8-5-2009

Distributed Algorithms for Maximizing the Lifetime of Wireless Sensor Networks

Akshaye Dhawan
Georgia State University

DISTRIBUTED ALGORITHMS FOR MAXIMIZING THE LIFETIME OF WIRELESS SENSOR NETWORKS

by

AKSHAYE DHAWAN

Under the direction of Sushil K. Prasad

ABSTRACT

Wireless sensor networks (WSNs) are emerging as a key enabling technology for applications domains such as military, homeland security, and environment. However, a major constraint of these sensors is their limited battery. In this dissertation we examine the problem of maximizing the duration of time for which the network meets its coverage objective.

Since these networks are very dense, only a subset of sensors need to be in "sense" or "on" mode at any given time to meet the coverage objective, while others can go into a power conserving "sleep" mode. This active set of sensors is known as a cover. The lifetime of the network can be extended by shuffling the cover set over time.

In this dissertation, we introduce the concept of a local lifetime dependency graph consisting of the cover sets as nodes with any two nodes connected if the corresponding covers intersect, to capture the interdependencies among the covers. We present heuristics based on some simple properties of this graph and show how they improve over existing algorithms. We also present heuristics based on other properties of this graph, new models for dealing with the solution space and a generalization of our approach to other graph problems.

INDEX WORDS: Wireless sensor networks, Target coverage, Lifetime, Scheduling

DISTRIBUTED ALGORITHMS FOR MAXIMIZING THE LIFETIME OF WIRELESS SENSOR NETWORKS

by

AKSHAYE DHAWAN

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2009

DISTRIBUTED ALGORITHMS FOR MAXIMIZING THE LIFETIME OF WIRELESS SENSOR NETWORKS

by

AKSHAYE DHAWAN

Committee Chair: Sushil K. Prasad

Committee: Rajshekhar Sunderraman
Yingshu Li
Shamkant B. Navathe

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
December 2009

ACKNOWLEDGMENTS

I would like to thank numerous people who have been there for me in various capacities during this journey.

I would like to begin by thanking my grandparents for always teaching by example.

To my parents, a thank you seems to be insufficient to express my gratefullness for all that you have done and for all the love and support you have always given me.

I would also like to thank my sister for always providing a listening ear and being a friend before being a sibling.

No amount of thanks is enough for Mark, Beth and Mara, for giving me a family and a home away from home.

Over the course of this degree, my advisor Dr. Sushil Prasad has helped me in innumerable ways. He always gave me the freedom to pursue my interests and sound advice and direction when it was needed. For helping me mature as a professional, I am forever thankful to him.

I would also like to thank the members of my committee. Dr. Sunderamman has always been an invaluable mentor and I have learned a lot from him. I will always have memories of our numerous conversations on a variety of topics. Dr. Li has shared her expertise in this domain with me and Dr. Navathe has been an excellent mentor.

My friends have always been there to make sure I have a life outside work. Special thanks goes to my roommate Avik - I will fondly remember our conversations and many shared interests! I would also like to thank my friend Ashwin for always inspiring me to do better work and for our weekly nights at trivia. Navin has been a part of this entire journey and I would like to thank him for all the conversation over endless cups of coffee and numerous lunches. Finally, I would like to thank Mounica for being there at the end of this journey and the beginning of the next one.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 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. The radio on board these sensor nodes has limited range and allows the node to transmit over short distances. In most deployment scenarios, it is not possible for each node to communicate directly to the sink and hence, the model of communication is to transmit over short distances to other peers in the direction of the sink nodes.

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. The nodes also have limited memory and processing capabilities. Hence, harnessing the potential of these networks involves tackling a myriad of different issues from algorithms for network operation, programming models, architecture and hardware to more traditional networking issues. For a more detailed survey on the various computational research aspects of Wireless Sensor Networks, see the survey papers [4, 18, 36, 59, 57], or the more recent books [35, 44] and a special issue of the CACM [19].

This dissertation focuses on the algorithmic aspects of Wireless Sensor Networks. Specifically, we look at the problem of covering a set of targets or an area for the longest duration possible. The next section focuses on a more detailed discussion of the problem and provides a formal statement for it. It is worth mentioning that there is an abundance of algorithmic research related to WSNs. A lot of this focuses on traditional distributed computing issues like localization, fault tolerance, robustness. This naturally raises the interesting question of

how different are WSNs as a computational model than more traditional distributed computing environments or even ad-hoc networks? This question has been explored briefly in [64].

One of the underlying themes of this dissertation comes back to this question. In our experiences on working with designing distributed algorithms for scheduling sensors to maximize coverage lifetime, we found several distributed algorithms in the literature that applied common distributed algorithm design principles to this problem. However, our approach achieved significant improvements over competing algorithms because it was based on a model of the problem structure. The lesson for us has been that while standard approaches are a good starting point to various problems in WSNs, these problems usually have a rich underlying structure, the exploration of which yields better algorithms. Hence, there is a clear and pressing need for finding good models for these networks that allow the development of algorithms for various problems in an efficient way.

1.1 Coverage Problems

Many intended applications of Wireless Sensor Networks involve having the network monitor a region or a set of targets. 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. Deployment is usually done by flying an aircraft over the region and air dropping the sensors. Since the cost of deployment far exceeds the cost of individual sensors, many more sensors are dropped than needed to minimally cover the region. The leads to a very dense network and 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. This can also lead to a very lossy network with several collisions happening in the medium access control (MAC) layer due to the density of nodes. 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 [12, 31].

Existing centralized and distributed heuristics for arriving at a lifetime extending schedule are discussed in Chapter 2. Centralized solutions like those in [61, 12] 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 latter is particularly relevant in the context of sensor networks since sensor nodes are deployed in hostile environments and are prone to frequent failures.

Existing distributed solutions in [62, 6, 7] 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. Distributed algorithms are organized into rounds so that the set of active sensors is periodically reshuffled at the beginning of each round. The problem with these algorithms is that they use simple greedy criteria to make their decision on which sensors become active at each round and thus, do not efficiently take into account the problem structure.

We look at coverage problems in much more detail in Chapter 2.

1.2 Problem Statement

The lifetime problem can be stated as follows. Given a monitored region R, a set of sensors S and a set of targets T, find a monitoring schedule for these sensors such that

- the total time of the schedule is maximized,
- all targets are constantly monitored, and
- no sensor is in the schedule for longer than its initially battery.

A related problem is that of monitoring an area of interest. In general, the area and target coverage problems have been shown to be equivalent. [61, 5, 15] provide ways to map an area to a set of points (targets) . In the work presented in the remainder of this dissertation, we focus on the target coverage problem with the implicit understanding that the algorithms and techniques presented can be translated to the area coverage problem by mapping the area to a set of points (virtial targets) with an appropriate granularity.

There are also several other variations of this basic problem. For example the $p\%$ coverage problem [47] requires only a certain percentage of all targets to be covered. The fault tolerant k-coverage version of this problem requires each target to be covered by at least k sensors [72, 40]. Also, the basic problem has been modified to include sensors that have adjustable sensing ranges, non uniform sensing shapes and othe heterogeneous sensor network models.

1.3 Our Contributions

In this dissertation, we present our work on a framework for the design of distributed algorithms that extend the lifetime of the network. This work has appeared in [55, 23, 24, 22, 25]. We have also studied centralized solutions for a variation of this problem applied to sensors with adjustable ranges in [21].

In [55] 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 [6][7] can be modeled by variations in the prioritizing scheme. We carried out simulations to show improvements of 10-20% over LBP[6] and similar performance to DEEPS[7] which is a 2-hop algorithm. A 2-hop version of our algorithm outperformed DEEPS [7]. This basic algorithm and its results are presented in Chapter 3. This work was fundamental in that the LD Graph allowed for a new representation of this problem in a manner that had not been looked at in the literature. This model also formed the basis of the work that followed in [23, 24, 22, 25].

Chapter 4 is based on our work in [24]. The key motivation behind our approach in this paper had been to start with the question of what an optimal schedule (OPT) would do with the lifetime dependency graph. We were able to prove certain basic properties of the OPT schedule that relate to the LD graph. Based on these properties, we have designed algorithms which choose the covers that exhibit these OPT schedule like properties. We present three new heuristics - *Sparse-OPT* based on the sparseness of connectivity among covers in OPT, *Bottleneck-Target* based on selecting covers that optimize the use of sensors covering local bottleneck targets, and *Even-Target-Rate* based on trying to achieve an even burning rate for all targets. These heuristics are, therefore, designed at a higher level and operate on top of our above mentioned degree-based heuristics to prioritize the local covers. Our experiments show an improvement in lifetime of 10-15% over our previous work in [55] and 25-30% over competing work in the literature [6, 7]. We also implemented two-hop versions of these heuristics and show that they give around 35% improvement over the two-hop algorithm of [7]. This work was innovative in that it first exhibited theoretical properties of the LD Graph and then designed heuristics that try and meet these properties.

In Chapter 5, we address the unresolved question of how to deal with the exponential space of all possible cover sets. We present a reduction of this exponential space to a linear

one based on grouping cover sets into *equivalence classes*. The partition into classes is defined by the equivalence relation on the set of all sensor covers: Given a set C and an equivalence relation $\Re$, the equivalence class of an element $C_i \in C$ is the subset of all elements in C which are equivalent to C_i. The notation used to represent the equivalence class of C_i is $[C_i]$. In the context of the problem being studied, C is the set of all sensor covers and for any single cover C_i, $[C_i]$ represents all other covers which are *equivalent* to C_i as given by the definition of some equivalence relation $\Re$. Our approach stems from the understanding that from the possible exponential number of sensor covers, several covers are very similar, being only minor variations of each other. We present the definition of the relation $\Re$, based on a grouping that considers cover sets equivalent if their lifetime is bounded by the same sensor. We then show the use of this relation to collapse the exponential LD Graph into an *Equivalence Class* (EC) Graph with linear number of nodes. This theoretical insight allows us to design a sampling scheme that selects a subset of all local covers based on their equivalence class properties and presents this as an input to our simple LD graph degree-based heuristic. Simulation results show that class based sampling cuts the running time of these heuristics by nearly half, while only resulting in a less than 10% loss in quality.

In Chapter 6, based on [23, 25] we extend our work on target coverage and present a generalized framework that can be applied to graph and network problems that exhibit certain properties. We also showed the use of this framework in solving the area and k-coverage problems. This work shows the general applicability of modeling dependencies into a graph and then designing distributed algorithms based on this. We also examine the application of this framework to the Independent Set problem and show its representation in our framework.

Chapter 7, presents a new model for sensing wherein sensors can smoothly adjust their sensing ranges in a given range. This model accurately reflects the capability of existing sensor hardware and allows a sensor to reduce its range to cover only those targets that are necessary, thereby saving energy. We present centralized and distributed algorithms for tar-

get coverage in this model that show improvements over their fixed range counterparts. The centralized algorithms in this chapter were published in [21].We also present two distributed algorithms ADEEPS and ALBP which are extensions of the LBP and DEEPS algorithms to the adjustable range model introduced by us.

Finally, we conclude and discuss future work in Chapter 8.

CHAPTER 2.

RELATED WORK

In this section, we briefly survey existing approaches to maximizing the lifetime of sensor networks, while meeting certain coverage objectives. [9] gives a more detailed survey on the various coverage problems and the scheduling mechanisms they use. [58] also surveys the coverage problem along with other algorithmic problems relevant to sensor networks. We end this section by focusing on two algorithms, LBP [6] and DEEPS [7], since we use them for comparisons against our algorithms.

A key application of wireless sensor networks is the collection of data for reporting. There are two types of data reporting scenarios: event-driven and on-demand [15]. Event-driven reporting occurs when one or more sensor nodes detect an event and report it to the sink. In on-demand reporting, the sink initiates a query and the nodes respond with data to this query.

Coverage problems essentially state how well a network observes its physical space. As pointed out in [50], coverage is a measure of the quality of service (QoS) for the WSN. The goal is to have each point of interest monitored by at least one sensor at all times. In some applications, it may be a requirement to have more than one sensor monitor a target for achieving fault tolerance. Typically, nodes are randomly deployed in the region of interest because sensor placement is infeasible. This means that more sensors are deployed than needed to compensate for the lack of exact positioning and to improve fault tolerance in harsh environments. The question of placing an optimal number of sensors in a deterministic deployment has been looked at in [38, 26, 54]. However, in this dissertation we focus on networks with very dense deployment of sensors so that there is significant overlap in the targets each sensor monitors. This overlap will be exploited to schedule sensors into a low power sleep state so as to improve the lifetime of these networks. Note that this definition

of the network lifetime is different from some other definitions which measure this in terms of number of operations the network can perform [33].

The reason for wanting to schedule sensors into sense-sleep cycles that we talked about in Chapter 1, stems from the fact that sensor nodes have four states - *transmit, receive, idle and sleep*. As shown in [56] for the WINS Rockwell sensor, the transmit, receive and idle states all consume much more power than the sleep state - hence, it is more desirable for a sensor to enter a sleep state to conserve its energy. The goal behind sensor scheduling algorithms is to select the activity state of each sensor so as to allow the network as a whole to monitor its points of interest for as long as possible. For a more detailed look at power consumption models for ad-hoc and sensor networks we refer the reader to [29, 28, 37]. We now look at coverage problems in more detail.

2.1 Coverage Problems in Other Areas

Coverage problems have long existed in other fields of work. In this section, we briefly survey major results on coverage in a context outside that of sensor networks.

The Art Gallery Problem [53], looks at the question of deploying observers in an art-gallery such that every point in the room is seen by at least one observer. There is a linear time solution provided for the 2D case but the 3D version of the problem is shown to be NP=hard and an approximation algorithm is given in [48].

[34] looks at problem of covering an ocean for satellited based monitoring of phytoplankton abundance. They show that having three satellites observe the region and merging their data results in 50 % improvements in coverage but adding more satellites produce minimal improvements over this.

Coverage has also been looked at in robot-systems by [30]. The author introduces the notions of blanket, barrier and sweep coverage. Blanket coverage attempts to maximize the total area covered. Barrier coverage arranges nodes in a manner that minimizes the probability of undetected intrusion. Sweep coverage is a mobile barrier detection problem.

Table 2.1. Centralized Algorithms

Name	Area/Target	Disjoint	Main Idea
Abrams, Goel [3]	Area	Yes	Greedy: Max uncovered area
Meguerdichian [52]	Area	No	Integer Linear Program
Cardei [12]	Target	Yes	Mixed Integer Programming
Shah [5]	Area	Yes	LP, Garg Könemann
Cardei [8]	Target	No	Integer Linear Program

In particular, the definition of barrier coverage introduced here has been used in the context of barrier coverage for wireless sensor networks in [43, 27, 50, 51, 68].

2.2 Coverage Problems in Wireless Sensor Networks

The maximum lifetime coverage problem has been shown to be NP-complete in [3, 12]. Initial approaches to the problem in [61, 12, 3] considered the problem of finding the maximum number of *disjoint* cover sets of sensors. This allowed each cover to be used independently of others. However, [8, 5] and others showed that using non-disjoint covers allows the lifetime to be extended further and this approach has been adopted since.

Broadly speaking, the existing work in this category can be classified into two parts - Centralized Algorithms and Distributed Algorithms. For centralized approaches, the assumption is that a single node (usually the base station) has access to the entire network information and can use this to compute a schedule that is then uploaded to individual nodes. Distributed Algorithms work on the premise that a sensor can exchange information with its neighbors within a fixed number of hops and use this to make scheduling decisions. We now look at the individual algorithms in both these areas.

A common approach taken with centralized algorithms is that of formulating the problem as an optimization problem and using linear programming (LP) to solve it [52, 12, 5, 21]. In [61], the authors develop a most-constrained least-constraining heuristic and demonstrated its effectiveness on variety of simulated scenarios. In this heuristic, the main idea is to minimize the coverage of sparsely covered areas within one cover. Such areas are identified

using the notion of the critical element, defined as the element which is a member of the smallest number of sets. Their heuristic favors sets that cover a high number of uncovered elements, that cover more sparsely covered elements, that do not cover the area redundantly and that redundantly cover the elements that do not belong to sparsely covered areas. [52] is a followup work by the same authors in which they formulate the area coverage problem using a Integer LP and relax it to obtain a solution. They also presented several ILP based formulations and strategies to reduce overall energy consumption while maintaining guaranteed sensor coverage levels. Additionally, their work demonstrated the practicality and effectiveness of these formulations on a variety of examples and provided comparisons with several alternative strategies. They also show that the ILP based technique can scale to large and dense networks with hundreds of sensor nodes.

In order to solve the target coverage problem, [12] considers the disjoint cover set approach. Modeling their solution as a Mixed Integer Program shows an improvement over [61]. The authors define the disjoint set covers (DSC) problem and prove its NP-completeness. They also prove that any polynomial-time approximation algorithm for DSC problem has a lower bound of 2. They first transform DSC into a maximum-flow problem (MFP), which is then formulated as a mixed integer programming. Based on the solution of the MIP, the authors design a heuristic to compute the number of covers. They evaluate the performance by simulation, against the most constrainedminimally constraining heuristic proposed in [61] and found that their heuristics has a larger number of covers (larger lifetime) at the cost of a greater running time.

[5] formulates a packing LP for the coverage problem. Using the $(1+\epsilon)$ Garg-Könemann approximation algorithm [32], they provide a $(1+\epsilon)(1+2lnn)$ approximation of the problem. They also present an efficient data structureto represent the monitored area with at most n^2 points guaranteeing the full coverage which is superior to the previously used approach based on grid points in [61]. They also present distributed algorithms that tradeoff between

monitoring and power consumption but these are improved upon by the authors in LBP and DEEPS.

A similar problem is solved by us for sensors with *adjustable* ranges in [21]. We present a linear programming based formulation that also uses the $(1+\epsilon)$ Garg-Könemann approximation algorithm [32]. The main difference is the introduction of an adjustable range model that allows sensors to vary their sensing and communication ranges smoothly. This was the first model that allows sensors to vary their range to any value upto a maximum. The model is an accurate representation of physical sensors and allows significant power savings over the discreetly varying adjustable model.

A different algorithm to work with disjoint sets is given in [13]. Disjoint cover sets are constructed using a graph coloring based algorithm that has area coverage lapses of about 5%. The goal of their heuristic is to achieve energy savings by organizing the network into a maximum number of disjoint dominating sets that are activated successively. The heuristic to compute the disjoint dominating sets is based on graph coloring. Simulation studies are carried out for networks of large sizes.

[3] also gives a centralized greedy algorithm that picks sensors based the largest uncovered area. They have designed three approximation algorithms for a variation of the SET K-COVER problem, where the objective is to partition the sensors into covers such that the number of covers that include an area, summed over all areas, is maximized. The first algorithm is randomized and partitions the sensors within a fraction of the optimum. The other two algorithms are a distributed greedy algorithm and a centralized greedy algorithm. The approximation ratios are presented for each of these algorithms.

[8] also deal with the target coverage problems. Like similar algorithms, they also extend the sensor network life time by organizing the sensors into a maximal number of set covers that are activated successively. But they allow non-disjoint set covers. The authors model the solution as the maximum set covers problem and design two heuristics that efficiently compute the sets, using linear programming and a greedy approach. The greedy algorithm

Table 2.2. Distributed Algorithms

Name	Area/Target	Disjoint	Main Idea
Sliepcivic [61]	Area	Yes	Greedy: Max uncovered fields
Tian [62]	Area	No	Geometric calculation of sponsored area
PEAS [67]	Area	No	Probing based determination of sponsored area
CCP [66]	Area	No	Random timers to evaluate coverage requirements
OGDC [69]	Area	No	Random back off node volunteering
Lu [45]	Area	No	Highest overall denomination sensor picks
Abrams [3]	Area	Yes	Randomized, Greedy picks max uncovered area
Cardei et al. [8]	Target	No	Sensor with highest contribution to bottleneck
LBP [6]	Target	No	Targets are covered by higher energy nodes
DEEPS [7]	Target	No	Minimize energy consumption for bottleneck target

selects a critical target at each step. This is the least covered target. For the greedy selection step, the sensor with the greatest contribution to the critical target is selected.

The distributed algorithms in the literature can be further classified into greedy, randomized and other techniques. The greedy algorithms [61, 45, 3, 8, 6, 7] all share the common property of picking the set of active sensors greedily based on some criteria. [61] considers the area coverage problem and introduces the notion of a *field* as the set of points that are covered by the same set of sensors. The basic approach behind the picking of a sensor is to first pick the one that covers that largest number of previously uncovered fields and to then avoid including more than one sensor that covers a sparsely covered field. [3] builds on this work and presents three algorithms that solve variations of the set k-cover problem. The greedy heuristic they propose works by selecting the sensor that covers the largest uncovered area. [45] defines the sensing denomination (SD) of a sensor as its contribution, i.e., the area left uncovered when the sensor is removed. The authors assume that each sensor can probabilistically detect a nearby event, and build a probabilistic model of network coverage by considering the data correlation among neighboring sensors. The more the contribution of a sensor to the network coverage, the higher the sensors SD is. Based on the location information of neighboring sensors, each sensor can calculate its SD value in a distributed manner. Sensors with higher sensing denomination have a higher probability of remaining active.

[5] gives a distributed algorithm based on using the faces of the graph. If all the faces that a sensor covers are covered by other sensors with higher battery that are in an active or deciding state, then a sensor can switch off (sleep). Their work has been extended to target coverage in the load balancing protocol (LBP).

Some distributed algorithms use randomized techniques. Both OGDC [69] and CCP [66] deal with the problem of integrating coverage and connectivity. They show that if the communication range is at least twice the sensing range, a covered network is also connected. [69] uses a random back off for each node to make nodes volunteer to be the start node.

OGDC addresses the issues of maintaining sensing coverage and connectivity by keeping a minimum number of sensor nodes in the active mode in wireless sensor networks. They investigate the relationship between coverage and connectivity. They also derive, under the ideal case in which node density is sufficiently high, a set of optimality conditions under which a subset of working sensor nodes can be chosen for complete coverage. OGDC algorithm is fully localized and can maintain coverage as well as connectivity, regardless of the relationship between the radio range and the sensing range. OGDC achieves similar coverage with an upto 50% improvement in the lifetime of the network. A drawback of OGDC is that it requires that each node knows its own location.

In [49] the authors combine computational geometry with graph theoretic techniques. The use Voronoi diagrams with graph search to design a polynomial time worst and average case algorithm for coverage calculation in homogeneous isotropic sensors. The also analyze and experiment with using these techniques as heuristics to improve coverage.

[66] sets a random timer for each node following which a node evaluates its current state based on the coverage by its neighbors. The authors present a Coverage Configuration Protocol (CCP) that can provide different degrees of coverage requested by applications. This flexibility allows the network to self-configure for a wide range of applications. They also integrate CCP to SPAN [17, 16] to provide both coverage and connectivity guarantees. [3] also present a randomized algorithm that assigns a sensor to a cover chosen uniformly at random.

A different approach has been taken in PEAS [67, 62]. PEAS is a distributed algorithm with a probing based off-duty rule is given in [67]. PEAS is localized and has a high resilience to node failure and topology changes. Here, every sensor broadcasts a probe PRB packet with a probing range γ. Any working node that hears this probe packet responds. If a sensor receives at least one reply, it can go to sleep. The range can be chosen based on several criteria. Note that this algorithm does not preserve coverage over the original area. The results for PEAS showed an increase in the network lifetime in linear proportion to the

number of deployed nodes. In [62] the authors give a distributed and localized algorithm. Every sensor has an off-duty eligibility rule. They give an algorithm for a node to compute its sponsored area. To prevent the occurrence of blind-points by having two sensors switch off at the same time, a random back off is used. They show improved performance over PEAS.

To our knowledge, [66] was the first work to consider the k-coverage problem. [40] 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. [72] solves the problem of picking minimum size connected k-covers. The authors state this as an optimization problem and design a centralized approximation algorithm that delivers a near-optimal solution. They also present a communication-efficient localized distributed algorithm for this problem.

Now, we look at the two protocols that we compare our heuristics against. The load balancing protocol (LBP) [6] 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 [5]. 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 [7]. 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.

CHAPTER 3.

THE LIFETIME DEPENDENCY GRAPH MODEL

In this chapter, we introduce the Lifetime Dependency (LD) Graph as a model for the maximum lifetime coverage problem defined in Section 1.2. This model is a key contribution of this dissertation since the heuristics and algorithms that follow in subsequent chapters rely heavily on the LD Graph.

Recall from Section 1.2 that given a sensor network and a set of static targets, the maximum lifetime sensor scheduling problem is to select a subset of sensors that covers all targets and then periodically shuffle the members of this subset so as to maximize the total time for which the network can cover all targets.

Since these sensors are powered by batteries, energy is a key constraint for these networks. Once the battery has been exhausted, the sensor is considered to be dead. The lifetime of the network is defined as the amount of time that the network can satisfy its coverage objective, i.e., the amount of time that the network can cover its *area* or *targets* of interest. Having all the sensors remain "on" would ensure coverage but this would also significantly reduce the lifetime of the network as the nodes would discharge quickly. A standard approach taken to maximizing the lifetime is to make use of the overlap in the sensing regions of individual sensors caused by the high densit y of deployment. Hence, only a subset of all sensors need to be in the "on" or "sense" state, while the other sensors can enter a low power "sleep" or "off" state. The members of this active set, also known as a *cover* set, are then periodically updated so as to keep the network alive for longer duration. In using such a scheduling scheme, there are two problems that need to be addressed. First, we need to determine how long to use a given cover set and then we need to decide which set to use next. This problem has been shown to be NP-complete [3, 12].

A key problem here is that since a sensor can be a part of multiple covers, these covers have an impact on each other, as using one cover set reduces the lifetime of another set that

has sensors common with it. By making greedy choices, the impact of this dependency is not being considered, since none of the heuristics in the literature study this reduction in the lifetime of other cover sets caused by using a sensor that is a member of several such sets. The earlier disjoint formulations mentioned in the previous chapter, entirely avoided this problem by preventing it.

We capture this dependency between covers by introducing the concept of a local Lifetime Dependency (LD) Graph. This consists of the cover sets as nodes with any two nodes connected if the corresponding covers intersect. The graph is an example of an intersection graph since it represents the sensors common to different cover sets. By looking at the graph locally (fixed 1-2 hop neighbors), we are able to construct all the local covers for the local targets and then model their dependencies. Based on these dependencies, a sensor can then prioritize its covers and negotiate these with its neighbors. We also present some simple heuristics based on the graph. The material presented in this chapter was published in [55].

3.1 Symbols and Definitions

Let us begin with a few basic conventions and definitions that will be used in the rest of this dissertation. Individual chapters will introduce additional notation as and when necessary. The notation presented here applies to the basic LD grapg model and will be utilized for all the chapters that follow.

We will use s_1, s_2, etc., to represent sensors, t_1, t_2, etc., to represent targets, and C, C', etc., to denote covers.

Let us assume we have n sensors and m targets, both stationary.

Consider the sensor network in Figure 3.1 with $n = 8, s = \{s_1, s_2, ..., s_8\}$ and $m = 3$ targets, t_1, t_2, and t_3.

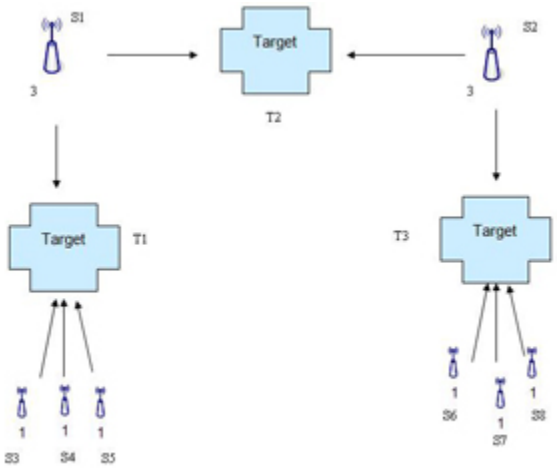

Figure 3.1. A sensor network

We will employ the following definitions, illustrated using this network.

- $b(s)$: strength of the battery of sensor s; for example, $b(s_1) = 3$ while $b(s_3)= 1$.
- $T(s)$: set of targets that sensor s can sense; e.g., $T(s_1) = \{t_1, t_2\}$;
- $N(s, k)$: closed set of neighbors of sensor s at no more than k hops (i.e, those neighbors that s can communicate with using $\leq k$ hops) - this contains s itself; thus, $N(s_1, 1) = \{s_1, s_2, s_3, s_4, s_5\}$.
- Cover: C is a cover for targets in set T if

  (i) for each target $t \in T$ there is at least one sensor in C which can sense t and

  (ii) C is minimal. For example, the possible (minimal) covers for the two targets of s_1 are $\{s_1\}, \{s_2, s_3\}, \{s_2, s_4\}$ and $\{s_2, s_5\}$. There are other non-minimal covers as well such as s_1, s_2 which need to be avoided. Likewise, the possible covers for the only target of sensor s_3 are $\{s_1\}, \{s_3\}, \{s_4\}$ and $\{s_5\}$.
- $lt(C) = min_{s \in C} b(s)$, the maximum lifetime of a cover. The bottleneck sensor of the cover $\{s_2, s_3\}$ is s_3 with the weakest battery of 1. Therefore, $lt(\{s_2, s_3\}) = 1$.

An optimal lifetime schedule of length 6 for this network is $(\{s_1, s_6\}, 1)$, $(\{s_1, s_7\}, 1)$, $(\{s_1, s_8\}, 1)$, $(\{s_2, s_3\}, 1)$, $(\{s_2, s_4\}, 1)$, $(\{s_2, s_5\}, 1))$ where each tuple is a cover for the entire network followed by its duration.

3.2 Lifetime dependency (LD) Graph

Let the local lifetime dependency graph be $G = (V, E)$ where nodes in V denote the local covers and edges in E exist between those pairs of nodes whose corresponding covers share one or more common sensors. For simplicity of reference, we will not distinguish between a cover C and the node representing it, and an edge e between two intersecting covers C and C' and the intersection set $C \bigcap C'$. Each sensor constructs its local LD graph considering its one- or two-hop neighbors and the corresponding targets. Figure 3.2 shows the local lifetime dependency graph of sensor s_1 in the example network of Figure 3.1, considering its one-hop neighbors $N(s_1, 1)$ and its targets $T(s_1)$.

In the LD graph, we will use the following two definitions:

- $w(e) = min_{s \in e} b(s)$, the weight of an edge e (if e does not exist, i.e., if e is empty, then $w(e)$ is zero).
- $d(C) = \sum_{e \in E \text{ and incident to } C} w(e)$, the degree of a cover C.

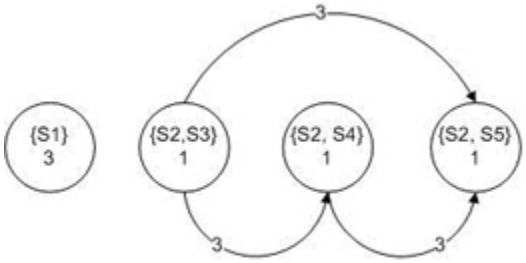

Figure 3.2. The local lifetime dependency graph of sensor s_1

In Figure 3.2, the two local covers $\{s_2, s_3\}$ and $\{s_2, s_4\}$ for the targets of sensor s_1 have s_2 in common, therefore the edge between the two covers is $\{s_2\}$ and $w(\{s_2\}) = 3$. Therefore,s_2's

battery of 3 is an upper bound on the lifetime of the two covers collectively. It just so happens that the individual lifetimes of these covers are each 1 due to their bottleneck sensors and, therefore, a tighter upper bound on their total life is 2. In general, given two covers C and C', a tight upper bound on the life of two covers is $min(lt(C) + lt(C'), w(C \bigcap C'))$.

3.3 The Basic Algorithm

For the purpose of this explanation, without loss of generality, let us assume that the covers are constructed over one-hop neighbors. The algorithm consists of two phases. During the initial setup phase, each sensor calculates and prioritizes the covers. Then, for each reshuffle round of predetermined duration, each sensor decides its on/off status at the beginning, and then those chosen remain on for the rest of the duration.

Initial setup: Each sensor s communicates with each of its neighbor $s' \in N(s, 1)$ exchanging mutual 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.

The "priority function" of a cover is based on its degree (lower the better). Ties among covers with same degree are broken first by preferring (i) those with longer lifetimes, then (ii) those which have fewer remaining sensors to be turned on, and finally (iii) by choosing the cover containing the smaller sensor id. A cover which has a sensor turned off becomes infeasible and falls out of contention. Also, a cover whose lifetime falls below the duration of a round is taken out of contention, unless it is the only cover remaining.

Reshuffle rounds: The automaton in Figure 3.3 captures the algorithm for this phase. A sensor s starts with its highest priority cover C as its most desirable configuration for its neighborhood. If successful, the end result would be switching on all the sensors in C, while

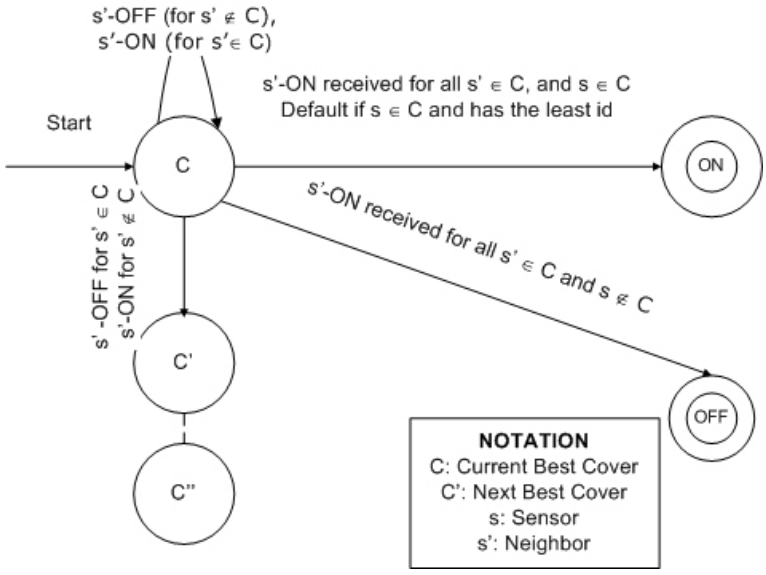

Figure 3.3. The state transitions to decide the On-Off Status

others can sleep. Else, it transitions to the next best priority cover C', C'', etc., until a cover gets satisfied. The transitions are as follows.

- Continue with the best cover C: Sensor s continues with its current best cover C if its neighbor $s' \notin C$ goes off (thus not impacting the chances of ultimately satisfying 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, 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 itself is not in cover 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.

The transitions from C' are analogous to that from C, with the possibility of even going back to C.

Correctness: We sketch a proof here that this algorithm ensures that, in each reshuffle round, all the targets are covered and the algorithm itself terminates enabling each sensor to decide and reach on/off status.

For contradiction, let us assume that in a given round a target t remains uncovered. This implies that either this target has no neighboring sensor within sensing range and thus network itself is dead, or else all the neighboring sensors which could have covered t have turned off. In the latter case, each of the sensor s whose $T(s)$ contains t has made the transition from its current best cover C to off status. However, s only does that if C covers all its targets in $T(s)$ and $s \notin C$. The last such sensor s to have turned off ensures that C is satisfied, which implies that all targets in $T(s)$ including t are covered, a contradiction. Next, for contradiction, let us assume that the algorithm does not terminate. This implies that there exists at least one sensor s which is unable to decide, i.e., make a transition to either on or off status. There are three possibilities: (i) all the covers of s have become infeasible, or

(ii) s is continually transitioning to the next best cover and none of them are getting satisfied, or

(iii) s is stuck at a cover C.

For case (i), for each cover C, at least one of its sensor $s' \in C$ has turned off. But the set of targets considered by sensor s is no larger than $T' = \bigcup_{s' \in N(s,1)} T(s')$. Since s itself can cover $T(s)$, there exist a target $t \in T' - T(s)$, from $T(s')$, that none of the cover sets at s are able to cover. This implies that s' is off, else $\{s, s'\}$ would have formed part of a cover at s covering t (given that s constructs all possible covers). This leads to the contradiction, as before turning off, s' ensures that $t \in T(s')$ is covered.

For case (ii), each transition implies that a neighbor sensor has decided its on/off status, thereby making some of the covers at s infeasible and increasingly satisfying portions of some other covers, thus reducing the choices from the finite number of its covers. Eventually, when

the last neighbor decides, s will be able to decide as well becoming on if any target in $T(s)$ is still uncovered, else going off.

For case (iii), the possibility that all sensors are stuck at their best initial covers is conventionally broken by a sensor $s \in C$ with least id in its current best cover C pro actively becoming on, even though C may not be completely satisfied. This is similar to the start-up problem faced by others distributed algorithms such as DEEPS with similar deadlock breaking solutions. At a later stage, if s is stuck at C, it means that either all its neighbors have decided or one or more neighbors are all stuck. In the former case, there exists a cover C at s which will be satisfied with s becoming on (case i). The latter case is again resolved by the start-up deadlock breaking rule by either s or s' pro actively becoming on.

Message and time complexities: Let us assume that each sensor s constructs the covers over its one-hop neighbors to cover its targets in $T(s)$ only. Let $S = \{s_1, s_2, ..., s_n\}$ $\Delta =_{max s \in S} |N(s,1)|$, the maximum number of neighbors a sensor can communicate with. The communication complexity of the initial setup phase is $O(\Delta)$, assuming that there are constant number of neighboring targets that each sensor can sense. Also, for each reshuffle round, a sensor receives $O(\Delta)$ status messages and sends out one. Assuming Δ is a constant practically implies that message complexity is also a constant. Let maximum number of targets a sensor considers is $\tau =_{max s \in S} |T(s)|$, a constant. The maximum number of covers constructed by sensor s during its setup phase is $O(\Delta^\tau)$, as each sensor in N(s,1) can potentially cover all its targets considered. Hence the time complexity of setup phase is $O((\Delta^\tau)2)$ to construct the LD graph over all covers and calculate the priorities. For example, if $\tau = 3$, the time complexity of the setup phase would be $O(\Delta^6)$. The reshuffle rounds transition through potentially all the covers, hence their time complexity is $O(\Delta^\tau)$.

3.4 Variants of the Basic Algorithm

We briefly discussed some of the properties of the LD graph earlier. For example, an edge e connecting two covers C and C' yields an upper bound on the cumulative lifetime of

both the covers. However, if $w(e)$, which equals $b(s)$ for weakest sensor $s \in e$, is larger than the sum of the lifetimes of C and C', then the edge e no longer constrains the usage of C and C'. Therefore, even though C and C' are connected, they do not influence each other's lifetimes. This leads to our first variant algorithm.

Variant 1: Redefine the edge weight e as follows:

If $min_{s \in e} b(s) < lt(C) + lt(C')$, then $w(e) = min_{s \in e} b(s)$, else $w(e) = 0$.

Thus, when calculating the degree of a cover, this edge would not be counted when not constraining, thus elevating the cover's priority. Next, the basic framework is exploiting the degree of a cover to heuristically estimate how much it impacts other covers, and the overall intent is to minimize its impact. Therefore, we sum the edge weights emanating from a cover for its degree. However, if a cover C is connected to two covers C' and C'' such that both C' and C'' have the same bottleneck sensor s, s is depleted by burning either C' or C''. That is, in a sense, only one of C' and C'' can really be burned completely, and then the other is rendered unusable because s is completely depleted. Therefore, for all practical purposes, C' and C'' can be collectively seen as one cover. As such, the two edges connecting C to C' and C'' can be thought of as one as well. This yields our second variant algorithm.

Variant 2: Redefine the degree of a cover C in the LD graph as follows. Let a cover C be connected to a set of covers $V' = C_1, C_2, , C_q$ in graph G. If there are two covers C_i and C_j in V' sharing a bottleneck sensor s, then if $w(C, C_i) < w(C, C_j)$ then $V' = V' - C_j$ else $V' = V' - C_i$. With this reduced set of neighboring covers V', the degree of cover C is

$d(C) = \sum_{C' \in V'} w(C, C')$

In the basic algorithm, each sensors constructs cover sets using its one-hop neighbors to cover its direct targets $T(s)$. However, with the same message overheads and slightly increased time complexity, a sensor can also consider its neighbors' targets. This will enable it to explore the constraint space of its neighbors as well.

Variant 3: In this variant, each sensor s constructs LD graph over one-hop neighbors $N(s,1)$ and targets in $\bigcup_{s' \in N(s,1)} T(s')$.

Variant 4: In the basic two-hop algorithm, each sensor s constructs LD graph over two-hop neighbors $N(s,2)$ and targets in $\bigcup_{s' \in N(s,1)} T(s')$. In this variant, each sensor s constructs LD graph over two-hop neighbors $N(s,2)$ and targets in $\bigcup_{s' \in N(s,2)} T(s')$.

3.5 Representing Existing Algorithms in the LD Graph Model

We now focus on two existing approaches LBP [6] and DEEPS [7], show how they operate on an example topology and then present their equivalent representation in the lifetime dependency model.

Load balancing protocol (LBP): LBP 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.

Thus, LBP is simplistic and attempts to share the load evenly between sensors instead of balancing the energy for sensors covering a specific target. Hence, for the example shown in Figure 3.1, LBP picks the sensor s_1 to be active since it is the largest sensor covering the bottom-left target T_2. Similarly, it picks s_2 for the bottom-right target T_3. This results in a total lifetime of 3 units when compared to the optimal of 6 units for the given example. Its schedule is $(\{s_1, s_2\}, 3)$.

Formulation using LD graph framework: LBP can be simulated as a special case in the lifetime dependency graph model as follows. Given a sensor s_i, its local covers for its targets $T(s_i)$ are the singleton sets $\{s\}$, for all $s \in N(s_i, 1)$. These singleton sets are then assigned priorities in order to choose which one to use next. There are two defaults: the priority is

highest if s_i is the only one covering a target (so s_i must switch on). On the other hand, the priority is lowest if all of s_i's targets are covered, so that s_i can switch off. Otherwise, the priority is assigned based on the battery level preferring to burn those sensors with higher battery, with preference for those covers not containing s_i. Thus, a key limitation of LBP is the lack of collective negotiation captured by non-singleton cover sets in our algorithms.

DEEPS protocol: The maximum duration that a target can be covered, its life, is the sum of the batteries of all its nearby sensors that can cover it. 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.

For the example shown in Figure 3.1, DEEPS achieves a lifetime of 5, assuming a shuffle round duration of 1 since initially both the sensors s_1 and s_2 are switched on. Its schedule would be $\{\{(s_1, s_2), 1\}, \{(s_1, s_6), 1\}, \{(s_1, s_7), 1\}, \{(s_2, s_3), 1\}, \{(s_2, s_4), 1\}\}$ for a total of 5 units.

Formulation using LD graph framework: DEEPS can also be represented with our lifetime dependency graph model. The representation is just like LBP with singleton set covers from $N(s_i, 1)$. The priority function of LBP is now modified suitably to account for the concept of in-charge sensors. Specifically, the order of priority preference is if a sensor alone can cover a target, a sensor is in-charge of a target, and then higher battery level. The default least priority is for a sensor if the target it is in-charge of is now covered. Again, singleton cover sets of DEEPS, as in LBP is its key limitation.

3.6 Simulation Results

In this section we first evaluate the performance of the one-hop and two-hop versions of the basic algorithm as compared to 1-hop algorithm LBP [6] and 2-hop algorithm DEEPS [7], respectively. We also consider the performance of the different variations of the basic algorithm as outlined in Section 3.4.

For the simulation environment, a static wireless network of sensors and targets scattered randomly in $100mx100m$ area is considered. We assume that the communication range of each sensor is two times the sensing range. Different variations of the number of targets, number of sensors and energy model are considered for the simulation. The linear energy model is one where the power required to sense a target at distance d is a function of d. In the quadratic energy model the power required to sense the target at distance d is a function of d^2.

One of the key things to note is that the LD graph requires all possible covers for the local targets being considered. However, since the algorithm operates on either 1-hop or 2-hop neighbors, the number of such covers is bounded, if not small. For the purpose of implementation we create a coverage matrix wherein each row represents a sensor and each column a target, and an entry i, j is set to 1 if sensor i covers target j and 0 otherwise. Note that the number of rows of this matrix is given by the size of the 1- or 2-hop neighborhood depending on the version being considered. Iterating through every column of this matrix and adding every covering sensor to all existing covers allows us to construct all combinations of covers for the targets being considered. The covers obtained in this fashion form the nodes of the LD graph at that sensor. Associated with each cover is a lifetime given by the minimum energy sensor of that cover. This forms the node weight of the LD graph. To allow easy construction of edges for the LD graph we implement the covers as sets and their intersection represents the edges.

In order to compare the algorithm against LBP and DEEPS, we use the same experimental setup as employed in [6]. We conduct the simulation with 25 targets randomly deployed,

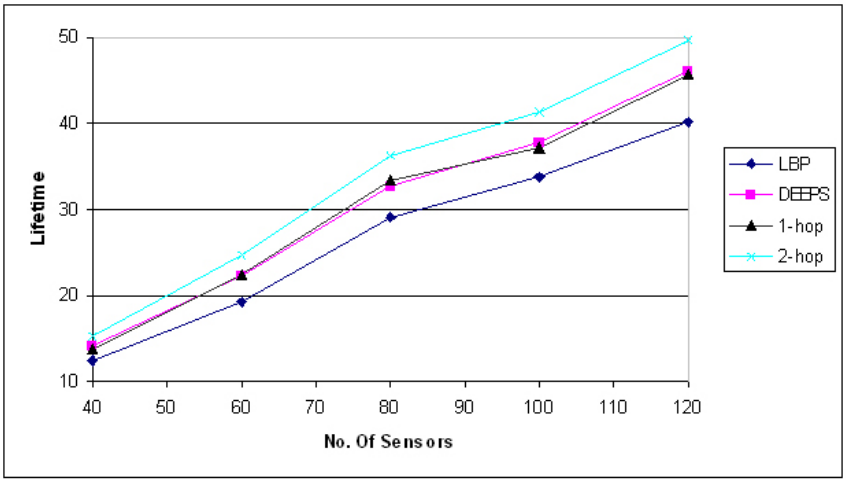

Figure 3.4. Network Lifetime for 25 targets with linear energy model

and vary the number of sensors between 40 and 120 with an increment of 20 and each sensor has a maximum sensing range (diameter) of $60m$. The energy consumption model is linear. The results are shown in Figure 3.4. As can be seen from the figure, both the basic 1-hop and 2-hop algorithms outperform LBP. The 1-hop algorithm is almost as good as the 2-hop DEEPS algorithm despite using much smaller number of messages.

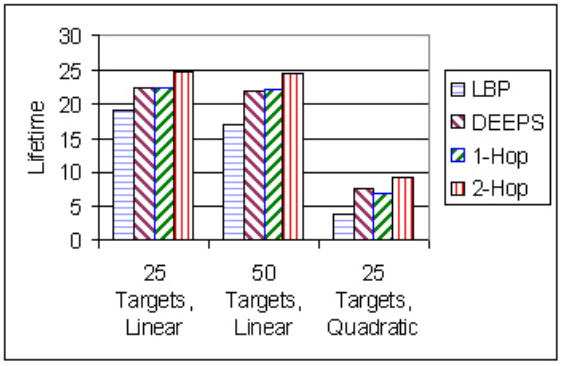

Figure 3.5. Lifetime for 60 sensors with varying energy model and targets

To study variations in targets and energy models, we simulate a network of 60 sensors with $60m$ sensing range for both 25 and 50 targets and linear and quadratic energy models. The results are presented in Figure 3.5. We see a trend consistent with the previous plot

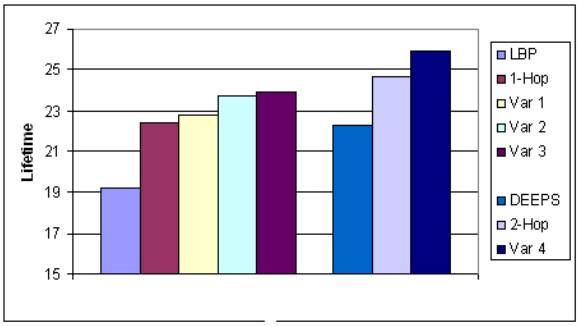

Figure 3.6. Performance of Variants of the Basic Algorithm: Network Lifetime for 60 sensors, 25 targets, linear energy model

with LBP being outperformed by the 1-hop and 2-hop algorithms and DEEPS and the 1-hop version showing similar lifetimes.

Finally, we simulate the different variants of the basic algorithm as outlined in Section 3.4 for a network of 60 sensors, 25 targets and a linear energy model. The results are shown in Figure 3.6 below. For the sake of comparison, we include the basic 1-hop and 2-hop algorithms in this plot also. We compare the 1-hop algorithm and its three variants against LBP. The percentage improvement for each algorithm against LBP is indicated on top of the bars. Similarly, we compare the 2-hop algorithm and Variant 4 against DEEPS. Overall improvements are in the $11 - 19\%$ range.

CHAPTER 4.

DISTRIBUTED ALGORITHMS BASED ON PROPERTIES OF AN OPTIMAL SCHEDULE

In Chapter 3, we saw the definition of the LD Graph and some simple heuristics based on the degree of nodes in this graph. In this chapter, we present additional heuristics that are based on certain properties of how the LD Graph would be used in an optimal schedule. It is important to realize that the prioritization phase of the basic algorithm is simply an ordering of the local covers. By using criteria other than the degree, it is possible to prioritize these covers differently. The heuristics we design in this chapter prioritize the choice of covers in the LD graph by considering the above mentioned properties of an optimal schedule.

The key motivation behind our approach in this chapter has been to start with the question of what properties an optimal schedule (henceforth called OPT) exhibits and how can we relate these properties to the LD graph. Digging deeper into the problem structure, our approach in this chapter has been to start with the question of what an optimal schedule would do with the lifetime dependency graph. We have been able to prove certain basic properties of the OPT schedule that relate to the LD graph. Based on these properties, we have designed algorithms which choose the covers that exhibit these OPT schedule like properties. We present three new heuristics - *Sparse-OPT* based on the sparseness of connectivity among covers in OPT, *Bottleneck-Target* based on selecting covers that optimize the use of sensors covering local bottleneck targets, and *Even-Target-Rate* based on trying to achieve an even burning rate for all targets. These heuristics are, therefore, at a higher level and operate on top of degree-based heuristics to prioritize the local covers. Our experiments show an improvement in lifetime of 10-15% over our basic heuristics presented in Chapter 3 and 25-30% over competing work in [6, 7]. We also implement two-hop versions of these heuristics and show that they give around 35% improvement over the two-hop algorithm of [7]. The material presented in this chapter was published in [24].

4.1 Definitions

In addition to the notation introduced in Section 3.1, we introduce the following terms:

• $lt(t_i)$: The lifetime of a target $t_i \in T$ is given by $lt(t_i) = \sum_{\{s|t_i \in T(s)\}} b(s)$.

• Bottleneck Sensor: Bottleneck sensor s of cover C is the sensor $s \in C$ with minimum battery, i.e., it is the sensor s that upper bounds $lt(C)$.

• Bottleneck Target (t_{bot}): The target with the smallest lifetime $lt(t_{bot})$.

• Lifetime of a schedule of covers: We can view the set of currently active sensors as a cover C_i that is used for some length of time l_i. After this time, when we shuffle the set of active sensors, we get a new cover set C_j that is then used for some time l_j. Note that C_i and C_j may have sensors in common. Hence, we obtain a schedule of covers of the form,

$(C_1, l_1), (C_2, l_2), ..., (C_r, l_r)$.

The lifetime of this schedule is given by $\sum_{i=1}^{r} l_i$

• OPT: The optimal schedule of covers that achieves the maximum lifetime. Note that this includes both the covers and their corresponding time periods.

4.2 Properties of the Optimal Sequence

In this chapter, our approach to the problem of further extending the lifetime of the network has been to turn the argument on its head. Instead of trying to come up with different schemes of scheduling sensors into cover sets, let us suppose we know the best schedule. Such a schedule OPT comprises a collection of covers and the time for which each cover is used in the optimal schedule,

$$OPT = \{(C_{opt_1}, l_1), (C_{opt_2}, l_2), ..., (C_{opt_r}, l_r)\}.$$

The optimal sequence can be viewed as a partition of the space of covers into those covers that are in OPT and those that are not. Now visualize all possible cover sets in the LD graph. How would one identify from amongst all these sets, those that are candidates for being in OPT? Our goal is to identify certain properties of the OPT that can help make this decision.

Lemma 4.2.1. *OPT burns all the covers.*

Proof: *Suppose there is a cover C' that is not burned by OPT, i.e., its weakest sensor has some battery left. Burning this cover would further increase the lifetime of the network by an amount $lt(C')$. This implies that OPT is non-optimal, which gives us a contradiction.*□

Lemma 4.2.2. *If a cover C is not used in OPT, C has at least one neighboring cover in the LD Graph in OPT.*

Proof: *If a cover C is not used in OPT, then it has at least one sensor $s \in C$ that has exhausted its battery. If this is not true, then C still has some lifetime left and this gives us a contradiction according to Lemma 1. The fact that sensor s has been completely burned for cover C implies that there is at least one other cover in OPT that contains this sensor s. Otherwise, s would not be burned. This means that there is at least one cover which is a neighbor of C in the LD graph (because they share the sensor s) and is also in OPT.* □

Corollary: If a cover C is not used in OPT, C has one or more neighboring covers in OPT such that total span of these neighbors in OPT is at least the life of C.

Lemma 4.2.3. *The covers in OPT form a dominating set of the LD Graph.*

Proof: *A dominating set of a graph is a set of vertices such that every vertex is either in this set or is a neighbor of a node in the dominating set.*

Let us consider a cover C. There are two possibilities. Either it is in OPT, in which case it is a dominating node, or it is not in OPT. By Lemma 2, if it is not in OPT, it has to have at least one neighbor in OPT. Hence it is a dominated node. Hence, the covers in OPT dominate the set of all covers in the LD graph. □

Lemma 4.2.4. *All permutations of OPT are optimal.*

Proof: *This lemma shows that the ordering of individual covers in the OPT sequence is irrelevant. Any permutation OPT' of OPT can be obtained by repeatedly moving (C_{OPT_i}, l_i) to its position j in OPT' for all eligible i. We prove that each such move preserves optimality.*

Moving C_{OPT_i} to position j changes its relative ordering with its neighboring covers in the LD graph which lie between position i and j. Let us call these neighbors $N'(C_{OPT_i})$. Other neighbors of C_{OPT_i} do not see any change, nor do other covers which are not neighbors. For each neighboring cover $C_{OPT_k} \in N'(C_{OPT_i})$, $w(C_{OPT_i} \cap C_{OPT_k}) \geq l_i + l_k$, as each edge upper bounds the cumulative lifetime. Therefore, if $i > j$ burning C_{OPT_i} before C_{OPT_k} will leave $w(C_{OPT_i} \cap C_{OPT_k}) - l_i \geq l_k$ of battery in $C_{OPT_i} \cap C_{OPT_k}$. Therefore, C_{OPT_k} can be burnt for a duration of l_k, for each $C_{OPT_k} \in N'(C_{OPT_i})$.

On the other hand, if $i < j$, each $C_{OPT_k} \in N'(C_{OPT_i})$ will be burnt for l_k time, leaving $w(C_{OPT_k} \cap C_{OPT_i}) - l_k \geq l_i$ of battery in $C_{OPT_k} \cap C_{OPT_i}$. Thus, C_{OPT_i} can be burnt for duration of l_i at position j. □

Corollary: If C occurs more than once in OPT, all its occurrences can be brought together, thereby burning C all at once for the cumulative duration.

Lemma 4.2.5. *Due to OPT, all sensors around at least one target are completely burnt.*

Proof: *This relates to Lemma 1, because if there is no such bottleneck target, then it is still possible to cover all targets, implying that a cover exists, and hence OPT is not optimal.* □

4.3 Optimal Schedule based Algorithms

Recall from our discussion on the Lifetime Dependency Graph model in Section 3.2 that we used the degree $d(C)$, sum of the edge weights, of a cover C in order to determine its priority. The defining of a priority function is a key step in our algorithms because it determines the order in which covers are used in the *Negotiation* phase.

The definition of the degree as the key prioritization criteria is limited since it only considers a local property of the LD graph. As we saw in the previous section, there are properties which show how an OPT schedule would choose covers in the LD graph. Our goal in this section is to design heuristics that utilize these properties in the prioritization phase of the algorithm described in Section 3.3. We introduce three heuristics based on the properties of the OPT schedule. Each of these heuristics define a different way to prioritize the local covers. Note that if all remaining covers are tied on the new priority functions, we revert to using the degree to break ties. Hence, these heuristics can be viewed as defining a higher level priority function, that acts on top of the degree $d(C)$ function.

Heuristic 1: Sparse-OPT

This heuristic is based on *Lemma* 2 and works on the idea that the covers in OPT are sparsely connected. Suppose we have a subsequence of OPT available and we were to pick a cover to add to this sequence. Clearly any cover we add to the OPT sequence should not be a neighbor of a cover that is already in OPT. By *Lemma* 2, we know that if a cover C is in OPT for time l, then its neighbors in the LD graph can only be in OPT if their shared sensor s has a battery $b(s) > l$. This implies that covers in OPT are likely to be sparsely connected. Hence, for any two nodes (covers) in OPT, their degree in the induced subgraph of the LD graph should be low.

For a cover C, we define its degree to other covers already selected (in OPT) as:

$$d_{OPT}(C) = \Sigma_{e \text{ incident to } C \text{ and to } C' \in OPT} \; w(e)$$

This leads us to a simple heuristic: *When choosing a cover C_i, pick the cover with the lowest degree to nodes already chosen.*

Implementation: When implementing this heuristic, the exchange of messages during the setup phase remains unchanged from before. As explained in Section 3.3, recall that in the setup phase, a sensor s exchanges information on its battery $b(s)$ and the targets it covers $T(s)$ with its neighbors. This information is then used to compute all local covers for targets in $T(s)$, and construct an LD graph for these.

The next step in the algorithm would be to prioritize the covers in the local LD graph. Instead of prioritizing a cover C by its degree $d(C)$, the heuristic we defined gives us a higher level of prioritization. Initially, there are no covers that have been selected for use so the heuristic starts off as before. A sensor orders its local covers by $d(C)$ (battery and id's can be used to break ties), and then enters a *negotiation* phase with its neighbors in order to decide which cover to use. This would result in some cover C' being selected for use.

In the subsequent round the sensor recomputes the priority function for its covers. Now, in computing the priority of a cover C, we can look at its degree to the previously used cover C', given by $d_{OPT}(C)$, and prioritize the covers in the order of lowest degree first. Note that if d_{OPT} is the same for several covers, we can break ties by using $d(C)$ as before. As the set of previously used covers increases over time, we compute the priorities at the beginning of each round by looking at the d_{OPT} of any cover to this set and assigning a higher priority to the covers with the lowest degree.

Heuristic 2: Bottleneck-Target

This heuristic is based on the property that covers in OPT should optimize the local bottleneck target and makes use of the ideas presented in *Lemma* 5.

Let us consider the set of all targets T. Some of these targets are more important than others because they represent bottlenecks for the lifetime of the network. Consider a target t_i. Then, the total amount of time this target can be monitored by any schedule is given by:

$$lt(t_i) = \sum_{\{s \mid t_i \in T(s)\}} b(s)$$

Clearly there is one such target with the smallest $lt(t_i)$ value, that is a bottleneck for the entire network. Since the network as a whole cannot achieve a lifetime better than this bottleneck, it follows that any cover should not use multiple sensors from the set of sensors that cover this bottleneck. However, without a global picture, it is not possible for any sensor to determine if one of its local targets is this global bottleneck.

However, every sensor is aware of which of its local targets is a local bottleneck. The key thing to realize is the fact that if every sensor optimizes its local bottleneck target, then one of these local optimizations is also optimizing the global bottleneck target.

Let t_{bot} be this local bottleneck target. Let C_{bot} be the set of sensors that can cover this local bottleneck target.That is,

$$C_{bot} = \{s \mid t_{bot} \in T(s)\}$$

Ideally, we would like to use a cover that has only one sensor in C_{bot} as a part of this cover. However this may not always be possible. Hence, while prioritizing the local covers, we can pick a cover that minimizes the cardinality of its intersection with C_{bot}.

Hence, the cover selected should be the cover C that minimizes $|C \cap C_{bot}|$.

Implementation: The implementation is similar to Heuristic 1. Again, the setup and LD graph construction phase remain unchanged. Every sensor s now computes its local bottleneck target $t_{bot} \in T(s)$ and the set of sensors that covers this target C_{bot}. Note that this can be done since at the end of the setup phase, every sensor knows who its neighbors are, which targets they cover, and how much battery they have.

When calculating the priority of each cover C in the LD graph, the priority function defined by this heuristic calculates the value of $|C \cap C_{bot}|$ for each cover, since this gives us the number of sensors in C_{bot} that are a part of the cover C. The heuristic then prioritizes covers in descending order of this cardinality. Again, if this value is the same for multiple covers, we break ties by using the degree $d(C)$.

Heuristic 3: Even-Target-Rate

This heuristic is based on the idea that OPT should try and burn all targets at the same rate. Consider a target $t_i \in T$. Let $lt(t_i)$ be the sum of the battery of all sensors covering t_i. Clearly the network cannot be alive for longer than $lt(t_{bot})$ where t_{bot} is the target with the smallest lifetime. Heuristic 2 stated above tries to maximize the time this bottleneck can be used. However, the danger in this is that by doing so, a different target may become the

bottleneck due to repeated selection of its covering sensors. To avoid this problem, and at the same time optimize the bottleneck target, we want to keep an even rate of burning for all targets. In order to arrive at a normalized burning rate for each target t_i, we define the impact of a cover C on a target t_i as given by,

$$Impact(C, t_i) = \frac{|\{\ s \in C \mid t_i \in T(s)\}|}{lt(t_i)}$$

The *Impact* should give a measure of how this cover C is reducing the lifetime of a target t_i. Since each round lasts for one time unit, the impact is measured by the number of sensors covering t_i that are in C. Hence, the definition. This gives us the heuristic:

Any cover chosen should be the best fitting cover in that it burns all targets at an even rate.

Implementation: After setup and constructing its LD Graph as before, each sensor s enters the prioritization phase. For every target being considered, s computes the impact of a cover C on that target, i.e., the sensor s calculates $Impact(C, t_i)$ for all $t_i \in T(s)$. Let $Impact_{max}(C)$ be the highest impact of this cover C and let $Impact_{min}(C)$ be the lowest impact of C for all targets. A good cover will have a similar impact on all targets, since it burns them at the same normalized rate. Hence, we prioritize covers in descending order of this difference, given by $Impact_{max}(C) - Impact_{min}(C)$. Once again, if the impact is the same, we can break ties using $d(C)$.

4.4 Experimental Evaluation

In this section, we first evaluate the performance of the one-hop and two-hop versions of the three heuristics based on OPT schedule as compared to the 1-hop algorithm LBP [6], the 2-hop algorithm DEEPS [7], and our basic Degree-Based heuristic from our previous work in [55]. Next, we create appropriate hybrids of the three 1-hop heuristics in various combinations, showing that all three combined together yield around 30% improvement over 1-hop LBP and 25% over the 2-hop DEEPS algorithm (Fig. 4.1 and 4.2). Even though

theoretically our algorithms are exponential in the number of targets, practically their computation time was no more than three times the LBP algorithm with same communication complexities. We also implement the 2-hop versions of these heuristics and obtain a 35% improvement in lifetime over DEEPS (Fig. 4.3). As compared to an upper bound on the longest network lifetime (the lifetime of the global bottleneck target), our 1-hop algorithms have moved the state of art to no worse than 30-40% on an average (Fig. 4.4). The 2-hop algorithms are no worse that 25% on an average.

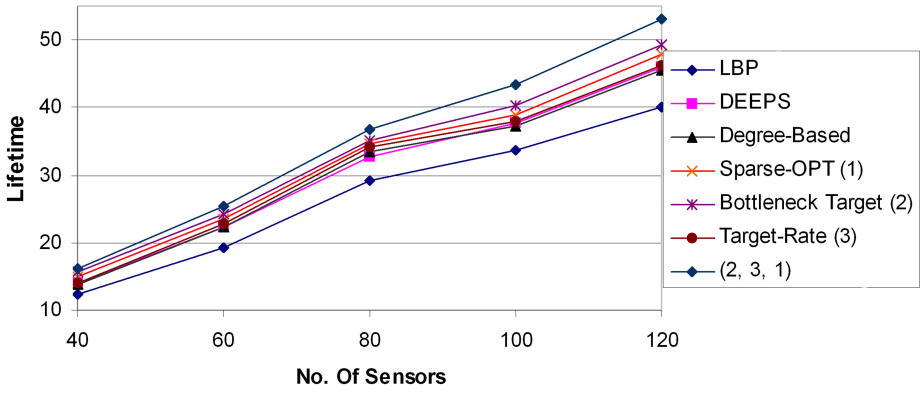

Figure 4.1. Lifetime with 25 Targets

In order to compare the algorithm against LBP, DEEPS, and our previous work, we use the same experimental setup and parameters as employed in [6]. We carry out all the simulations using $C++$. For the simulation environment, a static wireless network of sensors and targets scattered randomly in $100m \times 100m$ area is considered. We conduct the simulation with 25 targets randomly deployed, and vary the number of sensors between 40 and 120 with an increment of 20 and each sensor with a fixed sensing range of $60m$. We assume that the communication range of each sensor is two times the sensing range [69, 66]. For these simulations, we use the linear energy model wherein the power required to sense a target at distance d is proportional to d. We also experimented with the quadratic energy model (power proportional to d^2). The results are similar to [55] and the same trends as the

linear model apply. Due to space constraints, we only show a snapshot of these results in Fig. 4.4.

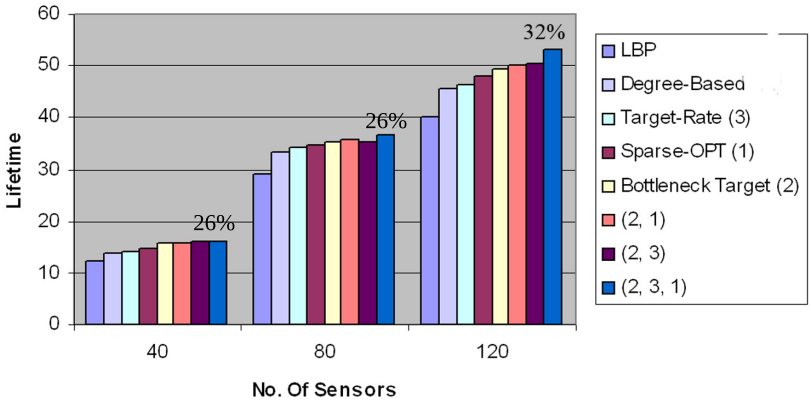

Figure 4.2. Comparing LBP [3] against the 1-hop OPT-based heuristics

The results are shown in Fig. 4.1. As can be seen from the figure, among the three heuristics, the *Bottleneck-Target* heuristic performs the best giving about 10-15% improvement in lifetime over our previous 1-hop Degree-Based heuristic and about 25-30% over LBP and DEEPS. *Sparse-OPT* and *Even-Target-Rate* also improve over the Degree-Based heuristic.

We also ran simulations with a combination of the heuristics. $(2, 1)$ denotes the combination of heuristic 2, Bottleneck-Target, followed by heuristic 1, Sparse-OPT, and so on. For these experiments, the priority function was modified to implement the priority function of both heuristics in the same algorithm. For example, in $(2, 1)$, we first optimize the local bottleneck target (based on heuristic 2) and in case of ties, we break them by checking the degree to previously selected covers (based on heuristic 1). Other implementations follow similarly. The combination of heuristics give a better lifetime than the individual heuristics; the best hybrid was (2,3,1) plotted in Fig. 4.1.

In Fig. 4.2, we highlight the performance of the three heuristics as compared to LBP. The simulation is run for 40, 80 and 120 sensors, with 25 targets and a linear energy model. As can be seen, a similar trend in improvements is observed. Overall lifetime improvements are

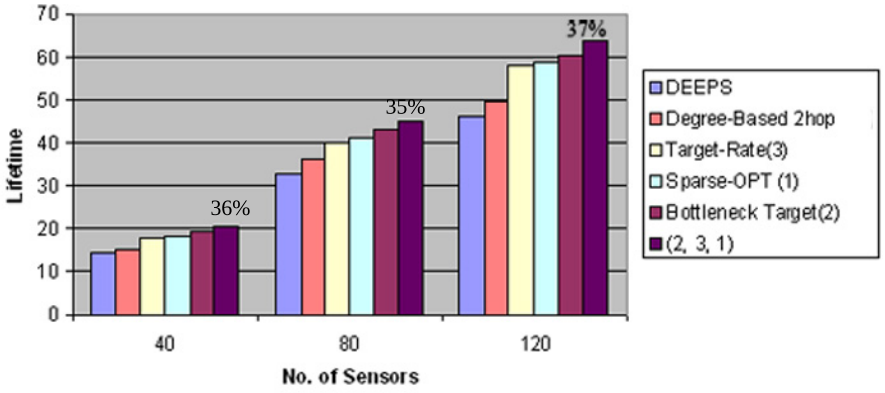

Figure 4.3. Comparing DEEPS [7] against the 2-hop OPT-based heuristics

in the range of 25-30% over LBP as the baseline. The heuristics are presented in ascending order of performance in the figure.

Since DEEPS is a 2-hop algorithm, we also compared 2-hop versions of our proposed heuristics, where the target set $T(s)$ of each sensor is expanded to include $\cup_{s' \in N(s,1)} T(s')$ and the neighbor set is expanded to all 2-hop neighbors, i.e., $N(s, 2)$. The results are shown in Fig. 4.3. As can be seen, the 2-hop versions give a further gain in lifetime over DEEPS, with overall improvement of 35% for the hybrid of the three heuristics.

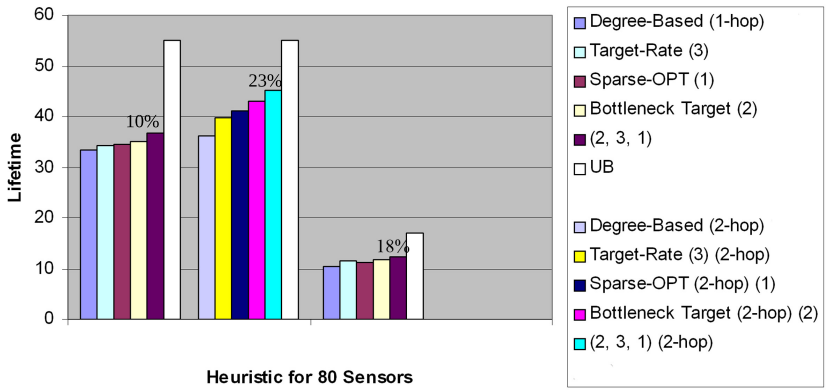

Figure 4.4. Comparing 1-hop and 2-hop Degree-Based [55] heuristics against OPT-based heuristics

Finally, we highlight the comparison of the proposed heuristics against our previous work [55] in Fig. 4.4. We show results here for the median value of $n = 80$ sensors. The two

series compare the 1-hop version of the Degree-Based heuristic against the 1-hop version of the OPT heuristics and then repeat these comparisons for the 2-hop versions of both. Overall, we achieve gains of 10-15% for the 1-hop heuristics and between 20-25% for the 2-hop heuristics. We also show (i) how far these heuristics are compared to the upper bound on network lifetime, and (2) that both linear and quadratic energy models follow similar trends.

In this chapter, we address the problem of scheduling sensors to extend the lifetime of a Wireless Sensor Network. We examine the properties that an optimal schedule would exhibit and use these to design three new heuristics that work on the lifetime dependency graph model. Simulations show an improvement of 25-30% over the algorithms in [6, 7] and 10-15% over our previous work in [55] for the 1-hop algorithms. Two-hop versions show additional improvements over their counterparts. These heuristics are designed to work on higher level properties of the dependency graph. The net effect is a significant improvement in the state of art, with our algorithms performing no worse than 30-40% compared to the optimal network lifetime on an average.

CHAPTER 5.

TAMING THE EXPONENTIAL STATE SPACE OF THE MAXIMUM LIFETIME SENSOR COVER PROBLEM

In the previous two chapters, we have introduced several distributed algorithms based on the concept of a *Lifetime Dependency* (LD) Graph. Our motivation was to study underlying properties of the problem and develop distributed heuristics that make use of these properties. If we consider the LD graph, it is quickly obvious that even creating this graph will take exponential time since there are 2^n cover sets to consider where, n is the number of sensors. However, the target coverage problem has a useful property - if the local targets for every sensor are covered, then globally, all targets are also covered. In [55], we make use of this property to look at the LD graph locally (fixed 1-2 hop neighbors), and are able to construct all the local covers for the local targets and then model their dependencies. Based on these dependencies, a sensor can then prioritize its covers and negotiate these with its neighbors. Simple heuristics based on properties of this graph were presented in [55] and showed a 10-15% improvement over comparable algorithms in the literature. [24] built on this work by examining how an optimal sequence would pick covers in the LD graph and designing heuristics that behave in a similar fashion. Though the proposed heuristics are efficient in practice, the running time is a function of the number of neighbors and the number of local targets. Both of these are relatively small for most graphs but theoretically are exponential in the number of targets and sensors.

A key issue that remains unresolved is the question of how to deal with this exponential space of cover sets. In this paper we present a reduction of this exponential space to a linear one based on grouping cover sets into *equivalence classes*. We use $[C_i]$ to denote the equivalence class of a cover C_i. The partition defined by the equivalence relation on the set of all sensor covers Given a set C and an equivalence relation $\Re$, the equivalence class of an element $C_i \in C$ is the subset of all elements in C which are equivalent to C_i. The notation

used to represent the equivalence class of C_i is $[C_i]$. In the context of the problem being studied, C is the set of all sensor covers and for any single cover C_i, $[C_i]$ represents all other covers which are *equivalent* to C_i as given by the definition of some equivalence relation $\Re$. Our approach stems from the understanding that from the possible exponential number of sensor covers, several covers are very similar, being only minor variations of each other. In Section 5.1, we present the definition of the relation $\Re$, based on a grouping that considers cover sets equivalent if their lifetime is bounded by the same sensor. We then show the use of this relation to collapse the exponential LD Graph into an *Equivalence Class* (EC) Graph with linear number of nodes. This theoretical insight allows us to design a sampling scheme that selects a subset of all local covers based on their equivalence class properties and presents this as an input to our simple LD graph degree-based heuristic. Simulation results show that class based sampling cuts the running time of these heuristics by nearly half, while only resulting in a less than 10% loss in quality.

5.1 Dealing with the Exponential Space

In this section, we present our approach of dealing with the exponential solution space of possible cover sets. The next section utilizes these ideas to develop heuristics for maximizing the lifetime of the network. Even though the total number of cover sets for the network may be exponential in the number of sensors, for any given cover set, there are several other sets that are very *similar* to this set. We begin by attempting to define this notion of similarity by expressing it as an equivalence relation.

Definition 1: Let $\Re$ be an equivalence relation defined on the set of all sensor covers such that $C_i \Re C_j$ if and only if C_i and C_j share the same bottleneck sensor s_{bot}.

Theorem: $\Re$ is an equivalence relation

Proof: $\Re$ is reflexive, since $C_i \Re C_i$. $\Re$ is symmetric, since if $C_i \Re C_j$ then, $C_j \Re C_i$ since both covers C_i and C_j share the same bottleneck sensor. Finally, if $C_i \Re C_j$ and $C_j \Re C_k$, then $C_i \Re C_k$ and $\Re$ is transitive since if C_i shares the same bottleneck sensor with C_j and

C_j shares the same bottleneck sensor with C_k then, clearly both C_i and C_k have the same bottleneck sensor in common. Therefore, $\Re$ is and equivalence relation. $\square$

Every equivalence relation defined on a set, specifies how to partition the set into subsets such that every element of the larger set is in exactly one of the subsets. Elements that are related to each other are by definition in the same partition. Each such partition is called an *equivalence class*. Hence, the relation $\Re$ partitions the set of all possible sensor covers into a number of *disjoint* equivalence classes.

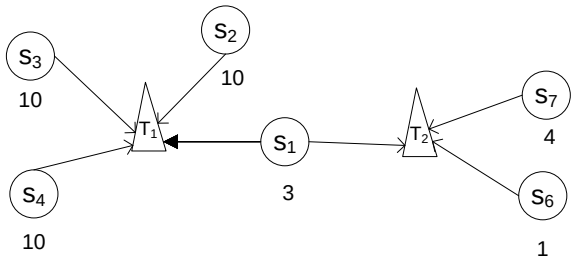

Figure 5.1. Example Sensor Network

Notation: Henceforth, we represent the equivalence class of covers sharing a bottleneck sensor s_i by $[s_i]$. Note that this is a slight abuse of notation since s_i is not a member of this class, but is instead the property that is common to all members of this class. Hence, $[s_i]$ can be read as the equivalence class for all covers having sensor s_i as their bottleneck sensor.

We now define what we would call the Equivalence Class (EC) Graph. Each node of this graph represents an equivalence class. Just as the LD graph models the dependency between sensor covers, the EC Graph models the dependency between classes of covers.

Definition 2: Equivalence Class Graph (EC). The Equivalence Class graph $EC = (V', E')$ where, V' is the set of all possible equivalence classes defined by $\Re$ and two classes $[s_i]$ and $[s_j]$ are joined by an edge for every cover in each class that share some sensor in common. Hence, the graph EC is a multi-edge graph.

The cardinality of the vertex set of the Equivalence Class Graph is at most n. This result follows from the observation that for any network of n sensors, there can be at most one

equivalence class corresponding to each sensor, since every cover can have only one of the n sensors as its bottleneck (in case two or more sensors all have the same battery and are the bottleneck, sensor id's can be used to break ties).

To better understand these definitions, let us consider an example. Consider the sensor network shown in Figure 5.1. The network comprises of seven sensors, $s_1, ..., s_7$ and two targets, T_1, T_2. Observe that T_2 is the bottleneck target for the network since it is the least covered target (8 units of total coverage compared to 33 for T_1). Also note that only one sensor, s_1 can cover both targets.

For the given network, the set of all possible minimal sensor covers, S is,

$S = \{\{s_2, \mathbf{s_6}\}, \{s_2, \mathbf{s_7}\}, \{s_3, \mathbf{s_6}\}, \{s_3, \mathbf{s_7}\}, \{s_4, \mathbf{s_6}\}, \{s_4, \mathbf{s_7}\}, \{\mathbf{s_1}\}\}$

For each individual cover in this set, the bottleneck sensor is the sensor shown in bold face.

Figure 5.2 shows the Lifetime Dependency graph for these covers. As defined, an edge exists between any two covers that share at least one sensor in common and the weight of this edge is given by the lifetime of the common sensor having the smallest battery (the bottleneck). For example, an edge of weight 4 exists between C_1 and C_2 because they share the sensor s_2 having a battery of 4.

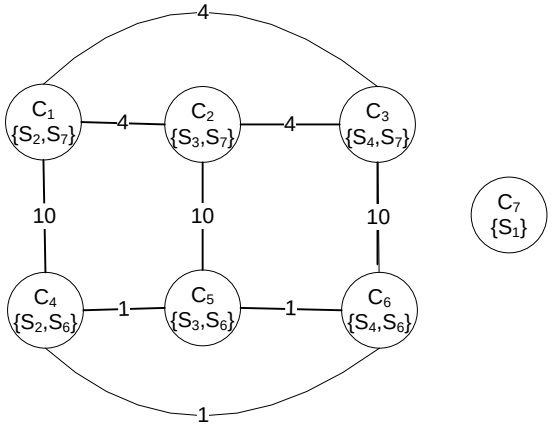

Figure 5.2. LD Graph for the example network

To obtain the EC Graph from this LD Graph, we add a node to represent the equivalence class for each sensor that is a bottleneck sensor for any cover. For the above example, given all sensor covers in the set S, there are three sensors s_1, s_6, s_7 that are each the bottleneck for one or more covers in S. Hence, the EC Graph is a three node graph. Figure 5.3 shows the complete EC Graph for the covers in S. There is a node corresponding to the equivalence class for each of the three sensors s_1, s_6, s_7 and for each cover in the class we retain edges to the class corresponding to the bottleneck sensor of the cover on which the edge terminated in the LD graph. Hence, we have three edges between the nodes s_6 and s_7.

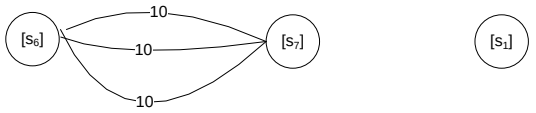

Figure 5.3. EC Graph for the example network

It is key to realize that the EC graph is essentially an encapsulation of the LD Graph that can have at most n nodes. This view is presented in Figure 5.4, where we show the LD Graph that is embedded into the EC Graph. Each rectangular box shows the nodes in the LD graph that are in the same equivalence class. This figure also illustrates our next theorem.

Theorem: For sensor covers in the same equivalence class, the induced subgraph on the LD Graph is a clique

Proof: This theorem states that for the nodes in the LD graph that belong to the same class, the induced subgraph is a clique. Since by definition, all sensor covers in a class $[s]$ share the sensor s as their bottleneck sensor, the induced subgraph will be a complete graph between these nodes. □

Also, a subtle distinction has been made between inter-class edges and intra-class edges in going from the LD graph to the EC graph.

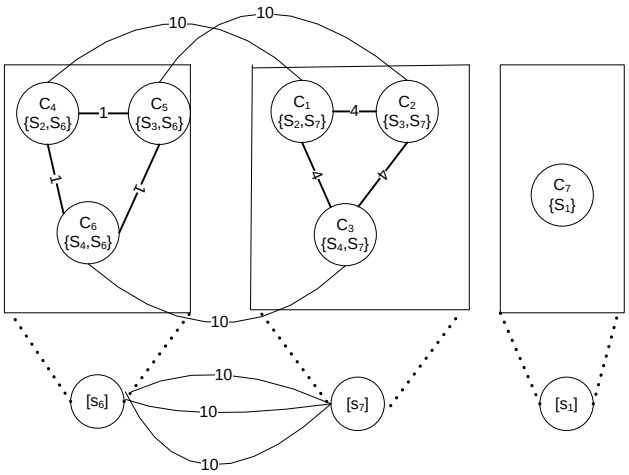

Figure 5.4. EC Graph for the example network along with the LD Graph embedded in it

5.2 Sampling Based on the Equivalence Class Graph

The previous section defined the concepts behind reducing the exponential space of covers in the LD Graph to the linear space of the EC Graph. In this section, we build on these concepts to discuss techniques for generating a limited number of covers for the LD Graph. Specifically, our goal is to improve the timing performance of the distributed algorithms we presented in [55, 24]. As presented, the EC Graph is not very useful since it still requires the exponential LD graph to be populated, before it can be constructed. However, by realizing that the exponential space of cover sets can be expressed in this linear space of equivalence classes, we can generate only a subset of the set of all covers.

Recall from Chapter 3 that even though the number of global sensor covers is exponential in the number of sensors, our heuristics presented in [55, 24] worked by constructing *local* covers. After exchanging one or two hop coverage information with neighboring sensors, a sensor can exhaustively construct all possible local covers. A local cover here is a sensor cover that covers all the local targets. The number of local covers is also exponential but is determined by the maximum degree of the graph and the number of local targets, typically much smaller values than the number of all sensors or targets. The heuristics then construct

the LD graph over these local covers. The choice of which cover to use is determined by looking at properties of the LD graph such as the degree of each cover in the LD graph.

By making use of the idea of related covers in the same equivalence class, our goal is to use our existing heuristics from [55, 24] but to modify them to run over a subset of the local covers as opposed to all local covers. This should give considerable speedup and if the subset is selected carefully, it may only result in a slight reduction of the overall lifetime. We present such a local cover sampling scheme in Section 5.2.1 and then present the modified basic algorithm of [55, 24] to operate on this sample in Section 3.3. Finally, we evaluate the effectiveness of sampling in Section 5.3.

5.2.1 Local Bottleneck Target based generation of local cover sets

Understanding the underlying equivalence class structure, we now present one possible way of generating a subset of the local cover sets. Our approach is centered around the bottleneck target. For any target t_i, the total amount of time this target can be monitored by any schedule is given by:

$$lt(t_i) = \sum_{\{s \mid t_i \in T(s)\}} b(s)$$

Clearly there is one such target with the smallest $lt(t_i)$ value, and is hence a bottleneck for the entire network [60]. Without global information it is not possible for any sensor to determine if the global bottleneck is a target in its vicinity. However, for any sensor s, there is a least covered target in $T(s)$ that is the *local* bottleneck. A key thing to realize is the fact that the global bottleneck target is also the local bottleneck target for the sensors in its neighborhood. Hence, if every sensor optimizes for its local bottleneck target, then one of these local optimizations is also optimizing the global bottleneck target. We use t_{bot} to denote this local bottleneck target. Let C_{bot} be the set of sensors that can cover this local bottleneck target.That is,

$$C_{bot} = \{s \mid t_{bot} \in T(s)\}$$

Implementation: This understanding of bottleneck targets, along with our definition of equivalence classes, now gives us a simple means to generate local covers. Since no coverage schedule can do any better than the total amount of time that the global bottleneck can be covered, instead of trying to generate all local covers, what we really need are covers in the equivalence classes corresponding to each sensor $s_i \in C_{bot}$, such that each class can be completely exhausted. Also, to only select covers that conserve the battery of the sensors in C_{bot}, we want to ensure that the covers we generate are disjoint in C_{bot}. In terms of equivalence classes, for any two classes $[s_i]$ and $[s_j]$ such that $s_i, s_j \in C_{bot}$, we want to generate cover sets that are in these classes but do not include *both* s_i and s_j.

To generate such cover sets, we can start by picking only one sensor s'_{bot} in C_{bot}. This ensures that the local bottleneck target is covered. For each target t_i in the one/two-hop neighborhood being considered, we can then randomly pick a sensor s, giving preference to any $s \notin C_{bot}$. Note that this does not necessarily create a sensor cover in the class $[s'_{bot}]$, since any one of our randomly picked sensors could be the bottleneck for the cover generated. However, replacing that sensor with another randomly picked sensor that covers the same target ensures that the we finish by using a cover in $[s'_{bot}]$. Such a selection essentially ensures that we burn the entire battery of this sensor s'_{bot} in C_{bot} through different covers, while trying to avoid using other sensors in C_{bot}. This process is then repeated for every sensor in C_{bot}. Hence, instead of generating all local covers, we only generate a small sample (constant number) of these corresponding to the equivalence class for each sensor covering the bottleneck target and some related randomly picked covers. We already showed that there can be at most n equivalence classes for the network. Thus, the sampled graph generated has O(n) nodes. If we consider the maximum number of sensors covering any target as a constant for the network, sampling only takes cumulative time of $O(n\tau)$, where $\tau = max_{s \in S}|T(s)|$, since we do this for n sensors, each of which has a maximum of τ targets to cover, which are in turn covered by a constant number of sensors (as per our assumption). Even if this assumption is removed, in the worst case, all n sensors could be covering the same target

making the time complexity $O(n^2\tau)$. Next, we run our basic heuristic from [55] on this sampled LD graph.

5.3 Performance Evaluation

In this section, we evaluate the performance of the proposed sampling scheme and evaluate it against our degree based heuristics of [55]. By not constructing all local covers and instead constructing a few covers for key equivalence classes, we should achieve considerable speedup. But the effectiveness of sampling can only be evaluated by analyzing its tradeoff between faster running time for possible reduced performance. The objective of our simulations was to study this tradeoff. For completeness, we create both one-hop and two-hop versions of our sampling heuristic and also compare its performance to two other algorithms in the literature, the 1-hop algorithm LBP [6] and the 2-hop algorithm DEEPS [7]. Details of both these algorithms are given in Section 2.

In order to compare the equivalence class based sampling against our previous degree based heuristics, LBP, and DEEPS, we use the same experimental setup and parameters as employed in [6]. We carry out all the simulations using $C++$. For the simulation environment, a static wireless network of sensors and targets scattered randomly in $100m \times 100m$ area is considered. We conduct the simulation with 25 targets randomly deployed, and vary the number of sensors between 40 and 120 with an increment of 20 and each sensor with a fixed sensing range of $60m$. The communication range of each sensor assumed to be two times the sensing range [69, 66]. For these simulations, we use the linear energy model wherein the power required to sense a target at distance d is proportional to d. We also experimented with the quadratic energy model (power proportional to d^2). The results showed similar trends to those obtained for the linear model.

Figure 5.5 shows the Network Lifetime for the different algorithms. As can be seen from the figure, the sampling heuristics is only between 7-9% worse than the degree based heuristic. Sampling also outperforms the 1-hop LBP algorithm by about 10%. It is interesting to

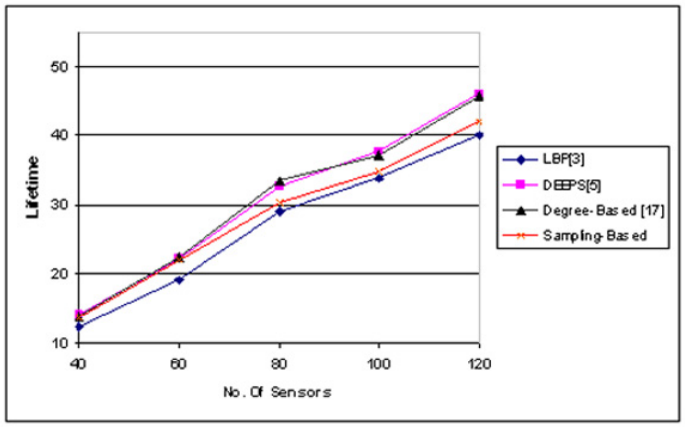

Figure 5.5. Comparison of Network Lifetime with 25 Targets

observe that for smaller network sizes, sampling is actually much closer to the degree-based heuristics in terms of performance.

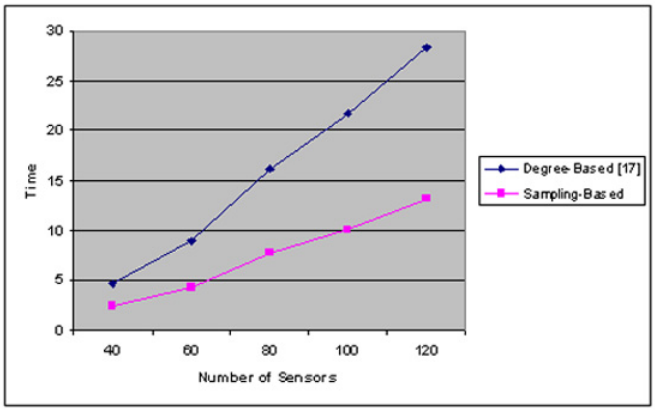

Figure 5.6. Comparison of Running Time with 25 Targets

Now that we have seen that sampling works well when compared to the degree based heuristic, the question that remains to be answered is how much faster is the sampling algorithm? Figure 5.6 compares head-to-head the running time for the degree based heuristic (potentially exponential in m) and the linear time sampling algorithm. As can be seen from the figure the running time for the sampling algorithm is about half of the running time for the degree-based heuristic.

Table 5.1. Comparison of Network Lifetime for 1-hop algorithms

Algorithm	n=40	n=80	n=120
LBP [6]	12.4	29.1	40.1
Degree-Based [55]	13.8	33.4	45.6
Sampling-Based	13.7	30.3	42.1
Randomized-Sampling	10.1	17.6	30.1

Table 5.2. Comparison of Network Lifetime of 2-hop algorithms

Algorithm	n=40	n=80	n=120
DEEPS [7]	14.1	32.7	46.1
Degree-Based (2-hop) [55]	15.2	36.2	49.6
Sampling-Based (2-hop)	14.4	33.4	47.5

Finally, we individually study the 1-hop (Table 5.1) and 2-hop (Table 5.2) sampling heuristics with comparable algorithms. For the 1-hop algorithms, we also include a randomized-sampling algorithm that makes completely random picks for each target, without considering properties of the equivalence classes. The intention is to ensure that the performance of our sampling-heuristic can be attributed to the selection algorithm. For the 2-hop versions of our proposed sampling heuristic, the target set $T(s)$ of each sensor is expanded to include $\cup_{s' \in N(s,1)} T(s')$ and the neighbor set is expanded to all 2-hop neighbors, i.e., $N(s, 2)$. Covers are now constructed over this set using the same process as before. As can be seen from the table, both the 1-hop and 2-hop version are under 10% worse than the comparable degree-based heuristics. Also, the 2-hop sampling slightly outperforms the DEEPS by a 5% improvement in network lifetime.

In this chapter, we introduced some key ideas on dealing with the exponential space of sensor covers for the maximum lifetime sensor scheduling problem. Our approach was based on defining a relation that partitions the set of all sensor covers into a linear number of disjoint partitions. We use this underlying theory to sample this exponential space efficiently and show that heuristics that use this sample achieve significant speedup with only slight reduction in quality. Unlike existing work based on simple greedy algorithms on the sensor

network graph, our algorithms are theoretically grounded and make use of the dependency information in the LD graph as well as the reductions of the EC graph, to achieve significant improvements in network lifetime.

CHAPTER 6.

A DISTRIBUTED ALGORITHMIC FRAMEWORK FOR COVERAGE PROBLEMS IN WIRELESS SENSOR NETWORKS

In this chapter, we generalize the concepts introduced in Chapter 3. 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 implies that a globally feasible solution can be reached, our framework can be used (See Section 6.1). In this chapter we use the solution in [55] 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. We also illustrate the use of the framework in a more general context by applying it to the problem of finding an independent set in a distributed environment. 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.

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:

- verifying the applicability of the framework,
- modeling the state space of local solutions and their interdependencies using a dependency graph structure,

- heuristically modeling the priority of local solutions based on the properties of the dependency graph structure, and
- determining the logistics of negotiating with neighbors to settle on mutually-compatible and high-priority local solutions.

6.1 Our General 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, since if all local targets are covered, this implies that globally, all targets are covered.

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.

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 6.2. Hence, in the discussion that follows, we describe the four steps 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. Compatibility may be checked by communicating with the neighbors. 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 [20]. 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 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*. 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 solutions 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 execution consisting of setup and negotiation phases. The setup phase is usually a round of information exchange with 1-hop 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. However, for effectiveness and scalability of the resulting distributed algorithm, the number of communication rounds with neighbors needs to be upper-bounded by a small constant.

An Example *Maximum Independent Sets:* To illustrate the use of this framework in a context outside that of the coverage problems for wireless sensor networks, we consider the problem of finding a valid independent set of vertices for a graph in a distributed manner. Given a graph $G = (V, E)$, an independent set of vertices V' is a subset $V' \subseteq V$ such that no two vertices in V' have an edge $e \in E$ that connects them. Let us now briefly develop the

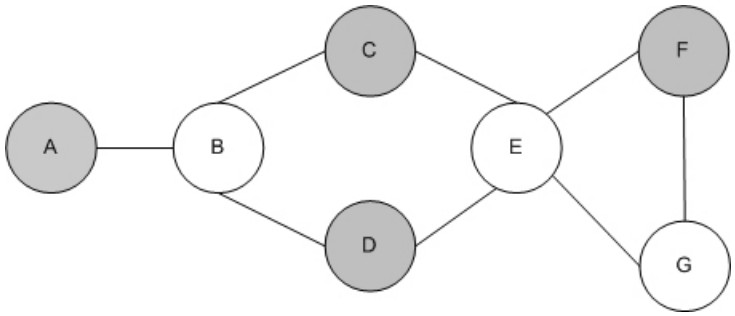

Figure 6.1. Example Graph for the Independent Set problem

proposed framework for this problem and look at an example. We will go into much greater depth in developing the framework for various coverage problems in the following section.

1. Applicability: This problem of finding an independent set can be solved within our framework since every node and its neighbors can decide their local independent sets which collectively yields the global independent set. There will never be any two nodes $x, y \in V$ that will both decide to be in the independent set if they have an edge $(x, y) \in E$, since they know of the existence of this edge.

2. Modeling the local solutions and their interdependencies: For the independent set problem, we define a local solution for a node to be given by a valid local independent set of all nodes in its closed one hop neighbor set. Hence, a node needs to construct all possible local independent sets to obtain all local solutions possible. For example, consider the graph shown in Figure 6.1. For the node B, the closed one hop neighbor set is given by $\{A, B, C, D\}$. All local solutions for this set are given by $\{A, C, D\}$ and $\{B\}$. For the node B, these two combinations are the only two possible local maximal independent sets. By maximal we mean that adding another vertex to the set would destroy the independence property. Similarly, for the node A, the set of local solutions is given by $\{A\}$ and $\{B\}$ (i.e., either A or B can be in the independent set) and so on.

3. Prioritize the local solutions: The objective of the prioritization phase is to have some criteria on the basis of which a node decides which local solution to use. For the independent set problem, one possible way to prioritize the local solutions may be given by ordering them in descending order of maximum cardinality. Hence, for the node B, $\{A, C, D\}$ would be the first choice local solution and $\{B\}$ would be the second. This is because the first would allow for a larger independent set. If cardinalities are equal, we can break ties by node id's.

4. Negotiate with neighbors for mutually satisfying solutions: A node must determine which of its solutions to use. Each solution is essentially a decision on which nodes in the one hop neighbor set are a part of the independent set. Since such a choice affects the solutions of other nodes in the neighborhood, a negotiation phase is required. For example, in the Figure 6.1, if the nodes $A, C,$ *and* D agree to the first choice of node B then this automatically renders the choice of $\{A\}$ invalid from the local solutions of node A, thereby making $\{B\}$ its local solution of choice. The nodes selected in the independent set are colored gray in the figure.

Let us now briefly sketch the implementation details of these phases. Initially every node broadcasts a message containing its *id* to discover who its neighbors are. Upon receiving this information, each node can then independently compute all local solutions and prioritize them. The negotiation phase would entail exchanging a message with its neighbors that describes the independent set it prefers to use. A neighbor can either accept or reject this proposed local partition. If a valid independent set exists, it has to be in the set of all local solutions and hence it will be satisfied. Else, none of the nodes will join the independent set. If the neighbors accept a node's choice, they have made a decision which now narrows their own choices. For the example in Figure 6.1, when the node B has negotiated its choice of $\{A, C, D\}$ with its neighbors and they have accepted this, they join the independent set. Node C can likewise negotiate with its neighbor E to select one of E's solutions in which E

is not in the independent set. In case E has chose itself in the independent set, C can drop itself without impacting anyone else.

6.2 Developing the Framework for Coverage Problems

In this section we define the steps of our framework presented in Section 6.1 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. As opposed to the independent set example in the previous section, we will look at the coverage problems in much greater detail in this section.

6.2.1 The Framework for Coverage Problems

6.2.1.1 Applicability of framework

For coverage problems, if every sensor ensures that its local coverage objective (local targets/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 sensors local solution does not invalidate that of another sensor's solution.

6.2.1.2 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 is usually small (where a cover could be for area or targets). This allows individual sensors to distributedly 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 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 defined the lifetime dependency graph in Chapter 3.

6.2.1.3 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 6.3. 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.

6.2.1.4 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.

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. 3.3 to decide whether it can switch off or if it needs to remain 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.

6.3 Applying the Coverage Framework

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

6.3.1 Target Coverage

In the target coverage problem, we are given a set of targets $T = \{t_1, t_2, ..., t_m\}$ 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.
- *Local Cover Set*: A local cover set for a sensor s is a minimal set of sensors in $N(s, 1)$ that cover all targets in $T(s)$. For better performance, all targets in $T(s'), s' \in N(s, 1)$ *or* $N(s, 2)$ can be considered.

Using the framework defined in Section 6.2, 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. 3.3. Each cover set C in the automata is a set of sensors that can cover the target set being considered.

Correctness: In [55] we present a proof to show that the proposed heuristic is correct, terminates and is deadlock free.

Message and Time Complexity: If the maximum degree Δ of the network graph is assumed to be a constant, the message complexity is given by $O(\Delta)$ and can also be considered to be constant. If τ is the maximum number of targets in any sensors neighborhood, it can be shown that the time complexity is given by $O(\Delta^{\tau})$. Even though this is exponential, since τ is usually small, the performance is not affected. See [55] for more details.

Self-organization and self-repair: The algorithm is self-organizing since each sensor s can run its own automaton independently to arrive at a decision to switch on/off. By exchanging messages, the sensor does however take into account its neighbors choices in making a decision. Also, because the distributed algorithm is organized in rounds, it is self-repairing. If a sensor fails, its neighbors can account for that in the following round as they will not receive a ON message for that sensor. This limits the loss of coverage to a maximum of one round.

6.3.2 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 minimal set of sensors in $s' \in N(s,1)$ that together cover this entire area.

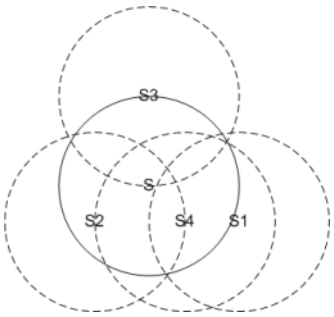

Figure 6.2. Example for area coverage

An example is shown in Figure 6.2. 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 [62] 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 in the target coverage problem.

Alternatively, [61] defines a field as a set of points that are covered by the same set of sensors. They then discretized 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 6.4. A different approach is taken in [5], where the authors use the idea of covering all the faces of a graph. This avoids having to define the granularity of a grid.

6.3.3 *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 $T = \{t_1, t_2, ..., t_m\}$ 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 δ is the minimum number of sensors covering any target $t \in T$. See [66] [40] [72] 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.

6.4 Simulation Results

In this section, we evaluate the performance of our coverage algorithms as compared with LBP [6] and DEEPS [7]. 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 6.3.2) 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 proportional to d. In the *non-linear* model, the energy needed to sense the same target is a function of d^p, where p varies typically from $2 - 6$. For our experiments, we fix the value of d to 2 (quadratic model).

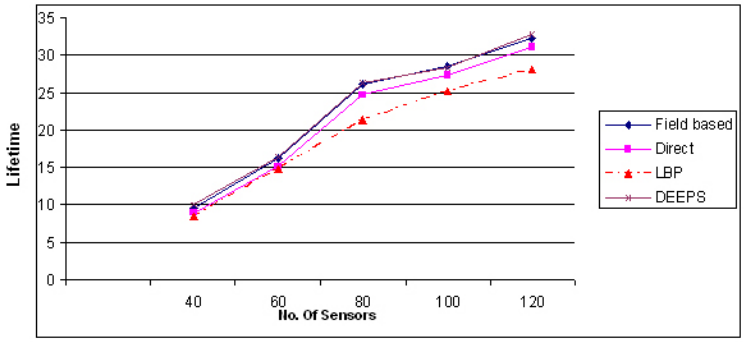

Figure 6.3. Area coverage with Number of Sensors varying, Linear Energy Model

Table 6.1. Lifetime of a 2-covered vs. 1-covered network with 25 targets

No. of Sensors	1-covered	2-covered
40	13.8	8.7
60	22.4	14.1
80	33.4	19.8
100	37.2	23.2
120	45.6	27.1

The results for target coverage were presented in Chapter 3.

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 6.3.2. The results are shown in Figure 6.3. 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 [62].

Finally, we implemented the k-coverage for $k = 2$, i.e., the two covered case where every target is covered by at least two sensors. Since both LBP and DEEPS have not be extended

to the k-coverage problem, we are unable to compare our performance relative to them. Instead, we compared the lifetime for the 2-covered case with the 1-covered case for our basic 1-hop algorithm. Using a network with 25 targets, we vary the number of sensors. The energy model we consider is linear. The results are summarized in Table 6.1. On an average we see a reduction of lifetime of 35-40% for the 2-covered case as compared to the 1-covered case.

CHAPTER 7.

ADJUSTABLE RANGE SENSING MODELS

In this chapter we present our work on centralized and distributed scheduling algorithms for sensor networks with a model where the sensors can adjust their sensing and communication radius. We begin by introducing the model in some detail and follow this up with our centralized algorithms. Finally, we present an adjustable range variation of LBP and DEEPS called ALBP and ADEEPS respectively.

7.1 The Adjustable Range Model

The adjustable range model was proposed independently by [65] and [70]. A significantly improved model was presented by us in [21]. In this subsection, we briefly survey the literature on extending the lifetime of WSNs using the adjustable range model.

In [65], the authors present two different coverage algorithms based on an adjustable model where the sensor can chose between one of two different ranges and one of three (maximum, medium and small) different ranges. Their objective was to minimize the overlapped area between sensor nodes, thereby resulting in energy savings. Their approach assumes that sensors can determine their own locations. In their simulation study they model a $50X50m^2$ area containing 1000 sensor nodes. One drawback of their paper is that they do not compare their heuristics to any other algorithms but compare their three adjustable models to each other. One of their model shows an energy saving of 20% but this comes at the cost of a 10% lapse in coverage area.

[70] also address the problem of selecting a minimum energy connected sensor cover when nodes have the ability to vary their sensing and transmission radius. The authors present a number of centralized and distributed heuristics for this problem including greedy formulations, Steiner Tree and Voronoi based approaches. The Voronoi based approach was shown to result in the best gains. Their model gives sensors the ability to vary both the

sensing and the communication range. The centralized algorithm (CGA) is shown to be with $O(logn)$ factor of the optimal, where n is the number of sensors in the network. The localized algorithm based on Voronoi diagrams is shown to be very close to CGA in performance. An extended version of this work was presented in [71].

In [11], the authors utilize a sensing model that allows a sensor to adjust its range from one of several different fixed values. The authors address the problem of finding the maximum number of sensor covers and they present a linear programming based formulation, a linear programming based heuristic and also greedy formulations for this problem. A more constrained case of the same problem is studied in [46]. The authors add the requirement of connectivity to the sensor covers and present distributed heuristics to maximize the total number of rounds. Their problem formulation attempts to maximize the number of set covers such that each set monitors all targets and every sensor in every set is assigned a range. They show that the problem is NP-complete by restriction and present an Integer Program for it. Since Integer Programming is NP-Hard, the relax and round their solution to derive a Linear Program for the problem. They also present a centralized greedy heuristic and a distributed greedy heuristic. The greedy criteria is the contribution of a sensor given as a ratio to the maximum possible contribution. An extended journal paper on this work was published in [14].

We present a different model for adjusting the range in [21]. Instead of allowing a sensor to adjust its range between a number of fixed options, we allow the nodes to vary their range smoothly between 0 and r_{max} where, r_{max} is the maximum value for the range. We give a mathematical model of this problem using a linear program with exponential number of variables and solve this linear program using the approximation algorithm of [32], to provide a $(1+\epsilon)(1+2lnn)$ approximation of the problem.

More recently, [63] uses the adjustable range model to give two localized range optimization schemes based on a one-hop approximation of the Delaunay Triangulation. They also

show that their approximation scheme achieves the same results as the original Delaunay Triangulation.

7.2 Network Model and Problem Statement

In this section we explain the sensor network model we use and give a formal definition of the lifetime problem with adjustable ranges.

7.2.1 Network Model

The sensor network model we use is very similar to that used in [10]. We consider a region R over which a set of static targets $T = \{t_1, t_2, ..., t_n\}$ are spread. A set of sensors $S = \{s_1, s_2, ..., s_m\}$ are randomly deployed over the region R. Since deployment is random, the number of sensors scattered is more than we need in the case of a determenistic deployment.

For each sensor, we also assume that it has the ability to adjust its sensing and transmission range. This model was introduced in [65, 70]. However, instead of assuming a discrete set of values for the range like [11], we assume a sensor can vary its range smoothly from 0 to some maximum value. Note that several commercially available sensors have the ability to do this.

7.2.2 Problem Statement

Sensor Network Lifetime Problem (SNLP) with range assignment

Given a monitored region R, a set of sensors $\{s_1, s_2, ..., s_m\}$ and a set of targets $\{t_1, t_2, ..., t_n\}$, and energy supply b_i for each sensor s_i, find a monitoring schedule (C_1,T_1),..., (C_k,T_k) *and* a range assignment for each sensor in a set C_i such that:

1. $\sum_{i=1}^{k} T_i$
2. each set cover monitors all targets t_1,...,t_n and,
3. each sensor s_i does not appear in the sets C_1,..C_k for a time more than b_i where, b_i is the initial energy of sensor s_i.

7.3 Centralized Algorithms

In this section we give a formal definition of the problem and the formulation of a linear program to solve it.

7.3.1 Linear Program Formulation

In this section we present a Linear Programming formulation for SNLP lifetime problem.

$$Maximize : \sum_{j=1}^{m} T_i \tag{7.1}$$

Subject to: $\sum_{j=1}^{m} C_{ij} T_i \leq b_i$

where,

b_i is the battery for sensor i

Rows i = 1,...,n represent each sensor

Columns j = 1,...,m represent each sensor cover, and

$$C_{ij} = \begin{cases} 0 & \text{if sensor i is not in the sensor cover j} \\ g(d) & \text{if sensor i is in sensor cover j with a range d} \end{cases}$$

g is a function of energy over distance.

This formulation for the LP is substantially different from the one in [10]. In their formulation the objective function attempts to maximize the number of set covers upto some limit K , whereas we maximize the actual network lifetime t. Also, it can be shown that having more than n covers C_j with non-zero t_j is of no use, where n is the order of sensors. Thus, if the goal is to maximize the network lifetime, then the objective function of the LP should reflect this. Also, our LP allows for sensors to have non-uniform battery life.

The linear program 7.1 is a packing LP that can be represented by the general form,

max $\{c^T | Ax \leq b, x \geq 0\}$

For the problem described above, the number of columns of the matrix A is exponential in the number of sensors and in order to overcome this, we use the Garg-Knemann algorithm [32] with an approximation ratio $(1+\epsilon)$. The algorithm assumes that the LP is implicitly given by a vector $b \in R^m$ and an algorithm that provides an f-approximation to find the column of A of minimizing length, where $length_y(j) = \sum_i A(i,j)y(i)/c(j)$ for any positive vector y. The algorithm is presented in Fig. 7.1.

Input: A vector $b \in R^m$, $\varepsilon > 0$, and an f-approximation algorithm F for the problem of finding the minimum length column $A_{q(y)}$ of a packing LP $\{\max\ c^T x | Ax \le b, x \ge 0\}$
Output: A set of columns $\{A^j\}_{j=1}^k$ each supplied with the value of the corresponding variable x^j, such that $(x^1,\ldots,x^k)$ correspond to all non-zero variables in a near-optimal feasible solution of the packing LP $\{\max\ c^T x | Ax \le b, x \ge 0\}$

(1) Initialize: $\delta = (1+\varepsilon)((1+\varepsilon)m)^{-1/\varepsilon}$, for $i = 1,\ldots,m$ $y(i) \leftarrow \frac{\delta}{b(i)}$, $D \leftarrow m\delta$, $j = 0$
(2) While $D < 1$
Find the column A_q using the f-approximation F.
Compute p, the index of the row with the minimum $\frac{b(i)}{A_q(i)}$
$j \leftarrow j+1$, $x^j \leftarrow \frac{b(p)}{A_q(p)}$, $A^j \leftarrow A_q$
For $i = 1,\ldots,m$, $y(i) \leftarrow y(i)\left(1+\varepsilon \frac{b(p)}{A_q(p)} / \frac{b(i)}{A_q(i)}\right)$, $D \leftarrow b^T y$.
(3) Output $\{(A^j, \frac{x^j}{\log_{1+\varepsilon}\frac{1+\varepsilon}{\delta}})\}_{j=1}^k$

Figure 7.1. The Garg-Konemann Algorithm [32]

Theorem: The Lifetime problem with adjustable sensing range assignment can be approximated within a factor of $(1+\epsilon)f$, for any $\epsilon > 0$ by using the Garg-Knemann Algorithm, where f is the approximation ratio of the algorithm that picks the minimum weight column.

This result is implied by the Garg-Konemann algorithm [32].

7.3.2 Minimum Weight Sensor Cover Problem with Adjustable Sensing Range

With an adjustable sensing range sensor network, the problem of generating covers becomes much more interesting since we can now generate more covers simply by varying the range of a sensor. In order to cover more targets we can increase the sensing range but this comes at the cost of increasing the energy consumed. So the question really is one of what is

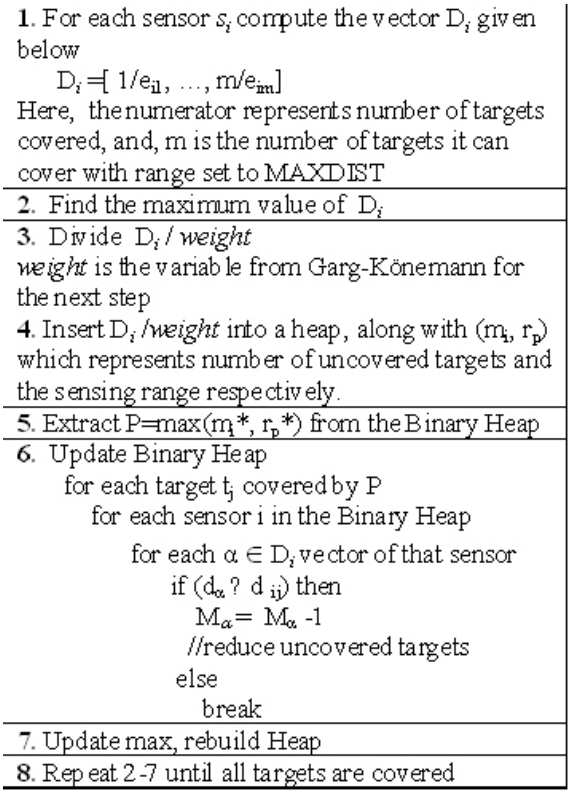

Figure 7.2. The Greedy Algorithm for the Minimum Weight Sensor Cover Problem with Adjustable Sensing Ranges

the best sensing range a sensor can pick to cover uncovered targets while taking into account increases in energy with distance.

Our f-approximation is a greedy heuristic that tries to add sensors to the set cover by picking a sensor s_i with a sensing range r_i that maximizes the following ratio:

$Gval(s_i) = No.\ of\ uncovered\ targets\ covered\ bys_i / \ weight\ xe_i$

Here, weight is the packing LP variable and is updated by Garg-Konemann. Also, e_i is a function of the distance d_{ij} between sensor s_i and target t_j and can be varied to study linear, quadratic and other energy models.

The algorithm is outlined in Fig. 7.2. We assume that there exists a cutoff distance MAXDIST beyond which no sensor can increase its distance.

Theorem: The Greedy Algorithm for the Minimum Weight Sensor Cover Problem with Adjustable Sensing Ranges has an approximation ratio $(1 + lnk)$.

This is from the standard greedy algorithm for the Minimum Weight Set Cover Problem with k points to cover.

COROLLARY. The Lifetime problem with adjustable sensing range assignment can be approximated within a factor of $(1 + \epsilon)(1 + lnm)$ for any $\epsilon > 0$ by using the Algorithm of Fig. 7.1. This result comes from the previous two theorems with $k = O(m)$ elements to cover, m being the number of targets.

7.3.3 Experimental Evaluation

In this section, we evaluate the performance of our heuristic and compare it to that proposed in [10]. For simulation purposes we use a static network of sensors scattered in a 100m x 100m area. The adjustable parameters are:

- N the number of sensor nodes. We vary this from 80 to 200.
- M the number of targets. Initial results are for 25 targets and 50 targets.
- The sensing range r which can vary smoothly from 5m to 60m.

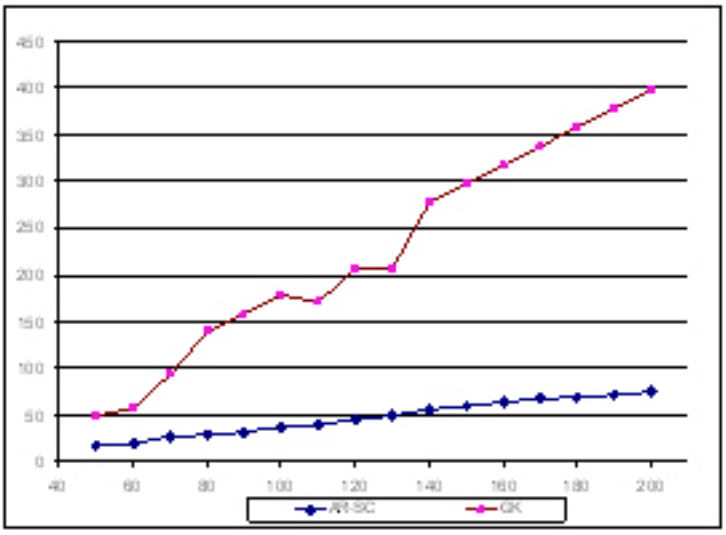

Figure 7.3. Variation in Network Lifetime with Number of Sensors. Number of Targets=25, Energy model is linear. AR-SC denotes the algorithm in [10]

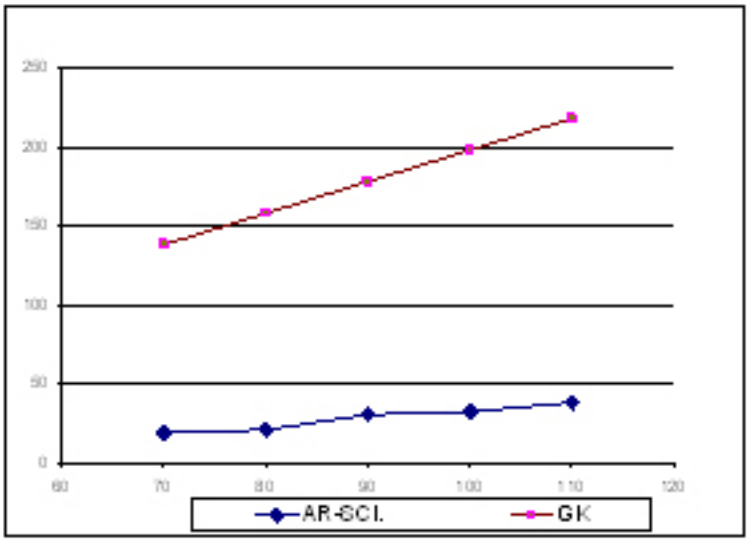

Figure 7.4. Variation in Network Lifetime with Number of Sensors. Number of Targets=50, Energy model is linear

In order to compare our results with [10], we use the linear and quadratic energy models defined by them. The linear model specifies the energy e_p needed to cover a target at distance r_p as $e_p = c_1 * r_p$ where, c_1 is a constant. For the quadratic model, $e_p = c_2 * r_p^2$ where, c_2 is a constant. The range variation is the same as that for the discrete model except for that fact that we allow it to vary smoothly. For comparison we run simulations against their distributed version.

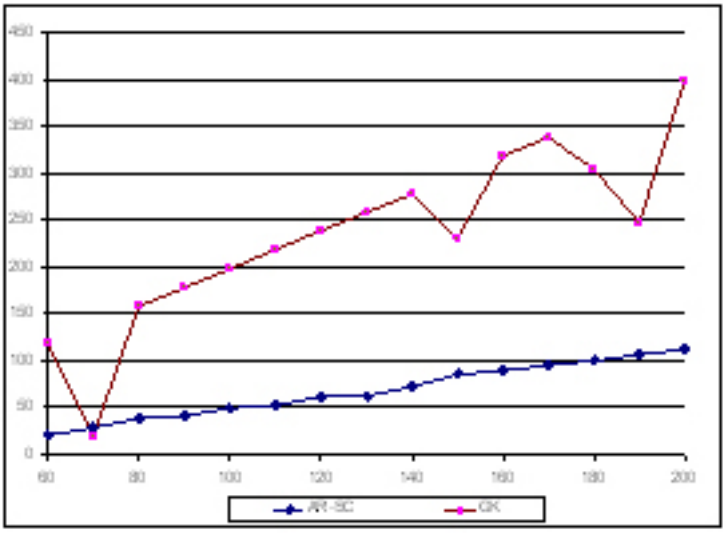

Figure 7.5. Variation in Network Lifetime with Number of Sensors. Number of Targets=25, Energy model is quadratic

We have implemented our algorithm in C++ and run experiments on randomly generated test cases. The ϵ value for the quality of the Garg- Konemann algorithm is set to 0.1. After finding sensor covers from Garg-Konemann we find the optimal schedule by assigning the best times for each cover by using CPLEX. Our results are shown below.

Figure 7.3 and Figure 7.4 illustrate the case with 25 and 50 targets respectively with a linear energy model. We measure the variation in Network Lifetime with an increase in the number of sensors. Figure 7.5 repeats the same experiment with 25 targets and a quadratic energy model. As can be seen from the graphs, we have about an order of 4 times improvement in network lifetime. The drastic performance improvement can be explained by the following reasons:

- Smoothly varying sensing range - The adjustable range model used in required that the sensing range be increased in 'P' discrete steps. We allow a smooth variation up to a cutoff distance MAXDIST. This has the advantage that the sensor spends only the energy required to reach the target and no more.

- Fractional time assignment to each sensor cover - Due to our LP formulation, we can exploit the ability to assign fractional times to each sensor cover instead of assigning time in fixed intervals.

- Provably good algorithm - As shown in Section 5 the heuristic has a provably good approximation ratio of $(1 + lnm)$.

7.4 Distributed Algorithms using Adjustable Range

7.4.1 Adjustable Range Load Balancing Protocol (ALBP)

In this section, we present a distributed load balancing protocol for sensors with an adjustable range called ALBP. This protocol extends the ideas of LBP [6] to the adjustable range model. As with LBP, the objective of the protocol is to maximize the time for which all targets in the network are covered. The intuition behind the protocol is also similar to LBP, the aim is to keep as many sensors alive by balancing their load so as to let them exhaust their batteries simultaneously. ALBP, however, differs from LBP in that while making a decision on switching a sensor to an active state, it also needs to decide what range this sensor should have.

We begin by defining the different states a sensor can be in at any point of time:

- **Active**: The sensor is monitoring a target('s).

- **Idle**: The sensor is listening to other neighboring sensors, but does not monitor targets.

- **Deciding**: The sensor is presently monitoring a target, but will change its state to either active or idle soon.

Setup: The setup phase starts as before. At the beginning of a round, each sensor exchanges information on its battery level and the set of targets it covers with its neighbors. However, after broadcasting this information, a sensor enters the *deciding* state with its *maximum* range.

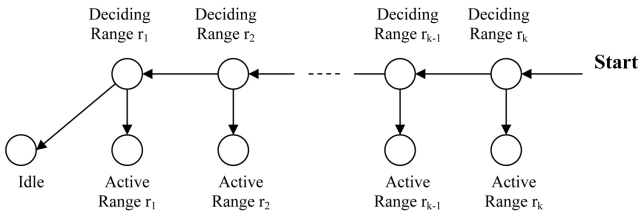

Figure 7.6. State transitions for the ALBP protocol

Transitions: Figure 7.6 shows the state transitions for ALBP. At the end of the setup phase, each sensor is in the deciding state with its maximum range. A sensor then changes its state according to the following transition rules:

- **Active state with a range** r: A sensor transitions to the Active State with a range r, if there is a target at range r which is not covered by any other active or deciding sensors.
- **Deciding state with lower range**: A sensor in the deciding state with some range r can decrease its range to the next closest target if all its targets at range r are covered by another sensor in the active state or by a sensor that is in the deciding state and has a higher battery life.
- **Idle state**: A sensor is in the idle state if it has reduced its range to zero (i.e., all its targets are covered by active sensors or higher energy sensors in the deciding state)

  Every sensor uses these rules to make a decision on whether to enter the active or idle state. The sensors will stay in this state until the end of a round, upon which the

process will repeat itself. When the network reaches a point where a target cannot be covered by any sensor, the network is considered dead.

Correctness: In ALBP, a sensor can enter the *idle* state only when its range reaches zero. To achieve this, all its targets had to have been covered by an active or deciding sensor that had a higher energy than it. Hence, all targets are always covered.

7.4.1.0.1 Time and Message Complexity: The time complexity of ALBP is $O(\Delta^2)$ and the message complexity is $O(\Delta)$ where, Δ is the maximum degree of the sensor graph.

At the start of a round, each sensor receives from every neighbor a message containing the targets that neighbor covers, and its battery life. If Δ is the maximum degree of the graph, a sensor can have no more than Δ neighbors. This means that a sensor can receive no more than Δ messages, which it can process in $O(\Delta)$ time.

In the worst case, a sensor may have to wait for all its neighbors to decide their state before it can make a decision. Thus, the waiting time accumulates as $O(\Delta^2)$, hence the time complexity.

Since each sensor has at most Δ neighbors and during a round a sensor sends at most two messages to its neighbors (its battery and targets covered information, and its status - on/off), at most $O(\Delta)$ messages are sent in the setup phase. Hence, the message complexity is $O(\Delta)$.

7.4.2 Adjustable Range Deterministic Energy Efficient Protocol (ADEEPS)

In this subsection, we present an adjustable range version of the DEEPS protocol [7]. Each sensor can once again be in one of three states - *active*, *deciding* or *idle*. The definition of these states remains as before.

We make use of similar concepts of *sink* and *hill* targets as DEEPS. However, instead of defining these using the total battery of the sensors covering a target (as DEEPS does), we

define them with respect to the maximum lifetime of the target. Let the lifetime of a sensor with battery b, range r and using an energy model e be denoted as $Lt(b, r, e)$. Then, the maximum lifetime of a target would be $Lt(b_1, r_1, e) + Lt(b_2, r_2, e) + Lt(b_3, r_3, e_3) + ...$ assuming that it can be covered by some sensor with battery b_i at distance r_i for $i = 1, 2, ...$

Now, we can define the sink and hill targets as follows. A target t is a *sink* if it is the smallest maximum lifetime target for at least one sensor covering t. A *hill* is a target which is not a sink for any of the sensors covering it. We define the in-charge sensors for a target t as follows:

- If the target t is a sink, then the sensor s covering t with the highest lifetime $Lt(b, r, e)$ for which t is the poorest is placed in-charge of t.
- If target t is a hill then out of the sensors covering t, the sensor s whose poorest target has the highest lifetime is placed in-charge of t. If there are several such sensors, then the richest among them is placed in-charge of t.

Setup: Each sensor initially broadcast its lifetime and covered targets to all neighbors of neighbors. This is similar to [7]. After this, it stays in the deciding state with its maximum range.

Transitions: A sensor that is in the deciding state with range r changes its state according to the following rules:

- **Active state with a range** r: If there is a target at range r which is not covered by any other active or deciding sensors, the sensor enters the Active state with range r.
- **Deciding state with lower range**: A sensor in the deciding state with some range r can decrease its range to the next closest in-charge target if all its in-charge targets at range r are covered by another sensor in the active state or by a sensor that is in the deciding state and has a higher battery life.
- **Idle state**: When a sensor s is not in-charge of any target except those already covered by on-sensors, s switches itself to the idle state.

The algorithm again operates in rounds. When there exists a target that cannot be covered by any sensor, the network is considered to be dead.

Correctness: The correctness of ADEEPS can be proved from the fact that each target has a sensor which is in-charge of that target and the transition rule to active state assures that the resultant sensor cover is minimal in which each sensor s has a target covered only by s.

Time and Message Complexity: In each round, the time complexity of ADEEPS is $O(\Delta^2)$ and the message complexity is $O(\Delta^2)$ where, Δ is the maximum degree of the graph.

Each sensor has no more than Δ neighbors. At the start of each round, every sensor receives from its neighbors and their neighbors (2-hops) information about their lifetime and the targets covered. Thus a sensor can receive at most Δ^2 messages. Once this information has been received, all decisions on sink/hill targets, in-charge sensor and the active/idle state can be made locally. Thus the time complexity is $O(\Delta^2)$.

Also, since a sensor has at most Δ neighbors and it needs to communicate the setup information to two-hops, each sensor sends $O(\Delta^2)$ messages in the setup phase. This means that the message complexity is $O(\Delta^2)$.

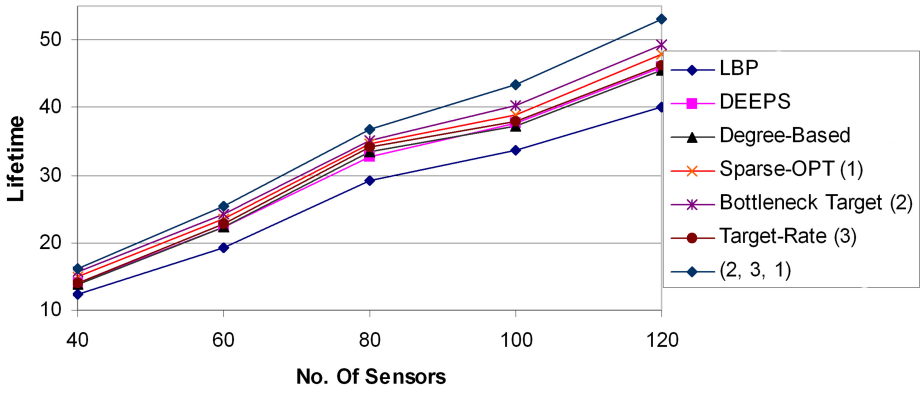

Figure 7.7. Variation in network lifetime with the number of sensors with 25 targets, linear energy model, 30m range

7.5 Simulations

To evaluate the performance of the new algorithms and to make comparison with the algorithms in [6, 7], the new algorithms are implemented by using C++. We built on the source code for [6, 7]. For the simulation environments, a static wireless network of sensors and targets which are scattered randomly, while ensuring that all targets can be covered, in $100m \times 100m$ area is considered. The location of the sensor nodes can be randomly generated and the targets can also be placed randomly.

We assume that the communication range of each sensor is two times the sensing range. Simulations are carried out by varying the number of sensors and the lifetime is measured. We also vary the maximum range, energy models, and numbers of targets with various combinations. For these simulations, we use the linear energy model wherein the power required to sense a target at distance d is proportional to d. We also experiment with the quadratic energy model (power proportional to d^2). Note that to facilitate comparison, we follow the simulation setup of [6, 7].

In the first simulation shown in Figure 7.7, we limit the maximum range to $30m$. This means that a sensor can smoothly vary its range from 0 to $30m$. The simulation is conducted with 25 randomly deployed targets, 40 to 200 sensors with an increment of 20 and a linear energy model. As is expected, increasing the number of sensors while keeping the number of targets fixed causes the lifetime to increase for all the protocols. Also, using the adjustable range model shows performance improvements when compared to the fixed range model. As can be seen from the figure, ALBP outperforms LBP by at least 10% and ADEEPS outperforms DEEPS by around 20%.

In the second simulation shown in Figure 7.8, we study the network lifetime while increasing the number of targets to 50 and keeping maximum range at $30m$. The numbers of sensors are varied from 40 to 200 with an increment of 20 and the energy model is linear. The results of simulations are consistent and showed that the network lifetime increases with the number of sensors. When compared with the results of Figure 7.7, the network lifetime

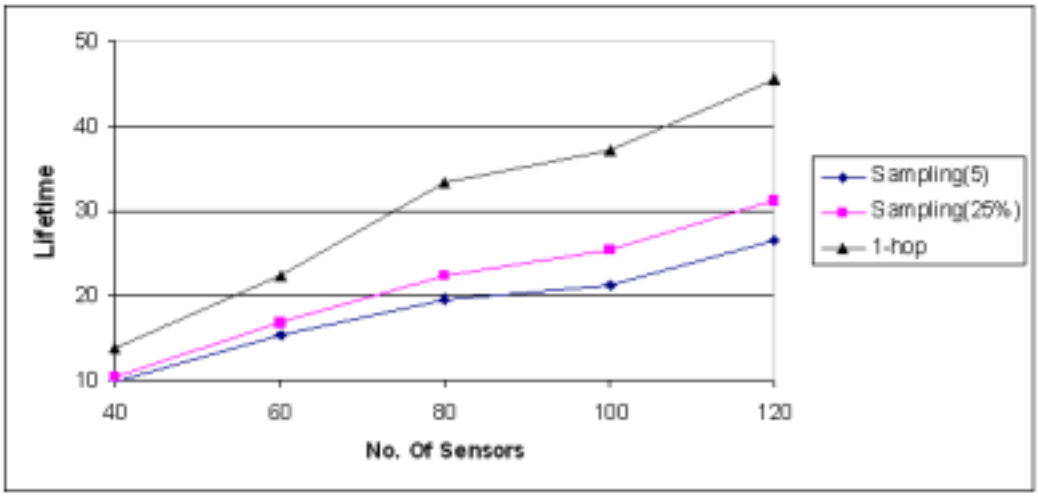

Figure 7.8. Variation in network lifetime with the number of sensors with 50 targets, linear energy model, 60m range

decreases as more targets are monitored. This is also a logical conclusion, since a larger number of targets implies that there is more work to be done by the network as a whole.

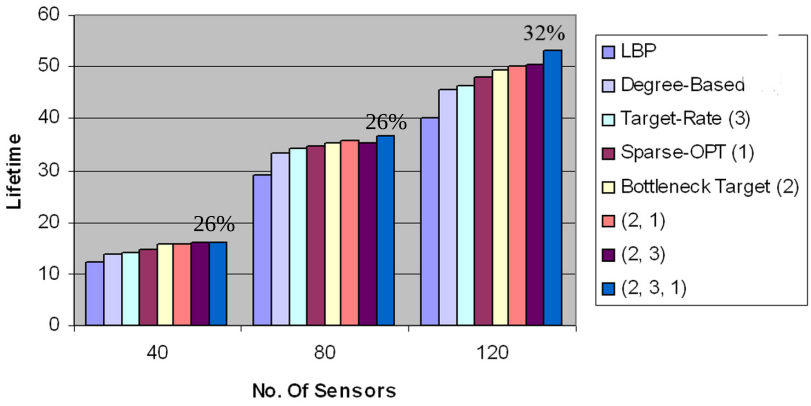

Figure 7.9. Variation in network lifetime with the number of sensors, with 25 targets, quadratic energy model and 30m maximum range

In Figure 7.9, we change the energy model to the quadratic model. We use the same number of sensors (40 to 200 with increment of 20), the maximum range is $30m$ and the energy model is quadratic. As in Figure 7.7, for both energy models, the result indicates that the network lifetime increases with the number of sensors. As is expected, the quadratic model

causes all protocols to reduce in total lifetime when compared to the linear model of Figure 7.7. It can also be seen that the network lifetime is significantly improved with ALBP and ADEEPS in the quadratic model. This phenomenon is quite logical since in the fixed sensing model, each sensor consumes more energy than the adjustable range model. Improvements here are in the range of 35-40% when compared to their fixed range counterparts.

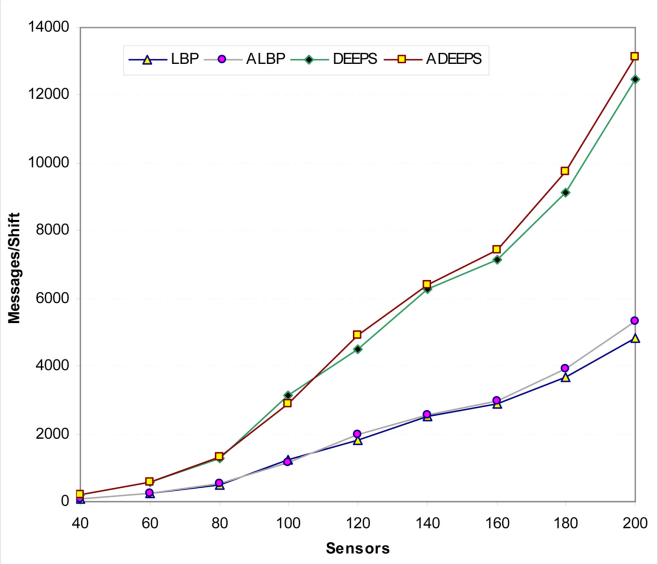

Figure 7.10. Average numbers of messages sent during each round

Finally, in Figure 7.10, we plot the average numbers of messages sent during each round. It can be seen that more messages are sent when the number of deployed sensors increases and the average messages sent in DEEPS and ADEEPS are much higher than LBP and ALBP. This is because in DEEPS and ADEEPS the communication range is four times higher than the sensing range and each sensor has more neighbors and needs to send more messages (in effect communicating with 2-hop neighbors).

From the results, the overall improvement in network lifetime of ALBP over LBP is around 10% and ADEEPS over DEEPS is about 20% for linear energy model. For quadratic energy model, the improvements are even higher.

The design process to transform LBP to ALBP and DEEPS to ADEEPS was fairly simple. This shows that the basic algorithms are easily extended to the adjustable range model with minimal effort. As part of our future work, we are examining the extension of the lifetime dependency graph model based heuristics to the adjustable range model. Accounting for the ability to adjust ranges is not trivial in the dependency graph and some thought is needed to determine how to assign ranges in that model to even compute cover sets for the LD graph.

CHAPTER 8.
CONCLUSION AND FUTURE WORK

Despite a lot of research effort, creating real-world deployable sensor networks remains a difficult task. A key bottleneck is the limited battery life of sensor motes. Hence, energy conservation at every layer of the network stack is critical. Creating realistic theoretical models for problems in this domain that take this into account remains a challenge. Our work addresses energy efficiency at only point in the network stack. However, a holistic approach to energy efficiency design should not only account for energy concerns in each layer of the network stack for problems like routing, medium access etc., but also consider cross-layer issues and interactions.

In this dissertation, we present innovative models and heuristics to address the coverage problem in Wireless Sensor Networks. Our work points to the potential of lifetime dependency graphs while serving to highlight the shortcomings of using standard distributed algorithms to this problem. In order to successfully bridge the gap between the theory and practice of wireless sensor networks, there is a clear need for algorithms that are designed keeping the unique constraints of these networks in mind. The improvements in network lifetime obtained by our approach using the dependency graph and heuristics that stem serve to underscore this point. The fact that this work can be extended to other graph and network problems shows the broader applicability of the underlying theory.

It is my goal to expand my work on energy efficiency in WSNs to examine related problems like routing and clustering. Also, developing application specific energy aware techniques is a key challenge specially in the area of environmental sensing applications.

Future Work

As part of our future work, we intend to extend the dependency graph and the equivalence class graph ideas to heterogeneous wireless sensor networks, work on tighter bounds and apply our results to related problems.

- *Extend Simulation Studies:* As part of the future work related to this dissertation, we intend to implement our algorithms in a more detailed network simulator environment like ns-2 [2]. This will allow us to study the impact of these algorithms across several network parameters. In particular, it will allows us to better study the message complexity and measure the energy spent on control messages for implementing the heuristics.

- *Adjustable Sensing Range:* Recent work in [10, 21] have introduced the model of sensors with adjustable sensing ranges varying between zero (switched off) and a maximum. The adjustable range problem is more complex since even locally it further increases the space of possible covers. This also relates to how we represent the adjustable ranges. [10] models each sensor as having k discrete ranges. In [21], we modeled sensors as having the ability to smoothly vary its range. This closely represents a real sensor and since a sensor s effectively has only a maximum of $|T(s)|$ number of ranges, the set of targets covered by sensor s, the smooth model results in further improvements in the lifetime.

  Our LD graph could be adapted to now construct all covers with each of the $|T(s)|$ ranges per sensor, but this will be very inefficient even for small neighborhoods. Hence we need to look beyond this and use some of the techniques that we propose to handle the exponential space in Chapter 5. As discussed in that section, we can split the space into n classes where covers are grouped by sharing the same bottleneck. This definition remains valid in the adjustable model also but now the different covers in the equivalent class of a sensor s can have s being used with different ranges. Thus, each class $[s]$ now has subclasses corresponding to each sensing range. However, when a cover is being burnt from class $[s]$, all the covers of that class are also burnt. This would indicate that s has a specific range in OPT at any time. It is clear that for sensors covering the bottleneck targets the smallest range should be used. For other sensors, some form of sampling across the different ranges can be used to represent each

subclass in the LD graph model. Thus, experimental techniques for the dependency graph can be extended to these networks.

- *Non-Identical Sensing Shape:* Studying the impact of non-identical sensing shapes on coverage algorithms can be done using two different scenarios. The first is one where the communication links are not bidirectional. This implies that the sensor communication graph, SN, is a directed graph. The second scenario is that the sensing range of a sensor is not represented by a fixed shape (like circular).

  Both these scenarios are common in real world applications and some models in the literature have been adapted to handle these [42, 41]. Our framework is robust enough to handle both of these conditions without much change. For the first condition, consider two sensors s_1 and s_2. Directional links implies that one of these sensors, say s_2, can receive messages from s_1 but not vice versa. This in turn means that s_2 has a richer information set to use in its decision making process - since it probably has better/more cover sets. But other than that, the different phases of our proposed approach are not impacted in any way. An interesting point to investigate here would be the degradation of performance due to directional links of our proposed approach when compared to other algorithms that can also operate with directional links.

  The second scenario of uneven sensing shape is handled easily since our algorithms make no assumptions of the shape. The only information a sensor s requires is knowledge of what targets $T(s)$ it can cover. Therefore, shape is not required to be circular.

- *Using an Overlay Network:* Another common view of a heterogeneous network in the literature is one where there are a few high/unlimited energy nodes that form a fast communication backbone for the network along with normal sensor nodes. An example of such nodes is the Intel Xscale [1] 802.11 nodes. This has the effect of folding the network since two sensors that may be many hops away from each other now appear to be much closer due to the super nodes. Experiments carried out at Intel have shown

that even with a few plugged-in XScale nodes, the lifetime of the network increases by about 20%.

In our model, the overlay network can be used in several different ways. Since the lifetime of the network is bounded by the global bottleneck target, this can be identified and this information shared using the overlay, thereby allowing a better selection of covers. Another instance where this is useful is in determining the duration of a round. Instead of using predetermined constant duration rounds, the overlay can propagate the battery level of the global bottleneck sensor and this can be used as the duration of a round for that particular iteration. This would reduce the number of rounds and thereby the costs associated with each reshuffle.

- *Approximation Ratios:* While the range of heuristics we have proposed have been extensively simulated and evaluated against comparable work in the literature, we have not investigated the approximation ratios for our algorithms. As part of the future theoretical work tied to this dissertation, we expect to establish approximation ratios for these heuristics. This also ties in to our next goal of providing tighter upper bounds for the maximum lifetime scheduling problem.

- *Upper Bounds:* In the literature, the upper bound on the network lifetime is computed by calculating the amount of time the weakest (least-covered) target can be covered [39]. This is given by the sum of the batteries of all sensors covering this target - known as the bottleneck target. This bound is simplistic and loose because it assumes that each sensor covering this target can be burnt in a mutually exclusive manner.

  The lifetime dependency graph holds the potential to give us a tighter upper bound on the lifetime. At the very least, the edges in the dependency graph tell us whether a sensor is being burnt exclusive of other sensors. For the two-node case, a tighter upperbound is provided by the adjoining edge in the dependency graph [55]. It is important to extend these bounds and investigate how the sub-clique that exists among

nodes in the same class can be burnt. This can then provide some clues on how to better bound the lifetime of the network. Another possibility is to consider the sensors covering the bottleneck target. There exists an equivalence class corresponding to each such sensor. The EC graph induced by these classes may result in tighter upperbounds than the trivial.

BIBLIOGRAPHY

[1] Intel heterogeneous sensors, http://www.intel.com/research/exploratory/heterogeneous.htm.

[2] The network simulator - ns-2, http://www.isi.edu/nsnam/ns/.

[3] Z. Abrams, A. Goel, and S. Plotkin. Set k-cover algorithms for energy efficient monitoring in wireless sensor networks. *Third International Symposium on Information Processing in Sensor Networks*, pages 424–432, 2004.

[4] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Commun. Mag.*, pages 102–114, 2002.

[5] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky. Power efficient monitoring management in sensor networks. *Wireless Communications and Networking Conference (WCNC)*, 4:2329–2334 Vol.4, 2004.

[6] 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*, 2005.

[7] Dumitru Brinza and Alexander Zelikovsky. Deeps: Deterministic energy-efficient protocol for sensor networks. *Proceedings of the International Workshop on Self-Assembling Wireless Networks (SAWN)*, pages 261–266, 2006.

[8] M. Cardei, M.T. Thai, Yingshu Li, and Weili Wu. Energy-efficient target coverage in wireless sensor networks. *INFOCOM 2005*, 3, March 2005.

[9] M. Cardei and J. Wu. Energy-efficient coverage problems in wireless ad hoc sensor networks. *Computer Communications*, 29(4):413–420, 2006.

[10] M. Cardei, Jie Wu, Mingming Lu, and M.O. Pervaiz. Maximum network lifetime in wireless sensor networks with adjustable sensing ranges. *IEEE International Conference on Wireless And Mobile Computing, Networking And Communications (WiMob)*, 3:438–445 Vol. 3, 2005.

[11] M. Cardei, Jie Wu, Mingming Lu, and M.O. Pervaiz. Maximum network lifetime in wireless sensor networks with adjustable sensing ranges. *Wireless And Mobile Computing, Networking And Communications, 2005. (WiMob'2005), IEEE International Conference on*, 3:438–445 Vol. 3, Aug. 2005.

[12] Mihaela Cardei and Ding-Zhu Du. Improving wireless sensor network lifetime through power aware organization. *Wireless Networks*, 11:333–340(8), 2005.

[13] Mihaela Cardei, David MacCallum, Maggie Xiaoyan Cheng, Manki Min, Xiaohua Jia, Deying Li, and Ding-Zhu Du. Wireless sensor networks with energy efficient organization. *Journal of Interconnection Networks*, 3(3-4):213–229, 2002.

[14] Mihaela Cardei, Jie Wu, and Mingming Lu. Improving network lifetime using sensors with adjustable sensing ranges. *Int. J. Sensor Networks*, 1(1/2):41–49, 2006.

[15] Jean Carle and David Simplot-Ryl. Energy-efficient area monitoring for sensor networks. *Computer*, 37(2):40–46, 2004.

[16] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *Proceedings of the 7th ACM International Conference on Mobile Computing and Networking*, pages 85–96, Rome, Italy, July 2001.

[17] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *ACM Wireless Networks*, 8(5), September 2002.

[18] Chee-Yee Chong and S.P. Kumar. Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, Aug. 2003.

[19] D. Culler and W. Hong. Wireless sensor networks. *Special Issue, CACM*, 2004.

[20] S.K Das, N. Deo, and S. K. Prasad. Two minimum spanning forest algorithms for fixed-size hypercube computers. *Parallel Computing*, 15:179–187, 1990.

[21] A. Dhawan, C. T. Vu, A. Zelikovsky, Y. Li, and S. K. Prasad. Maximum lifetime of sensor networks with adjustable sensing range. *Proceedings of the International Workshop on Self-Assembling Wireless Networks (SAWN)*, pages 285–289, 2006.

[22] Akshaye Dhawan. On distributed algorithms for lifetime of wireless sensor networks. In *HiPC 2008 Student Symposium.*

[23] Akshaye Dhawan and Sushil K. Prasad. A distributed algorithmic framework for coverage problems in wireless sensor networks. *Procs. Intl. Parallel and Dist. Processing Symp. Workshops (IPDPS), Workshop on Advances in Parallel and Distributed Computational Models (APDCM)*, pages 1–8, 2008.

[24] Akshaye Dhawan and Sushil K. Prasad. Energy efficient distributed algorithms for sensor target coverage based on properties of an optimal schedule. In *HiPC: 15th International Conference on High Performance Computing, LNCS 5374*, 2008.

[25] Akshaye Dhawan and Sushil K. Prasad. A distributed algorithmic framework for coverage problems in wireless sensor networks. *To Appear in International Journal of Parallel, Emergent and Distributed Systems(IJPEDS)*, 2009.

[26] S. S. Dhillon and K. Chakrabarty. Sensor placement for effective coverage and surveillance in distributed sensor networks. volume 3, pages 1609–1614 vol.3, 2003.

[27] Benyuan Liu Don. On the coverage and detectability of large-scale wireless sensor networks, 2003.

[28] Laura Marie Feeney. An energy consumption model for performance analysis of routing protocols for mobile ad hoc networks. *Mob. Netw. Appl.*, 6(3):239–249, 2001.

[29] L.M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1548–1557 vol.3, 2001.

[30] Douglas W. Gage. Sensor abstractions to support many-robot systems. In *Proceedings of SPIE Mobile Robots VII*, pages 235–246, 1992.

[31] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[32] Naveen Garg and Jochen Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, page 300, Washington, DC, USA, 1998. IEEE Computer Society.

[33] Arvind Giridhar and P.R. Kumar. Maximizing the functional lifetime of sensor networks. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 5–12, April 2005.

[34] W.W. Gregg, W.E. Esaias, G.C. Feldman, R. Frouin, S.B. Hooker, C.R. McClain, and R.H. Woodward. Coverage opportunities for global ocean color in a multimission era. *Geoscience and Remote Sensing, IEEE Transactions on*, 36(5):1620–1627, Sep 1998.

[35] S. Iyengar and R. Brooks. *Handbook of Distributed Sensor Networks*. Chapman and Hall/CRC, 2005.

[36] S. S. Iyengar and R. Brooks. Computing and communications in distributed sensor networks. *Special Issue, Jr. of Parallel and Distributed Computing*, 64(7), 2004.

[37] Eun-Sun Jung and Nitin H. Vaidya. Power aware routing using power control in ad hoc networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(3):7–18, 2005.

[38] K. Kar and S. Banerjee. Node placement for connected coverage in sensor networks. In *In Proc. of WiOpt 2003: Modeling and Optimization in Mobile, Ad-Hoc and Wirless Networks*, 2003.

[39] Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 25–32, 2003.

[40] Santosh Kumar, Ten H. Lai, and József 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.

[41] L. Lazos and R. Poovendran. Coverage in heterogeneous sensor networks. *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2006 4th International Symposium on*, pages 1–10, 03-06 April 2006.

[42] Loukas Lazos and Radha Poovendran. Stochastic coverage in heterogeneous sensor networks. *ACM Trans. Sen. Netw.*, 2(3):325–358, 2006.

[43] Xiang-Yang Li, Peng-Jun Wan, and Ophir Frieder. Coverage in wireless ad hoc sensor networks. *IEEE Transactions on Computers*, 52(6):753–763, 2003.

[44] Yingshu Li, My T. Thai, and Weili Wu. *Wireless Sensor Networks and Applications*. Springer, 2008.

[45] Jun Lu and T. Suda. Coverage-aware self-scheduling in sensor networks. *18th Annual Workshop on Computer Communications (CCW)*, pages 117–123, 2003.

[46] Mingming Lu, Jie Wu, Mihaela Cardei, and Minglu Li. Energy-efficient connected coverage of discrete targets in wireless sensor networks. *ICCNMC 2005, LNCS 3619*, pages 43–52, 2005.

[47] Yingchi Mao, Zhijian Wang, and Yi Liang. Energy aware partial coverage protocol in wireless sensor networks. *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, pages 2535–2538, 21-25 Sept. 2007.

[48] Mauricio Marengoni, Bruce A. Draper, Allen Hanson, and Ramesh Sitaraman. A system to place observers on a polyhedral terrain in polynomial time. pages 773–780, 2000.

[49] Seapahn Megerian, Farinaz Koushanfar, and Miodrag Potkonjak. Worst and best-case coverage in sensor networks. *IEEE Transactions on Mobile Computing*, 4(1):84–92, 2005. Senior Member-Srivastava, Mani B.

[50] Seapahn Meguerdichian, Farinaz Koushanfar, Miodrag Potkonjak, and Mani B. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *in IEEE INFOCOM*, pages 1380–1387, 2001.

[51] Seapahn Meguerdichian, Farinaz Koushanfar, Gang Qu, and Miodrag Potkonjak. Exposure in wireless ad-hoc sensor networks. In *Procs. of 7th Annual International Conference on Mobile Computing and Networking (MobiCom'01)*, pages 139–150, 2001.

[52] Seapahn Meguerdichian and Miodrag Potkonjak. Low power 0/1 coverage and scheduling techniques in sensor networks. *UCLA Technical Reports 030001*, 2003.

[53] Joseph O'Rourke. *Art gallery theorems and algorithms*. Oxford University Press, Inc., New York, NY, USA, 1987.

[54] Maulin Patel, R. Chandrasekaran, and S. Venkatesan. Energy efficient sensor, relay and base station placements for coverage, connectivity and routing. In *Performance, Com-*

puting, and Communications Conference, 2005. IPCCC 2005. 24th IEEE International, pages 581–586, April 2005.

[55] Sushil K. Prasad and Akshaye Dhawan. Distributed algorithms for lifetime of wireless sensor networks based on dependencies among cover sets. In *HiPC: 14th International Conference on High Performance Computing, LNCS 4873*, pages 381–392, 2007.

[56] Vijay Raghunathan, Curt Schurgers, Sung Park, Mani Srivastava, and Barclay Shaw. Energy-aware wireless microsensor networks. In *IEEE Signal Processing Magazine*, pages 40–50, 2002.

[57] K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004.

[58] S. Sahni and X. Xu. Algorithms for wireless sensor networks. *Intl. Jr. on Distr. Sensor Networks*, 1, 2004.

[59] S. Schmid and R. Wattenhofer. Algorithmic models for sensor networks. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 11 pp.–, April 2006.

[60] Stefan Schmid and Roger Wattenhoffer. Maximizing the lifetime of dominating sets.

[61] S. Slijepcevic and M. Potkonjak. Power efficient organization of wireless sensor networks. *IEEE International Conference on Communications (ICC)*, pages 472–476 vol.2, 2001.

[62] Di Tian and Nicolas D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *WSNA: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 32–41, New York, NY, USA, 2002. ACM.

[63] Jiong Wang and S. Medidi. Energy efficient coverage with variable sensing radii in wireless sensor networks. *Wireless and Mobile Computing, Networking and Communi-*

cations, 2007. WiMOB 2007. Third IEEE International Conference on, pages 61–69, Oct. 2007.

[64] Roger Wattenhoffer. Sensor networks: Distributed algorithms reloaded - or revolution?

[65] Jie Wu and Shuhui Yang. Coverage issue in sensor networks with adjustable ranges. *Parallel Processing Workshops, 2004. ICPP 2004 Workshops. Proceedings. 2004 International Conference on*, pages 61–68, Aug. 2004.

[66] Guoliang Xing, Xiaorui Wang, Yuanfang Zhang, Chenyang Lu, Robert Pless, and Christopher Gill. Integrated coverage and connectivity configuration for energy conservation in sensor networks. *ACM Trans. Sen. Netw.*, 1(1):36–72, 2005.

[67] Fan Ye, Gary Zhong, Songwu Lu, and Lixia Zhang. Peas: A robust energy conserving protocol for long-lived sensor networks. *IEEE International Conference on Network Protocols (ICNP)*, 00:200, 2002.

[68] Chi Zhang, Yanchao Zhang, and Yuguang Fang. Localized algorithms for coverage boundary detection in wireless sensor networks. *Wirel. Netw.*, 15(1):3–20, 2009.

[69] Honghai Zhang and Jennifer Hou. Maintaining sensing coverage and connectivity in large sensor networks. *Ad Hoc and Sensor Wireless Networks (AHSWN)*, 2005.

[70] Zongheng Zhou, S. Das, and H. Gupta. Variable radii connected sensor cover in sensor networks. *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 387–396, Oct. 2004.

[71] Zongheng Zhou, Samir R. Das, and Himanshu Gupta. Variable radii connected sensor cover in sensor networks. *ACM Trans. Sen. Netw.*, 5(1):1–36, 2009.

[72] 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.

APPENDIX:

RELATED POSTERS/PAPERS

1. Akshaye Dhawan, C.T. Vu, A. Zelikovsky, Y. Li and S.K. Prasad. Maximum lifetime of sensor networks with adjustable sensing range. In Proceedings of the International Workshop on Self-Assembling Wireless Networks, 2006

2. Sushil K. Prasad and Akshaye Dhawan. Distributed Algorithms for lifetime of Wireless Sensor Networks. In Proceedings of the 14th International Conference on High Performance Computing, Springer, 2007.

3. Akshaye Dhawan and Sushil K. Prasad. A Distributed Algorithmic Framework for Coverage Problems in Wireless Sensor Networks. In Proceedings of the 22nd IEEE Parallel and Distributed Processing Symposium, 2008.

4. Akshaye Dhawan and Sushil K. Prasad. Energy Efficient Distributed Algorithms for Sensor Target Cove rage based on Properties of an Optimal Schedule. In Proceedings of the 15th International Conference on High Performance Computing, Springer, 2008.

5. Akshaye Dhawan. On Distributed Algorithms for the lifetime of Wireless Sensor Networks, In HiPC08 Student Symposium: 15th International Conference on High Performance Computing, 2008.

6. Akshaye Dhawan and Sushil K. Prasad. A Distributed Algorithmic Framework for Coverage Problems in Wireless Sensor Networks, In International Journal of Parallel, Emergent and Distributed Systems(IJPEDS), Vol 24, Issue 4, pp. 331, 2009

7. Akshaye Dhawan. Distributed Algorithms for Maximizing the Network Lifetime in Wireless Sensor Networks based on Cover Set Dependencies. In Ph.D. Forum, 23rd IEEE Parallel and Distributed Processing Symposium, 2009.

8. Akshaye Dhawan and Sushil K. Prasad. Taming the Exponential State Space of the Maximum Lifetime Sensor Cover Problem. In Proceedings of the 16th International Conference on High Performance Computing, 2009.