Randomized k-Coverage Algorithms For Dense Sensor Networks

Mohamed Hefeeda
School of Computing Science
Simon Fraser University
Surrey, BC, Canada

Majid Bagheri
School of Computing Science
Simon Fraser University
Surrey, BC, Canada

***Abstract*— We propose new algorithms to achieve k-coverage in dense sensor networks. In such networks, covering sensor locations approximates covering the whole area. However, it has been shown before that selecting the minimum set of sensors to activate from an already deployed set of sensors is NP-hard. We propose an efficient approximation algorithm which achieves a solution of size within a logarithmic factor of the optimal. We prove that our algorithm is correct and analyze its complexity. We implement our algorithm and compare it against two others in the literature. Our results show that the logarithmic factor is only a worst-case upper bound and the solution size is close to the optimal in most cases. A key feature of our algorithm is that it can be implemented in a distributed manner with local information and low message complexity. We design and implement a fully distributed version of our algorithm. Our distributed algorithm does not require that sensors know their locations. Comparison with two other distributed algorithms in the literature indicates that our algorithm: (i) converges much faster than the others, (ii) activates near-optimal number of sensors, and (iii) significantly prolongs (almost doubles) the network lifetime because it consumes much less energy than the other algorithms.**

I. Introduction

Mass production of sensor devices with low cost enables the deployment of large-scale sensor networks for real-life applications such as forest fire detection and vehicle traffic monitoring. A fundamental issue in such applications is the quality of monitoring provided by the network. This quality is usually measured by how well deployed sensors *cover* a target area. In its simplest form, coverage means that every point in the target area is monitored by, i.e., within the sensing range of, at least one sensor. This is called 1-coverage. In this paper, we consider the more general k-coverage ($k \geq 1$) problem, where each point should be within the sensing range of k or more sensors. Covering each point by multiple sensors is desired for many applications, because it provides redundancy and fault tolerance. Furthermore, k-coverage is necessary for the proper functioning of many other applications, such as intrusion detection [1], data gathering [2], and object tracking [3].

When deployed sensors are dense, area coverage can be approximated by point coverage. That is, if all sensor locations are covered by the set of activated sensors, the entire area is covered. In this paper, we address the problem of selecting the minimum set of sensors to activate from an already deployed set of sensors such that all locations are k-covered. Achieving a minimal set of sensors is critical, because it reduces interference among active sensors, reduces total energy consumption, and thus prolongs the lifetime of the whole network.

The problem of selecting the minimum number of sensors, however, is NP-hard [4].[1] We propose an efficient approximation algorithm for it, which achieves a solution of size within a logarithmic factor of the optimal and terminates quickly (in the order of seconds in most cases). We show by simulation that although the approximation factor is logarithmic, it is only a worst-case upper bound and the solution size is close to the optimal in most cases. We take a novel approach in solving the k-coverage problem. In particular, we model the problem as a set system for which an optimal *hitting set* corresponds to an optimal solution for coverage. Finding the optimal hitting set is NP-hard [6], but there is an efficient approximation algorithm for it [7]. Our k-coverage algorithm is inspired by the approximation algorithm for the optimal hitting set problem. We prove that our algorithm is correct and analyze its complexity. We implement our algorithm and compare it against other centralized algorithms in the literature. Our comparison reveals that our algorithm is about four orders of magnitude faster than the currently-known k-coverage algorithms.

A key feature of our centralized k-algorithm is that it can be implemented in a distributed manner with local information and low message complexity. We design and implement a fully distributed version of our algorithm. Our distributed algorithm does not require sensors to know their locations. Comparison with two other distributed algorithms in the literature indicates that our algorithm: (i) converges much faster than the others, (ii) activates near-optimal number of sensors, and (iii) significantly prolongs (almost doubles) the network lifetime because it consumes much less energy than the other algorithms.

The rest of the paper is organized as follows. In Section II, we review previous works. In Section III, we present an overview of the k-coverage problem and our solution approach. The details and analysis of our k-coverage algorithms are presented in Sections IV and V. We evaluate our algorithms and compare them against others in Section VI. We conclude

[1] Note that this problem is different from the problem of *placing* sensors in an area to cover it, which can be solved efficiently [5].

the paper in Section VII.

II. Related Work

The closest work to ours are [4] and [8]. In [4], the authors address the problem of selecting the minimum number of sensors to activate from a set of already deployed sensors for k-coverage. They prove that the problem is NP-hard since it is an extension of the dominating set problem [6]. They formulate the problem and provide a centralized approximation solution based on integer linear programming. The algorithm works by relaxing the problem to ordinary linear programming, where the variables may take real values. They also design a distributed algorithms, PKA, which uses pruning to reduce the number of active sensors. The work in [8] presents a centralized algorithm that works by iteratively adding a set of nodes which maximizes a measure called *k-benefit* to an initially empty set of nodes. The authors also present a distributed algorithms, DPA, that works by pruning unnecessary nodes. We compare our algorithms against the algorithms in [4], [8].

III. The K-Coverage Problem and Our Solution Approach

Problem 1 (k-Coverage Problem): Given n already deployed sensors in a target area, and a desired coverage degree $k \geq 1$, select a minimal subset of sensors to cover all sensor locations such that every location is within the sensing range of at leas k different sensors. It is assumed that the sensing range of each sensor is a disk with radius r, and sensor deployment can follow any distribution.

The above k-coverage problem is proved to be NP-hard by reduction to the minimum dominating set problem in [4]. We propose an efficient approximation algorithm for solving the k-coverage problem. We start describing our solution approach with the following definition [7].

Definition 1 (Set System and Hitting Set): A set system $(X, \mathcal{R})$ is composed of a set X and a collection $\mathcal{R}$ of subsets of X. We say that $H \subseteq X$ is a hitting set if H has a non-empty intersection with every element of $\mathcal{R}$, that is, $\forall R \in \mathcal{R}$ we have $R \cap H \neq \emptyset$.

Our solution does not require a grid deployment, and any node deployment such as uniform or Poisson distribution can be used. We define X to be the set of all sensor locations. Thus, we have $|X| = n$. We define the collection $\mathcal{R}$ as follows. For each point p in X, we draw a circle of radius r centered at p. All points in X that fall within that circle constitute one set in $\mathcal{R}$. Fig. 1(b) shows only three elements of $\mathcal{R}$ that correspond to the three highlighted points p_1, p_2, p_3 in Fig. 1(a). Now the minimum hitting set problem on $(X, \mathcal{R})$ is to find the minimum set of points in X that hit (intersect) all elements (disks) of $\mathcal{R}$. Fig. 1(c) shows a possible hitting set for the three disks of $\mathcal{R}$ shown in Fig. 1(b). The hitting set has two points c_1 and c_2. If we consider c_1 and c_2 to be locations of sensors, we will ensure that points p_1, p_2 and p_3 are 1-covered, because each of them is within the sensing range of at least one of the sensors located at c_1 and c_2, as shown in Fig. 1(c).

For k-coverage, elements in the hitting set are not locations for individual sensors. Rather, each element in the hitting set is a center of what we call a *k-flower*, which is a set of k sensors that all intersect at that center point. Fig. 1(d) shows one 3-flower centered at point c that 3-covers point p_3. Details of constructing k-flowers are discussed in Section IV. The k-coverage problem now reduces to finding a minimum hitting set where elements in that set are the centers of k-flowers. Since finding the minimum hitting set is NP-hard, we try to find a near optimal hitting set. We propose an approximation algorithm that uses the concept of ϵ-nets [9], which is defined as follows.

Definition 2 (ϵ-Net): Let $0 < \epsilon \leq 1$ be a constant. The set $N \subseteq X$ is called an ϵ-net for the set system $(X, \mathcal{R})$ if N has a non-empty intersection with every element of $\mathcal{R}$ of size greater than or equal to $\epsilon|X|$, that is, $\forall R \in \mathcal{R}$ such that $|R| \geq \epsilon|X|$ we have $R \cap N \neq \emptyset$.

The definition of ϵ-net is similar to that of the hitting set, except that the ϵ-net is required to hit only *large* elements of $\mathcal{R}$ (ones that are greater than or equal to $\epsilon|X|$), while the hitting set must hit every element of $\mathcal{R}$. This similarity is exploited by our approximation algorithm to find a near optimal hitting by finding ϵ-nets of increasing sizes (i.e., decreasing ϵ) till one of them hits all elements of $\mathcal{R}$. For this to work, we clearly need to efficiently: (i) compute ϵ-nets, and (ii) verify coverage. We use a simple verifier that checks all points in $O(n)$ steps. Computing ϵ-nets can be done efficiently for set systems with finite VC-dimensions (defined in [9]). Specifically, Haussler and Welzl [9] show that for any set system $(X, \mathcal{R})$ with a finite VC-dimension d, randomly sampling $m \geq \max\left(\frac{4}{\epsilon}\log\frac{2}{\delta}, \frac{8d}{\epsilon}\log\frac{8d}{\epsilon}\right)$ points of X constitutes an ϵ-net with a probability at least $1-\delta$, where $0 < \delta < 1$. Notice that m does not depend on the size of X, which allows X to be arbitrarily large with no effect on the size of the ϵ-net. Brönnimann and Goodrich [7] further extend the concept of ϵ-net by assigning *weights* to elements of X. Weights accelerate the process of finding a near optimal hitting set, and help in establishing an upper bound on its size, as we discuss in Section IV.

The VC-dimension of our set system is proved to be 3 by the following lemma. Due to space limitation, the proof is given in [10].

Lemma 1: Consider the set system $(X, \mathcal{R})$, where X is the set of points, and $\mathcal{R}$ contains a disk of radius r for each point in X. This set system has a VC-dimension of 3.

To summarize, we model the k-coverage problem as a set system $(X, \mathcal{R})$ where X is the set of sensor locations and $\mathcal{R}$ is the collection of subsets of X created by intersecting disks of radius r with points of X. This set system has a VC-dimension of 3, therefore, we can efficiently implement a *net-finder* algorithm to find ϵ-nets of various sizes. Our approximation algorithm for the k-coverage problem employs the net-finder to compute ϵ-nets of increasing sizes, and for each ϵ-net it verifies the coverage until all points are sufficiently covered. We assign weights to points of X to guarantee termination and to bound the approximation factor of the output solution.

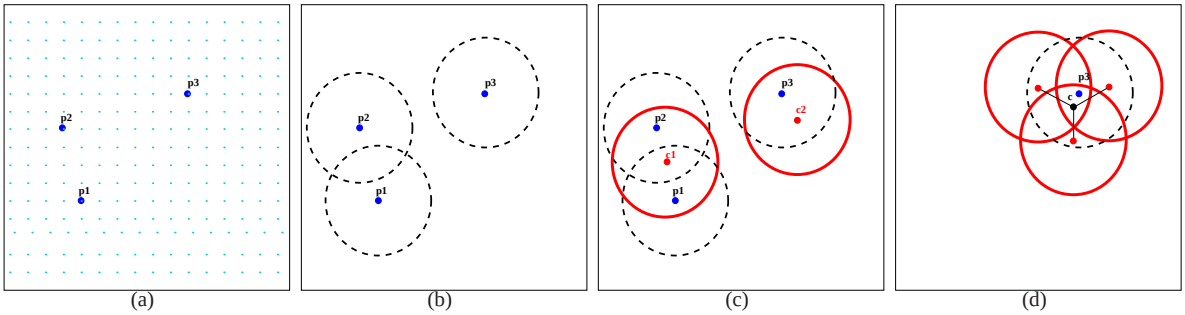

Fig. 1. Modeling the k-coverage problem as a set system $(X, \mathcal{R})$. (a) shows the set of points which constitute X. (b) shows only three subsets of $\mathcal{R}$ that are associated with the three highlighted points in (a). (c) shows a hitting set $\{c_1, c_2\}$ that 1-covers the three subsets in (b). (d) shows one 3-flower that 3-covers only one subset in $\mathcal{R}$.

Finally, each element in the output represents the center of what we call a k-flower, which is a set of k sensors that all intersect at that center point and should be activated for k-coverage.

IV. Centralized K-Coverage Algorithm

The pseudo code of the k-coverage algorithm, which we call RKC (Randomized k-Coverage algorithm), is given in Fig. 2. The algorithm takes as input the set of sensor locations X, sensing range of sensors r, and required degree of coverage k. If the algorithm succeeds, it will return a subset of nodes to activate in order to ensure k-coverage. The algorithm may only fail if activating all sensors is not enough for k-coverage because of low density. The minimum required density can be calculated as follows. If every point is to be k-covered, it has to be in the sensing range of at least k sensors. Thus, for each node p, there should be at least k other nodes inside a disk of radius r centered at p.

In every single iteration of the while loop, the algorithm tries up to $4c\log(n/c)$ $\frac{1}{c}$–nets one at a time (the for loop in lines 5–11). Each $\frac{1}{c}$–net is computed by the net-finder (Section IV-A), and hits all disks with weight greater than or equal to $\frac{1}{c}|X|$. For each net, the verifier checks whether this net is a hitting set, i.e., it completely k-covers all points. We use a simple verifier that checks all points in $O(n)$ steps.[2] If a net is a hitting set, the algorithm returns it and terminates. Otherwise, the algorithm doubles the weight of a point that was under covered by that net. Then, the algorithm chooses another $\frac{1}{c}$–net. Points with increased weights will have higher probability of being included in the new net. The size of each returned $\frac{1}{c}$–net is $O(c\log c)$ (see the description of the net-finder algorithm for details). The reason behind trying up to $4c\log(n/c)$ nets is that a result (Lemma 3.4) in [7] states that if there is a hitting set of size c, the weight doubling process cannot iterate more than $4c\log(n/c)$ times. This also means if we iterate beyond $4c\log(n/c)$ without finding a hitting set, it is guaranteed that there is no hitting set of size c [7]. This helps us to establish the following bound on the number of sensors required to achieve k-coverage.

Lemma 2: The solution returned by the k-coverage algorithm is no more than a logarithmic factor of the optimal number of sensors required to k-cover all sensor locations.

Proof: Suppose that the algorithm terminates with c and the optimal number of nodes required for k-coverage is $\widehat{N} \leq c$. This means that the algorithm has failed to find a solution for $c/2$. Since the algorithm iterated $4(c/2)\log(n/(c/2))$ times, doubling weights of uncovered points in each iteration, then by Lemma 3.4 in [7], there is no hitting set of size $c/2$. That is, we must have $\widehat{N} > c/2$. Therefore, we have $c < 2\widehat{N}$. Since the size of the $\frac{1}{c}$-net is $O(c\log c)$, the size of the solution is $O(\widehat{N}\log\widehat{N})$. ■

Notice that the analysis in the above lemma is not tight. Our simulation results (Section VI) show that the upper bound in this lemma is indeed very conservative, and our algorithm produces solution sizes that are a constant factor from the optimal in most cases.

Next, we prove the time complexity of the algorithm in the following lemma, the proof is given in [10].

Lemma 3: The k-coverage algorithm terminates in time $O(n\log n(T_F + T_V))$, where T_F and T_V are the running times of the net-finder and verifier algorithms, respectively.

A. The Net-Finder Algorithm

The idea of the net-finder algorithm is based on Corollary 3.8 in [9], which states that randomly selecting at least $\max\left(\frac{4}{\epsilon}\log\frac{2}{\delta}, \frac{24}{\epsilon}\log\frac{24}{\epsilon}\right)$ points of the set X yields an ϵ-net with a probability at least $1-\delta$, where $0 < \delta < 1$. Selecting an arbitrary small δ yields an ϵ-net with probability almost 1.

Let the term *net-size(ϵ)* denote the number of k-flowers in the ϵ-net. The net-finder algorithm iterates for *net-size(ϵ)* steps, and in every iteration, selects a random point q biased based on the weights. Then it finds a k-flower centered at q and adds it to the *net*. Any point q is selected with probability $w(q)/w(X)$, where w is a function which assigns weights to points. The weight of a set is the summation of weights of all points in that set. After the center point q of the k-flower is selected, k other points $p_1, \ldots, p_k$ are selected uniformly inside a disk of radius r centered at q. The location of each of these points is

[2]Asymptotically more efficient verifiers are possible to design using order-k Voronoi diagrams [11]. However, these verifiers are complex to implement in practice, and the performance gain is not significant due to the large constants in the time complexity.

Randomized K-Coverage: RKC(X, r, k)

```
1.  c = 1; // sets the initial size of ε-net
2.  while (net-size(1/c) ≤ n) do
3.      set weights of all points to 1;
4.      ε = 1/c;
5.      for i = 1 to 4c log n/c
6.          N = net-finder(X, k, ε, r);
7.          u = verifier (X, N, k, r);
8.          if (u == null)
9.              return N;
10.         else
11.             double weight of u;
12.     c = 2 × c;
13. return ∅;
```

Fig. 2. A centralized approximation algorithm for the k-coverage problem.

given by: $p_i = (x_q + d_i \cos\theta_i,\ y_q + d_i \sin\theta_i)$, where x_q and y_q are coordinates of q, and θ_i and d_i are selected at random from $[0, 2\pi]$ and $[0, r]$, respectively.

The following lemma provides the time complexity of the net-finder algorithm and the size of the net returned [10].

Lemma 4: The algorithm net-finder terminates in $O(n \log n)$ steps and returns an ϵ-net of size $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$.

Remark: A more efficient net-finder algorithm, i.e., one that returns an ϵ-net of size $O(\frac{1}{\epsilon})$, is possible to design [12]. However, the constant in this linear bound is quite high. Moreover, the algorithm involves triangulation which requires sensors to be aware of their locations, and more importantly, it is not clear how the algorithm can be implemented in a distributed manner. Therefore, although the efficient net-finder in [12] would make our RKC algorithm produce a solution that is a constant factor from the optimal, we opt to use the simpler net-finder algorithm because it can be implemented in a distributed manner, and it produces near-optimal results on the average, as shown by our simulations in Section VI.

B. Algorithm Correctness and Complexity

The following theorem proves that our algorithm is correct, provides its time complexity, and proves the upper bound on the solution.

Theorem 1: The k-coverage algorithm (RKC) in Fig. 2 ensures that every point in the area is k-covered, terminates in $O(n^2 \log^2 n)$ steps, and returns a solution of size at most $O(\widehat{N} \log \widehat{N})$, where $\widehat{N}$ is the minimum number of sensors required for k-coverage.

Proof: Suppose that the algorithm terminates by providing a set S of sensor locations. By construction, this set of points is guaranteed to hit every disk of radius r. Since for our set system $(X, \mathcal{R})$, we put a disk in $\mathcal{R}$ for each point $p \in X$, there should be at least one element (i.e., a k-flower) in S that hits the disk centered at p. In addition, the center of each sensor in the k-flower is within a distance r from p (see Section IV-A for details on constructing k-flowers). Therefore, p is k-covered by sensors of this k-flower. Hence, all points are k-covered by sensors in S.

The time complexity follows from Lemmas 3, and 4, and by using a simple verifier that checks all n points in $O(n)$ steps. The bound on the solutions size follows from Lemma 2. ■

V. DRKC: Distributed Randomized K-Coverage Algorithm

In the previous section, we presented a centralized algorithm for the k-coverage problem. A key feature of this algorithm is that it does not rely heavily on global information. Therefore, it can be implemented in a distributed manner.

Our centralized k-coverage algorithm (shown in Fig. 2) maintains two global variables: the size of the current ϵ-net, and weights of all points. At every iteration of the outer loop, the size of the ϵ-net is doubled, and at every iteration of the inner loop, the weight of one under-covered node is doubled. The basic idea of our distributed algorithm, which we call DRKC (Distributed Randomized k-Coverage algorithm), is to emulate the centralized algorithm by keeping *local* estimates for these two global variables.

Estimating the current ϵ-net size and the total weight in the network allows a node to decide (locally) whether it should be a member of the ϵ-net.If a node decides to be part of the ϵ-net, it will activate k other nodes to create a k-flower as in the centralized algorithm by broadcasting an ACTIVATE message to its neighbors. When a node receives an ACTIVATE message, it becomes active and broadcasts a NOTIFY message informing all its neighbors that it has become active.

Finally, k-coverage verification in the centralized algorithm is done by checking all nodes one by one. In the distributed algorithm, each node independently checks its own coverage by listening to NOTIFY messages exchanged in its neighborhood, and counting number of active nodes. A node terminates the algorithm if it is sufficiently covered. Otherwise, it doubles its weight, and starts another loop iteration. A node may also terminate the algorithm if it has been looping for $\log n$ steps without getting sufficiently covered, which can occur because of low node density.

In the following theorem, we provide the average- and worst-case communication complexities of the DRKC protocol. The proof as well as detailed description of DRKC are given in [10].

Theorem 2: The number of messages sent by a node in any round of the DRKC protocol is $O(\log n)$ in the worst case, and $O(1)$ on average.

VI. Performance Evaluation

Due to space limitation, we present only a sample of our results. More details and plots are given in [10]. We first compare the output size of our RKC algorithm against the asymptotic necessary and sufficient conditions for k-coverage proved in [2] for uniformly deployed sensors. We use a large area of size $1000m \times 1000m$ with $30,000$ deployed sensors and vary the sensing range r. The results for $k = 4$ are

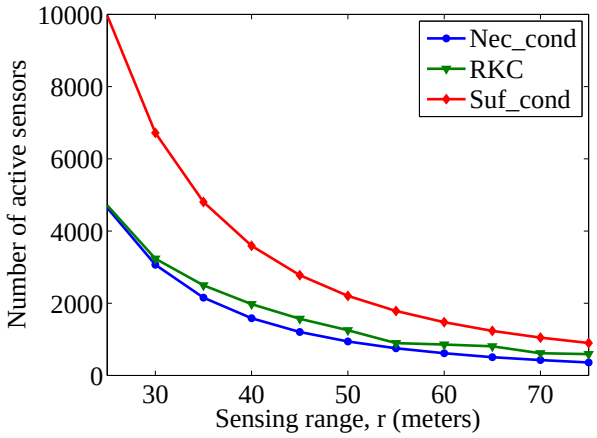

Fig. 3. Efficiency of our centralized k-coverage algorithm (RKC). The figure compares the number of active sensors produced by our RKC algorithm versus the necessary (Nec_cond) and sufficient (Suf_cond) conditions proved in [2].

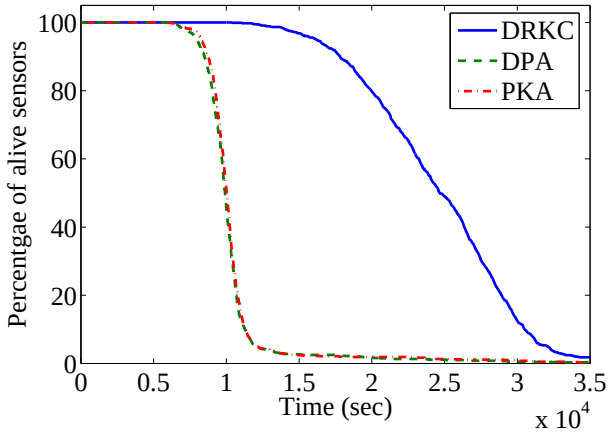

Fig. 4. Comparing the network lifetime under different distributed k-coverage algorithms.

shown in Fig. 3, where Nec_cond and Suff_cond denote the necessary and sufficient conditions, respectively. The figure shows that our algorithm does not unnecessarily activate too many sensors, because its output is very close to the necessary condition. The results of this experiment show that the worst-case logarithmic factor proved in Theorem 1 is very conservative, and on average our centralized algorithm produces near-optimal number of active sensors.

Next, we compare our centralized RKC algorithm against two other centralized k-coverage algorithms: CKC [8] and LPA [4]. Simulation results show that our centralized RKC algorithm runs up to four orders of magnitude faster, while producing same or better solution sizes than the other algorithms [10].

Now, we compare our distributed DRKC algorithm against two other distributed k-coverage algorithms: DPA [8] and PKA [4] along various performance metrics. The results indicate that the DRKC algorithm converges much faster than the other two algorithms and always results in much smaller numbers of activated sensors [10].

Finally, we look at the lifetime of the sensor network under different distributed algorithms. we compare the percentage of alive sensors as the time progresses for the three algorithms. As Fig. 4 indicates, our algorithm prolongs (almost doubles) the lifetime of the network, because it consumes much smaller amount of energy than the other two algorithms [10].

VII. CONCLUSIONS

In this paper, we presented a novel approach to solve the k-coverage problem in large-scale sensor networks. We modeled the k-coverage problem as a set system for which an optimal hitting set corresponds to an optimal solution for k-coverage. We proposed an approximation algorithm for computing near-optimal hitting sets efficiently. We proved that our algorithm produces a solution that is at most a logarithmic factor from the optimal. Furthermore, we showed through simulation that the logarithmic factor is only a conservative upper bound, and the solution is typically close to the optimal in most cases. We compared our algorithm against the currently-known k-coverage algorithms and showed that it runs up to four orders of magnitude faster, while producing same or better solution sizes than the other algorithms. We also designed and implemented a fully distributed version of our algorithm that uses only local information. Our distributed algorithm has low message complexity and it does not require sensors to know their locations.

REFERENCES

[1] D. Mehta, M. Lopez, and L. Lin, "Optimal coverage paths in ad-hoc sensor networks," in *Proc. of IEEE International Conference on Communications (ICC'03)*, May 2003.

[2] S. Kumar, T. H. Lai, and J. Balogh, "On k-coverage in a mostly sleeping sensor network," in *Proc. of ACM International Conference on Mobile Computing and Networking (MOBICOM'04)*, Philadelphia, PA, September 2004, pp. 144–158.

[3] D. Hall and J. Llinas, *Handbook of Multisensor Data Fusion*. CRC Press, 2001.

[4] S. Yang, F. Dai, M. Cardei, and J. Wu, "On connected multiple point coverage in wireless sensor networks," *Journal of Wireless Information Networks*, May 2006.

[5] R. Iyengar, K. Kar, and S. Banerjee, "Low-coordination topologies for redundancy in sensor networks," in *Proc. of ACM Mobihoc'05*, Urbana-Champaign, IL, May 2005.

[6] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[7] H. Bronnimann and M. Goodrich, "Almost optimal set covers in finite VC-dimension," *Discrete and Computational Geometry*, vol. 14, no. 4, April 1995.

[8] Z. Zhou, S. Das, and H. Gupta, "Connected k-coverage problem in sensor networks," in *Proc. of International Conference on Computer Communications and Networks (ICCCN'04)*, Chicago, IL, October 2004.

[9] D. Haussler and E. Welzl, "Epsilon-nets and simplex range queries," *Discrete and Computational Geometry*, vol. 2, no. 1, December 1987.

[10] M. Hefeeda and M. Bagheri, "Efficient k-coverage algorithms for wireless sensor networks," School of Computing Science, Simon Fraser University, Tech. Rep. TR 2006-22, September 2006.

[11] A. So and Y. Ye, "On solving coverage problems in a wireless sensor network using voronoi diagrams," in *Proc. of Workshop on Internet and Network Economics (WINE'05)*, Hong Kong, December 2005.

[12] J. Matousek, R. Seidel, and E. Welzl, "How to net a lot with little: Small ϵ-nets for disks and halfspaces," in *Proc. of the 6th Annual ACM Symposium on Computational Geometry (SoCG'90)*, Berkeley, CA, June 1990.