of looking at these problems. An initial version has been presented here and several variations of the Dependency Graph and the weight functions are currently being explored.

## Acknowledgment

The authors would like to thank Dr. Yingshu Li for fruitful discussions on different coverage problems

## References

- [1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Commun. Mag.*, 2002.
- [2] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky. Efficient energy management in sensor networks. In *Ad Hoc and Sensor Networks, Wireless Networks and Mobile Computing*, 2, 2005.
- [3] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky. Power efficient monitoring management in sensor networks. *WCNC'04: Wireless Communications and Networking Conference, IEEE*, 4, 21-25 March 2004.
- [4] D. Brinza and A. Zelikovsky. Deeps: Deterministic energy-efficient protocol for sensor networks. *SAWN '06: Proceedings of the International Workshop on Self-Assembling Wireless Networks*, 2006.
- [5] M. Cardei and D.-Z. Du. Improving wireless sensor network lifetime through power aware organization. *Wireless Networks*, 11:333–340(8), May 2005.
- [6] M. Cardei and J. Wu. Energy-efficient coverage problems in wireless ad hoc sensor networks. *Computer Communications*, 29(4):413–420, 2006.
- [7] S. Das, N. Deo, and S. K. Prasad. Two minimum spanning forest algorithms for fixed-size hypercube computers. *Parallel Computing*, 15:179–187, 1990.
- [8] A. Dhawan, C. T. Vu, A. Zelikovsky, Y. Li, and S. K. Prasad. Maximum lifetime of sensor networks with adjustable sensing range. *SAWN '06: Proceedings of the International Workshop on Self-Assembling Wireless Networks*, 2006.
- [9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
- [10] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *FOCS*, pages 300–309, 1998.
- [11] S. Kumar, T. H. Lai, and J. Balogh. On k-coverage in a mostly sleeping sensor network. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 144–158, New York, NY, USA, 2004. ACM.
- [12] S. K. Prasad and A. Dhawan. Distributed algorithms for lifetime of wireless sensor networks based on dependencies among cover sets. In *HiPC '07: Proceedings of the 14th International Conference on High Performance Computing*. Springer, 2007.
- [13] M. Slijepcevic, S.; Potkonjak. Power efficient organization of wireless sensor networks. *Communications, 2001. ICC 2001. IEEE International Conference on*, 2:472–476 vol.2, 2001.
- [14] D. Tian and N. D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, 2002.
- [15] G. Xing, X. Wang, Y. Zhang, C. Lu, R. Pless, and C. Gill. Integrated coverage and connectivity configuration for energy conservation in sensor networks. *ACM Trans. Sen. Netw.*, 1(1):36–72, 2005.
- [16] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, 2001.
- [17] F. Ye, G. Zhong, S. Lu, and L. Zhang. Peas: A robust energy conserving protocol for long-lived sensor networks. *ICNP '02: IEEE International Conference on Network Protocols*, 00:200, 2002.
- [18] H. Zongheng Zhou; Das, S.; Gupta. Connected k-coverage problem in sensor networks. *Computer Communications and Networks, 2004. ICCCN 2004. Proceedings. 13th International Conference on*, pages 373–378, 11-13 Oct. 2004.