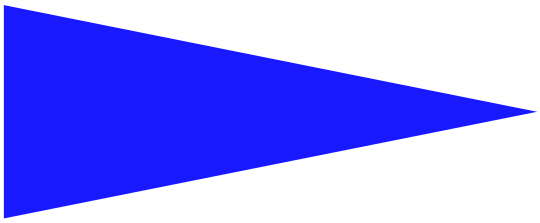
Self-* Distributed Query Region Covering in Sensor Networks

Ajoy Datta, Maria Gradinariu, Preethy Linga, Philippe Raïpin-Parvédy

I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

P U B L I C A T I O N
I N T E R N E
N° 1715



SELF-* DISTRIBUTED QUERY REGION COVERING IN SENSOR NETWORKS

A.K. DATTA , M. GRADINARIU, P. LINGA , P. RAIPIN-PARVÉDY

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

Self-* Distributed Query Region Covering in Sensor Networks

A.K. Datta* , M. Gradinariu, P. Linga , P. Raipin-Parvédy

Systèmes communicants
Projet ADEPT

Publication interne n ° 1715 — Mai 2005 — 20 pages

Abstract: In this paper** , we design *self-*∗ novel solutions to the minimal connected sensor cover problem. The concept of *self-*∗ has been used to include fault-tolerant properties like *self-configuring*, *self-reconfiguring/self-healing*, etc. We will present two *self-stabilizing, fully distributed, strictly localized, and scalable* solutions, and show that these solutions are both self-configuring and self-healing. The proposed solutions are space optimal in terms of the number of states used per node. Another feature of the proposed algorithms is that the faults are contained only within the neighborhood of the faulty nodes. The paper also includes a comparison of the performance of the two proposed solutions in terms of the stabilization time, cover size metrics, and ability to cope with transient and permanent faults.

Key-words: Connectivity, coverage, fault-tolerance, query response system, self-configuring, self-healing, self-stabilizing, Self-∗, sensor networks.

(Résumé : tsvp)

* Ajoy Kumar Datta and Preethy Linga are with School of Computer Science, University of Nevada Las Vegas
** A preliminary version of this work was published in [5]

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes
Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Couverture Auto-* d'une zone d'impacte dans les réseaux de capteurs

Résumé : Ce rapport présente une première solution distribuée de type auto* au problème de couverture connectée d'une zone d'impacte d'une requête dans un réseau de capteurs.

Dans les réseaux de capteurs une requête est envoyée pour capter des données ou des événements dans une zone géographique appelée - zone d'impacte. Généralement suite au déploiement massif des capteurs dans une zone se trouvent plus de capteurs que le nombre minimal nécessaire à répondre à une requête donnée. Pour des raisons d'économie d'énergie une partie des capteurs doivent être passés dans un mode de veille. Le problème est de trouver l'ensemble minimal de capteurs qui couvre une zone d'impacte tout en gardant la connectivité de la couverture.

Nous proposons deux solutions de type auto∗ (auto-organisantes, auto-stabilisantes) à ce problème, les deux étant optimales en nombre d'états par processus.

Mots clés : réseau de capteurs, couverture connexe, auto∗

1 Introduction

After spending the first era of computing with mainframes, we are now in the era of personal computing. The next wave, the third era of computing was visioned by Late Mark Weiser. In 1988 at the Computer Science Lab at Xerox PARC, he articulated the next age of computing, called *ubiquitous computing* [14]. In ubiquitous world, we expect to see thousands of invisible computing devices used per person, maybe, even in a household. We can now build ad-hoc networks composed of a large number of low-cost, low-power, and small sensor nodes. These ad-hoc *wireless sensor networks* [8] have applications everywhere — military, business, commercial, health, and home.

Sensor networks [1] are expected to be very large. In many applications, they will be densely deployed. These networks are energy constrained. Not only the sensors have limited battery power, it is extremely difficult if not impossible to replace the battery. They may be deployed in inaccessible terrains or disaster areas. So, it is very important to design energy efficient sensor networks to enable untethered and unattended operation for an extended period of time. The topology may change very frequently due to various reasons, like position, reliability, available energy, malfunctioning, etc. Thus, designing reliable wireless sensor networks is challenging.

Deploying pre-configured network of a huge number of sensors is impractical. Expecting to be able to manually maintain that size of a network is absurd. Considering all these constraints, the sensor network must be *self-configuring* and *self-maintaining* or *self-healing* [17]. A system is considered to be *self-configuring* if starting from an arbitrary state and an arbitrary input, the system will eventually satisfy the specification or start behaving properly. A *self-healing* system automatically recovers from different perturbations and dynamic changes. A self-healing system can also be characterized as a *self-maintaining* system.

Software systems are being used for almost all business-critical applications. Thus, the availability of these systems is extremely important. The system must be able to adjust to different inputs, adapt to all possible changes of the environment, and handle different faults. The different concepts or terms encapsulated in *self-*$*$ have been introduced to characterize different ways of detecting, adjusting, and recovering from the above situations.

In this paper, we will present two *self-stabilizing* solutions to an important energy saving problem in sensor networks. Then we will show that these solutions can also be considered as self-$*$ solutions. In a self-stabilizing system, every computation, upon starting from an arbitrary state, eventually reaches a state from where the computation satisfies the specification. The paradigm of self-stabilization, introduced by Dijkstra in 1974 [6], is considered to be the most unified strategy to design fault-tolerant systems. Although it is intended to handle transient faults (e.g., memory errors, message omissions/duplications, program counter corruptions), it has been established that almost all types of faults can be dealt with in a stabilizing manner. Readers can refer to [7] for an overview of self-stabilization.

Motivation and Related Work. In sensor networks, *queries* are sent from some devices (could be a satellite, PDA, laptop, or any computer) to sense some data/events over some time period and a geographical region, called a *query region*. A query could be like "Every I ms for the next Y seconds, tell me how many vehicles of type T are moving in direction D in region R". The query region is usually a subset of the total region covered by all the sensors in the network. Considering the limited energy available, one of the most important goals in any protocol on sensor networks is to save energy. Since the sensors are usually densely deployed, there are usually a lot more sensors than required to process a given query. One possible way to minimize usage of energy is not to keep all sensor nodes fully active all the time. Some of them can be put in passive mode some times while

others are active in sensing the data in the environment. However, for the sensor networks to be effective, the active nodes must be able to cover the whole query region and maintain the network connectivity at all times.

In [9], a new optimization problem in sensor networks, called the *connected sensor cover* was introduced to model the query response system. The problem can be informally defined as follows: Given a query over a sensor network, select a minimum or nearly minimum set of sensors, called connected sensor cover, such that the selected sensors cover the query region, and form a connected network among themselves. In its general form, this problem is known to be NP-hard [9, 10].

In this work, we address the minimal connected sensor cover problem (introduced in [9]) that constructs a minimal cover with respect to the inclusion operation. Coverage and connectivity issues were addressed in literature with different assumptions: centralized, distributed using some coordinators, and probabilistic. An overview of these approaches is presented below. Two self-organizing solutions were presented in [9]. None of the solutions is localized. The first solution is centralized. In the second solution, a particular sensor node behaves as the coordinator or leader. This special node collects all the global information related to the possible new sensor nodes to be added, then decides which ones to choose.

The issues of coverage and connectivity, and the relations between them were analyzed in a unified framework in [13]. The CCP protocol [13] can be used to provide different degrees of coverage. It was shown in [13, 18] that if the communication range is at least twice of the sensing range, the complete coverage implies connectivity. When the above condition does not hold, CCP was integrated with SPAN [3] to provide both coverage and connectivity. However, in SPAN, the nodes need to maintain information about two-hop neighborhood. SPAN is a connectivity maintenance protocol where a node volunteers to be a coordinator when it finds that two of its neighbors cannot communicate with each other directly or indirectly. After a node decides to be a coordinator, it announces that with a random delay to reduce the number of redundant coordinators. A similar approach was discussed in ASCENT [2]. ASCENT nodes use the number of active neighbors and message losses to decide if they should be active or passive. However, this protocol does not guarantee complete coverage of the query region.

Probabilistic studies related to coverage and connectivity in unreliable sensor networks were done in [11]. A sensor grid network of unit area was considered. This work includes a necessary and sufficient condition for the network to remain covered and connected in terms of the probability of a node to be active (i.e., not failed) and transmission radius of the nodes. Some optimal conditions for coverage were established in [18]. An algorithm for coverage was proposed based on those optimal conditions. However, that result is valid only when complete coverage implies connectivity (as discussed above). A coverage protocol using a random delay to announce decision to turn off was proposed in [12]. The issue of connectivity was not addressed in [12]. The GAF protocol [15] uses GPS to reduce the redundant nodes to maintain routing paths in ad-hoc networks. A randomized probing-based density control algorithm was used to maintain coverage under node failures in PEAS protocol [16]. The probing range can be changed to provide different degrees of coverage.

Contributions. None of the above mentioned solutions explicitly addresses fault-tolerance issues and in particular, transient faults. The main contribution of this research is to design self-$*$ solutions to the connected sensor cover problem. To the best of our knowledge, these are the first localized, distributed, and self-$*$ solutions to the query connected cover problem in sensor networks. Localized solutions in large networks are desirable due to their high reliability and scalability. We implemented the self-$*$ properties by using the self-stabilization paradigm [7]. Self-stabilization is the most adapted

theoretical toolkit for the design of algorithms that cope with a broad range of faults in dynamic and large scale networks. Our solutions can handle different types of faults including node and link (wireless communication) failures, change of power level, topology changes due to faults or new joinings, and memory and program counter corruptions.

In the next section, we define the model and specify the connected sensor cover problem. In Sections 4 and 5, we present self-stabilizing solutions to the problem. Formal analysis and the proofs of the self* aspects of our solutions are presented in the Appendix. Discussion about the complexity of the algorithms and simulation results are included in Section 6. The experiments were conducted with respect to the following metrics: stabilization time, cover size and fault-tolerance. Finally, in Section 7, we present some concluding remarks and we will give some ideas to extend this research.

2 Preliminaries and Model

Sensor Network. In this research, we consider *sensor networks* [9, 13] consisting of a large number of sensors (also referred as *sensor nodes* and *nodes* in this paper) randomly distributed in a geographical region. We model the sensor network as a *directed* communication graph $G(V, E)$, where each node in V represents a sensor, and each edge $(i, j) \in E$, called *communication edge*, indicates that j is a *neighbor* of i.

For a sensor i, there is a region, called *sensing region*, which signifies the area in which the sensor i can sense a given physical phenomenon maintaining a desired confidence level. The *sensing range* of a sensor i indicates the maximum distance between i and any point p in the sensing region of i. A point p is *covered* (or monitored) by a sensor node i if the Euclidean distance between p and i is less than the sensing range of i.

The *communication region* of a sensor i (also called the *transmission region*) defines the area in which i can communicate directly (i.e., in single hop) with other sensor nodes. The maximum distance between node i and any other node j, where j is in the communication region of i, is called the *communication range* of i. A directed path (sequence) of sensors $i = i_1, i_2, \ldots, i_m = j$, where i_x is a neighbor of i_{x+1} for $1 \leq x \leq m-1$, is called a *communication path* from i to j. The length of the shortest (communication) path (which is the number of sensors on the shortest path) from i to j is called the *communication distance* from i to j.

Program. We consider the local shared memory model of communication as used by Dijkstra [6]. The program of every processor consists of a set of *shared variables* (henceforth, referred to as variables) and a finite set of actions. It can only write to its own variables, and read its own variables and variables owned by the neighboring nodes.

Each action is of the following form: $< label >:: < guard > \longrightarrow < statement >$. The guard of an action in the program of p is a boolean expression involving the variables of p and its neighbors. The statement of an action of p updates one or more variables of p. An action can be executed only if its guard evaluates to true. We assume that the actions are atomically executed, meaning, the evaluation of a guard and the execution of the corresponding statement of an action, if executed, are done in one atomic step.

The *state* of a node is defined by the values of its variables. The *state* of a system is the product of the states of all nodes. We will refer to the state of a node and system as a (*local*) *state* and (*global*) *configuration*, respectively.

Fault Model. This research deals with the following types of faults: (i) The state or configuration of the system may be arbitrarily corrupted. However, the program (or code) of the algorithm cannot be corrupted. (ii) Nodes may crash. That is, the faults can fail-stop nodes. (iii) Nodes may recover

or join the network. The topology (both actual and logical topologies) may change due to faults. Faults may occur in any finite number, in any order, at any frequency, and at any time.

3 Problem and Description

Our research is focused on designing a reliable, self-organizing, and self-healing query-response system. A query in sensor networks asks for some data/measurements/events sensed/observed over some period of time at some frequency over a geographical region. Upon receiving a query, the sensors will sense or measure the data and collaborate among themselves to disseminate or fuse the collective data to the sink of the query. Although a query can be initiated in the whole geographical region, typically, a query refers to a subset of the region, called the *query region* denoted by R_Q in this report.

The sensors only inside the query region should be involved in generating the response to the query. Considering the redundancy and our goal of designing an efficient query-response system, all sensors inside the query region should not be actively participating in the protocol to answer the query. Our approach is for the sensors inside R_Q to self-organize to form a logical network sufficient enough *to cover* the query region. However, in order for the sensors in the logical network (i.e., the region covered by the selected sensors) to be able to collaborate to detect the events, and compute and deliver the response, they must be able to communicate with each other directly or indirectly. That is, the logical graph not only needs to satisfy the coverage criterion, it must also be a strongly connected communication graph.

Definition 3.1 (Connected Sensor Cover). *Consider a sensor network G consisting of n sensors $I_1, I_2, \ldots, I_n$. Let S_i be the sensing region associated with sensor I_i. Given a query Q over a region R_Q in the sensor network, a set of sensors $\mathcal{SC}_Q = I_{i_1}, I_{i_2}, \ldots, I_{i_m}$ is called a* connected sensor cover *for Q if the following two conditions are satisfied:* **(a) Coverage:** $R_Q \subseteq (S_{i_1} \cup S_{i_2} \cup \ldots S_{i_m})$. **(b) Connectivity:** *The subgraph induced by $\mathcal{SC}_Q$ is strongly connected in the sense that any two sensors in this set can communicate with each other directly or indirectly.*

Definition 3.2 (Minimal Connected Sensor Coverage Problem). *Given a sensor network and a query over the network, the connected sensor coverage problem is to find the minimal connected sensor cover (we will call it $\mathcal{MCSC}_Q$). A cover is considered minimal if it does not include another connected cover.*

Additionally, we require the algorithm (solving the above problem) to be self-organizing, self-healing, and self-stabilizing [7, 17]. That is, regardless of the initial state (wrong initialization of the local variables, memory or program counter corruptions) nodes self-configure/self-organize using only local information in order to make the system self-stabilize to a *legitimate state*. The legitimate state is defined with respect to a minimal connected cover formed out of the nodes that can communicate with each other either directly or indirectly. The nodes in this set are the only nodes that remain active. Moreover, under various perturbations, such as node joins, failures (due to crash or energy loss), state corruptions, or weakening of power, the minimal connected cover should be able to self-heal without any external intervention and the impact should be confined within a tightly bounded region around the perturbed area.

In this paper, we will present two space optimal self-stabilizing, self-configuring and self-healing solutions to the connected sensor cover problem. Computing a minimum sensor cover in its general form is NP-hard [9, 10]. So, the proposed solutions make an attempt to approach an optimal solution

by checking and removing redundant sensor nodes from the final cover set. However, the solutions although suboptimal in terms of the number of sensors, satisfy Definition 3.2.

Note that removing redundancy while constructing the distributed connected covers is a difficult task, and its degree of accuracy depends on the exposure of a node. If nodes farther away are considered in the computation of the redundancy, the connected cover set will approach closer to the optimal value, i.e., it is expected to be smaller. This trade-off has been pointed out in [4]. Our solutions use only two states per node and use knowledge of nodes at a distance of up to two hops away. We conjecture that using only two states, it is impossible to construct a fully local (i.e., only the immediate neighborhood is known) self-stabilizing connected covers, hence our solutions use the minimal knowledge per node.

Our solutions do not require the sensors to have unique identifiers (ID's). However, each sensor i maintains a set of distinct labels, denoted as N_i, such that each label identifies a (unique) neighbor of i. Note that these labels are unique only *locally*.

The query region forms a convex region, and its boundary (hence, its center which is used by our first implementation) is known to all sensors. The energy level of the sensors may change over time due to various reasons. The proposed solutions cope with that.

We distinguish three types of sensors in or around the query region R_Q. In our algorithms, the rules for these three types of nodes are different. A sensor is termed as a *boundary sensor* if its sensing region intersects with the boundary of the query region R_Q. A sensor is called an *interior sensor* if its sensing region is completely inside the query region R_Q and it is not a boundary sensor. All sensors which are neither boundary nor interior are called *exterior sensors*.

Data Structures. Three variables (R_Q, N_i, and Pos_i) are used as **Constants** by the proposed solutions. That is, the algorithms do not write into these variables. The input query includes the geographical information about the region R_Q to be covered. N_i represents the neighboring sensors of sensor i. Our solutions assume that there is an underlying self-stabilizing topology maintenance protocol which computes N_i. The sensors use either some device or/and protocol to know their geometric location.

The algorithms also use two **Variables**: S_i and $Color_i$. S_i represents the sensing region of sensor i. The $Color$ variable is used to represent the different status of a sensor. Sensors can be either in *red* or *black* initially. Eventually, if a sensor turns *black* and stays in that color, then it is considered to be a member of $\mathcal{MCSC}_Q$. Other sensors will remain in *red* color.

Macros. The macros do not represent variables, but return values when referred to in the code. Our first solution assumes that the sensors know the location of the center. So, they can use their location information (Pos) to compute their distances to the center. Note that this information is used uniquely by the first solution. The macro $Dst(i)$ computes the distance of Sensor i from the center of R_Q uncovered by the sensing region of i. We consider directed communication graph of sensors. So, a sensor i may not have a two-way communication with all its neighbors. Sensor i may need this knowledge (i.e., which of its neighbors have a two-way communication with it) to check redundancy. The macro $BidirN(i)$ returns a set of neighbors of i that effectively have a bidirectional communication with i. The predicate $BlackConNbrs(i)$ is used to implement the redundancy checking. This predicate is the local version of the "Rule k" of [4]. It returns true if the subgraph of black neighbors of a node is connected. We also introduce a novel version of the "Rule k", $BlackConNbrs(i,t)$, that checks if the subgraph defined by the black neighbors of two neighboring nodes, i and t, is connected.

Module 3.1 Data structures used by Cover Algorithms (Algorithms $\mathcal{DMSC}$ and $\mathcal{IMSC}$).

.

Constants:

R_Q:: Query region; represented as a set in the algorithm;

N_i :: Set of sensors within the communication range of Sensor i;

Pos_i :: Geometric location or coordinates of Sensor i;

Shared Variables:

S_i :: Sensing region of Sensor i;

$Color_i \in \{black, red\}$ :: Color of Sensor i;

Macros:

$Dst(i) \equiv$ Returns the distance of Sensor i from the center of R_Q uncovered by i sensing region (used only in Algorithm $\mathcal{DMSC}$);

uses R_Q and Pos_i to compute the distance;

$BidirN(i) \equiv \{j |\ j \in N_i \wedge i \in N_j\}$;

$RealBlackN_i :: \{j | j \in BidirN_i \wedge (Color_j = Black)\}$;

$RealBlackN_{i,t} :: \{j | j \in BidirN_i \bigcup BidirN_t \wedge (Color_j = Black)\}$;

$BlackConNbrs(i) \equiv \forall j,k \in RealBlackN_i : \exists l_1, \ldots, l_n \in RealBlackN_i : l_1 \in RealBlackN_j \wedge (\forall x < n : l_{x+1} \in RealBlackN_{l_x}) \wedge l_n \in RealBlackN_k$;

$BlackConNbrs(i,t) \equiv \forall j,k \in RealBlackN_{i,t} : \exists l_1, \ldots, l_n \in RealBlackN_{i,t} : l_1 \in RealBlackN_j \wedge (\forall x < n : l_{x+1} \in RealBlackN_{l_x}) \wedge l_n \in RealBlackN_k$;

4 Distance-Dependent Query Cover

The first solution to the connected sensor cover problem is given as Algorithm 4.1 (referred in this paper as Algorithm $\mathcal{DMSC}$). The proposed algorithm starts from the boundary sensors of R_Q, and proceeds towards the center of the region. Starting from any configuration, the algorithm selects a few sensors among many (due to our assumption of very dense network) to include in the (minimum) cover set $\mathcal{MCSC}_\mathcal{Q}$.

If the system starts from a good initial configuration, it first selects some boundary sensors, then some interior sensors, and keeps repeating the process of selection moving towards the center of the query region until it covers the whole region R_Q. Our solution is localized, meaning the decision to be selected in $\mathcal{MCSC}_\mathcal{Q}$ is taken locally by all nodes. So, unlike the solution in [9], nodes do not collect any global information to compute $\mathcal{MCSC}_\mathcal{Q}$. Sensors consult only their immediate neighbors to decide if they should be included in the final set cover.

4.1 Normal Behavior

In the following, we assume that the system starts from a good initial configuration, meaning, all sensors are *red* initially. If the system starts from a good initial configuration, it first selects some boundary sensors, then some interior sensors, and keeps repeating the process of selection moving towards the center of the query region until it covers the whole region R_Q. In the following, we will first describe how some boundary *red* sensors are selected to turn *black* to cover the boundary of R_Q. Then we discuss the general case of covering any uncovered region inside the query region.

Algorithm 4.1 Self-stabilizing Distance-Dependent Connected Sensor Cover Algorithm (Algorithm $\mathcal{DMSC}$) for Sensor i.

Predicates:

$Boundary(i) \equiv S_i$ is a boundary sensor;

$Interior(i) \equiv S_i$ is an interior sensor;

$Exterior(i) \equiv S_i \cap R_Q = \emptyset$; /* S_i is an exterior sensor; */

$Redundant(i) \equiv BlackConNbrs(i) \wedge (S_i \bigcap R_Q) \subseteq \bigcup_{j \in RealBlackN_i} S_j$;

$SameIntrsctn(i,j) \equiv Interior(i) \wedge Interior(j) \wedge (\exists x, y \in \{N_i \cap N_j\} : (Color_x = Color_y = black) : (i \in N_x \wedge i \in N_y) \wedge Pos_i \in (S_x \cap S_y) \wedge Pos_j \in (S_x \cap S_y))$;

$BestCandidate(i) \equiv \forall j \in N_i : SameIntrsctn(i,j) : (Dst(i) \leq Dst(j))$;

Actions:

$\mathcal{A}_1 :: \; Boundary(i) \wedge \neg Redundant(i) \wedge Color_i \neq black \; \longrightarrow \; Color_i = black$

$\mathcal{A}_2 :: (Redundant(i) \vee Exterior(i)) \wedge Color_i \neq red \; \longrightarrow \; Color_i = red$

$\mathcal{A}_3 :: \; BestCandidate(i) \wedge \neg Redundant(i) \wedge Color_i \neq black \; \longrightarrow \; Color_i = black$

Boundary of R_Q. The initial task is to select enough sensors to cover the boundary with a communication network of sensors. The selected sensors will be colored *black*, and the rest will remain *red*. This is implemented in Action $\mathcal{A}_1$ using the predicates *Boundary* and *Redundant*. A boundary node will turn *black* and remain *black* only if it is not a redundant node. The redundant nodes will be eventually marked *red*. The redundancy is checked by using the predicate *Redundant*, and is described in detail in the next paragraph. Action $\mathcal{A}_1$ changes a sensor from *red* to *black*. Instead of wasting energy and time, we added the redundancy check in Action $\mathcal{A}_1$ itself. As we are using an asynchronous model, some nodes may be slow in executing Action $\mathcal{A}_1$, while their neighbors have already changed to *black* by executing the same action. So, the slow nodes soon after turning to *black* may find out that they are redundant. Then, they will have to turn to *red* by executing another action ($\mathcal{A}_2$). Note that, a *red* sensor turns *black* only after checking for possible redundancy in the neighborhood.

The $Redundant(i)$ predicate checks if the query region covered by sensor i is covered by some neighboring nodes.

The above tests for redundancy are implemented by a sensor i before it decides to withdraw itself from further consideration into the set cover $\mathcal{MCSC}_\mathcal{Q}$. However, those tests only verify the coverage of i by other sensor (s). They do not implement the test of connectivity of the neighborhood of i. Recall that the set $\mathcal{MCSC}_\mathcal{Q}$ must be a connected set cover. So, before removing itself, Sensor i wants to secure the connectivity in its neighborhood. This is implemented in the predicate $BlackConNbrs(i)$ which is a part of redundancy checking. Ideally, Sensor i needs to check if every pair of its back neighbors j and k, will remain connected if i is marked redundant and removed. However, if the path between j and k contains any node $l \notin RealBlackN_i$, then i cannot verify this path because our solution is strictly localized. So, our implementation of $BlackConNbrs(i)$ verifies if j and k are connected using some intermediate nodes $l_1, \ldots, l_n$ where all the intermediate nodes are black neighbors of i.

Inside R_Q. Current *black* nodes (their creation is discussed in the next paragraph) are used to create more *black* nodes to gradually cover the uncovered region. Future *black* nodes are selected from the intersections of pairs of existing *black* nodes. The algorithm considers every intersection of two *black* nodes. It chooses one or more *red* nodes from the intersection as the new members of the cover set $\mathcal{MCSC}_\mathcal{Q}$ (using the predicate *BestCandidate*).

Note that the current *black* nodes may have been created using the boundary sensors selected earlier. Or, they were created by using the predicate *BestCandidate* among some sensors inside an intersection of two other *black* nodes.

Newly selected nodes for the cover set create a new (virtual) boundary of the uncovered query region. Thus, the algorithm reduces the uncovered portion of the query region R_Q by effectively pulling the (virtual) boundary towards the center of the query region.

4.2 Faults and Recovery

In this section, we focus on the fault handling features of the proposed algorithm (Algorithm 4.1). There are two variables used in the solution: S_i and $Color_i$ for a Sensor i. So, we need to show that our solution can cope with all possible corruptions associated with these two variables.

(1) Wrong initialization of the color variable. As discussed in Section 4.1, all sensors, if properly initialized start as *red*. *(a) Boundary Sensor.* Assume that a boundary sensor i starts in *black* color. If i is not a redundant node, then i remains *black* (see Action $\mathcal{A}_1$). That is, no correction is necessary. If i is redundant, then it will satisfy the predicate *Redundant*, hence the guard of Action $\mathcal{A}_2$. Node i then executes $\mathcal{A}_2$ and changes its color to *red*. *(b) Interior Sensor.* Assume that an interior node is initialized as a *black* colored sensor. If i is not a redundant node, then i remains *black* (see Action $\mathcal{A}_2$). So, no correction is necessary. If i is redundant, then it will satisfy the predicate *Redundant* and execute Action $\mathcal{A}_2$ which will change its color to *red*. *(c) Exterior Sensor.* All exterior sensors must be eventually colored *red*. If any exterior sensor starts as *black* initially, then it will execute Action $\mathcal{A}_2$ to change its color to *red*.

(2) Best candidate sensor's color is corrupted from *black* to *red*. Action $\mathcal{A}_3$ corrects the color back to *black*.

(3) A redundant sensor's color changes from *red* to *black*. The node, regardless of whether boundary or interior, will satisfy *Redundant* and hence, the guard of Action $\mathcal{A}_2$. So, it will change its color to *red*.

(4) Weakening or Failure of sensors, both in terms of communication and sensing ability. The weakening or failure of sensors will affect the sensing and communication range of the sensors. In other words, the constant set N and the variable S will change. Change of S may change the values of *Redundant* and *BestCandidate*. All these changes will be reflected in the change of values of the guards. So, eventually, the color of the affected node will change due to the execution of the actions. All change of colors have already been discussed in earlier cases above.

5 Distance-Independent Query Cover

In the previous section, we proposed a distance-based solution to covering a query region. In this section, we present a solution which does not need to compute the center of the query region. The proposed algorithm given as Algorithm 5.1 (referred in this paper as Algorithm $\mathcal{IMSC}$) has a different approach than Algorithm $\mathcal{DMSC}$. It chooses non-redundant nodes inside the query region. Note that in a good initial configuration (when all nodes are *red*), every node inside the query region is a potential candidate to be selected in the final cover $\mathcal{MCSC}_\mathcal{Q}$. Our algorithm removes the redundant nodes from the $\mathcal{MCSC}_\mathcal{Q}$. The two phases of the algorithm are executed based only on the local information available to the nodes.

Algorithm 5.1 Self-stabilizing Distance-Independent Connected Sensor Cover Algorithm (Algorithm $\mathcal{IMSC}$) for Sensor i.

Predicates:

$Useless(i) \equiv (BidirN_i = \emptyset)$;

$Exterior(i) \equiv (S_i \cap R_Q) = \emptyset)$;

$Redundant(i) \equiv BlackConNbrs(i) \wedge (S_i \bigcap R_Q) \subseteq \bigcup_{j \in RealBlackN_i} S_j$;

$Potential\_Candidate(i) \equiv \neg(Useless(i) \vee Exterior(i)) \wedge \neg Redundant(i)$;

$Candidate(i) \equiv Potential\_Candidate(i) \wedge (\forall j \in BidirN_i : \ Potential\_Candidate(j) :: \ i < j)$

$Potential\_Link(i,j,k) \equiv Potential\_Candidate(i) \wedge \neg Candidate(i) \wedge (\exists t \in Bidir(i), \exists j,k \in RealBlackN_i \bigcup RealBlackN_t, \neg(BlackConNbrs(i) \vee BlackConNbrs(i,t))$

$Link(i) \equiv \exists j,k, Potential\_Link(i,j,k) \wedge (\forall s : Potential\_Link(s,j,k) :: \ i < s)$

Actions:

$\mathcal{A}_1$ **:** $(Candidate(i) \vee Link(i)) \wedge Color_i \neq Black \rightarrow Color_i = Black$;

$\mathcal{A}_2$ **:** $\neg(Candidate(i) \vee Link(i)) \wedge Color_i \neq Red \rightarrow Color_i = Red$;

5.1 Normal behavior

Starting from a good configuration, all candidates or links eventually become *black* by executing $\mathcal{A}_1$. A process is a *candidate* if it is inside the query region or on the boundary, but not a redundant node. A node is *critical* if removing it disconnects at least a pair of its neighbors locally. Due to the asynchronous execution of the algorithm, all nodes may become candidates at the same time. In order to break the symmetry, only one node can be a candidate in any neighborhood. However, choosing one candidate per neighborhood may disconnect the cover. This is avoided by using *link nodes*. A node is a *potential link* node between two black nodes if it is a potential candidate but not a candidate, and the two black nodes are not neighbors of each other. That is, a potential link node works as a bridge between two black nodes that would otherwise be disconnected. A *link* node between two black nodes is the node with the minimum ID (as per the local label) among the potential link nodes of the black nodes.

5.2 Faults and Recovery

As in Algorithm 4.1, this algorithm also has two variables used in the solution: S_i and $Color_i$ for a Sensor i. So, we need to show that our solution can cope with all possible corruptions associated with these two variables.

(1) Wrong initialization of the color variable. In Algorithm $\mathcal{IMSC}$, all sensors, if properly initialized start as *red*. *(a) Interior & Boundary Sensors.* Assume that an interior or a boundary node is initialized as a *black* sensor. Let i be this node. If i is not a redundant node or it is critical, then i remains *black* (see Action $\mathcal{A}_2$). So, no correction is necessary. If i is redundant, then it executes Action $\mathcal{A}_2$ which will change its color to *red*. *(b) Exterior Sensor & Useless Sensors.* All exterior sensors must be eventually colored *red*. If any exterior sensor starts as *black*, then it will execute Action $\mathcal{A}_2$ to change its color to *red*. If a *black* sensor cannot be sensed by none of its neighbors, then it will turn *red* by executing $\mathcal{A}_2$.

(2) Candidate or link sensor's color is corrupted from *black* **to** *red*. Action $\mathcal{A}_1$ corrects the color back to *black*.

Algorithm	S.T.	CS	Algorithm	S. T.	C. S.
$\mathcal{DMSC}$ grid, range 9	41	116	$\mathcal{IMSC}$ grid, range 9	65	122
$\mathcal{DMSC}$ random, range 9	32	146	$\mathcal{IMSC}$ random, range 9	116	155
$\mathcal{DMSC}$ random, range 5-15	41	145	$\mathcal{IMSC}$ random, range 5 -15	95	154

Table 6.1: Average Performance Metrics.

(3) Non-candidate or non-links sensor's color changes from *red* **to** *black*. The node satisfies the guard of Action $\mathcal{A}_2$. So, it will change its color to *red*.
(4)Weakening or Failure of sensors, both in terms of communication and sensing ability. The algorithm behaves as Algorithm $\mathcal{DMSC}$ in this situation.

6 Complexity and Simulation

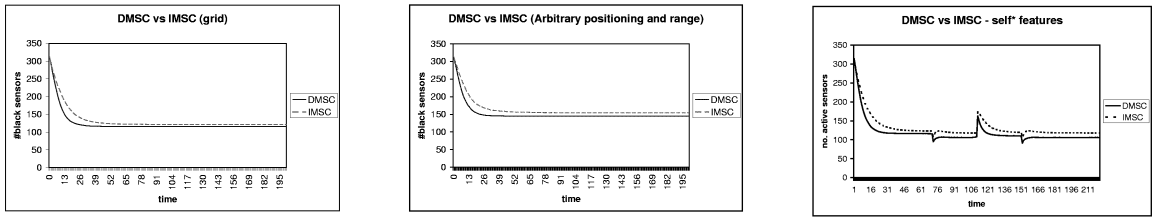
Algorithms $\mathcal{DMSC}$ and $\mathcal{IMSC}$ are space *optimal* solutions. Recall the problem specification. Upon termination of the sensor cover algorithm, a node will know if it should be active or passive. So, it must use at least two states to distinguish its two possible roles. Our solutions use exactly two states in the *Color* variable to implement this. Moreover, both solutions compute a minimal cover for the query region.

In our simulations, for the first set of experiments, we assumed that nodes are fixed and uniformly deployed on a grid of size 25×25 (625 nodes). Similar to [9, 11, 18] we consider the sensing region associated with a sensor modeled as a circular region around itself. We considered a homogeneous network (i.e. all sensors have the same sensing region — circular of radius 9). Then in the second set of experiments, we considered 625 sensors randomly deployed in a square region of size 100, with different sensing regions uniformly distributed in the range between 5 and 15. In all the simulations, we considered a circular query region of radius 40. The number of experiments performed for each simulated point was 100. The performance metrics we measured are the stabilization time (S.T.) and the number of nodes per cover (C.S.). The average results are summarized in Table 6. Our experiments for both the topologies (grid and arbitrary) show that Algorithms $\mathcal{DMSC}$ and $\mathcal{IMSC}$ have almost identical behaviors in terms of the coverage size, Algorithm $\mathcal{DMSC}$ computing a slightly better cover.

Algorithm $\mathcal{DMSC}$ stabilizes faster than Algorithm $\mathcal{IMSC}$ (see Figures 6.1 a,b,c) . The slower stabilization of Algorithm $\mathcal{IMSC}$ is due to the fact that a sensor inside the query region can change its color more than once. In Algorithm $\mathcal{DMSC}$, a sensor inside the query region waits until the covering wave arrives at its neighborhood.

Interestingly, the type of sensing regions (homogeneous or different) and the placement of sensors (grid or arbitrary) do not have a strong impact on the coverage size and stabilization time for Algorithm $\mathcal{DMSC}$. Algorithm $\mathcal{IMSC}$ seems to be more sensitive to the topology and sensors characteristics.

We observed a strong impact of the redundancy predicate on the coverage size. Due to the space limitation, the plots for this case are not shown. Initially, we assumed that a node is considered redundant only if it is covered by one, two, or three of its neighbors. Intuitively, in order to define a cover for a circle, it is sufficient to provide a triangle that includes the circle. The average size of the cover at the stabilization time was around 400 nodes for a grid placement. Note that in the grid topology, the sensing region of an arbitrary sensor cannot be covered by a triangle. The covering



(a) Grid - homogeneous sensing range 9.

(b) Arbitrary placement, heterogeneous sensing range 5-15

(c) Self* in the presence of transient and permanent faults

Figure 6.1: Experimental Results

region needs to be a square in this case. That is, we need at least four neighboring nodes to compute the redundancy predicate.

In the simulations shown in Figures 6.1 a,b,c each node uses all its neighbors to decide its redundancy. As shown in Table 6, the coverage size stabilizes to approximately 150 nodes for the both algorithms (see Table 6). Moreover, we simulate the impact of various types of faults (crash and memory corruption) on the construction of the cover computed by IMSC and DMSC (Figure 6.1c). At times 72 and 112, 30% of nodes experience crash and transient failures, respectively. Figure 6.1c shows that the systems is able to self-heal. At time 152, 30% of black nodes (nodes that form the cover) experience memory corruptions (i.e., their color change from black to red). The system self-heals again without any external intervention.

7 Future work and Conclusion

The main motivation of our research was to design a totally distributed self-$*$ query response system in sensor networks. We presented the first local, distributed, scalable, self-$*$ designs of the connected sensor cover problem introduced in [9]. We presented two stabilizing solutions to the problem and showed how the solutions are self-organizing and self-healing as well. Algorithms are space optimal — only two colors are used. Once the system is stabilized, the faults can be corrected in their neighborhood. Hence the system is self-containing. This research showed that the concept of self-stabilization subsumes many other self-$*$ properties.

The connected sensor cover problem is a global task, meaning nodes cannot locally compute the final response to the query. However, we still require the algorithm to be *local* in the sense that the nodes collect information from their immediate neighbors. Unlike the solution in [9], no node in the proposed algorithm collects global information, and no node behaves as a special node in any stage of the execution of the algorithm. In our solution, every node can locally decide if it should be an active or passive node in the current computation of the response to the query. In summary, we achieve a global objective by using local algorithms.

Sensing coverage characterizes the monitoring quality provided by a sensor network in a designated region. Different applications may require different degrees of sensing coverage. In this regard, we can extend our solutions in a couple of ways. Firstly, we may write a parametric solution where the input query will include the degree of coverage expected. The predicate *Redundancy* will be relaxed to allow the corresponding higher degree of coverage. Secondly, we can simply assume a particular degree (> 1) of coverage in our algorithm. Similar to the implementation of a higher degree of coverage to achieve better robustness, we may also require a higher degree of connectivity for the same purpose (i.e., to increase the level of fault-tolerance). We can extend the neighborhood connectivity checking to k-node ($k > 1$) disjointness in the communication graph. Unfortunately, higher degree of coverage/connectivity would require more communication cost, i.e., consuming more power. We can conduct a study on the trade off between connected cover size optimality vs. robustness and energy efficiency.

References

[1] IF Akyildiz, W Su, Y Sankarasubramaniam, and E Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, Aug 2002.

[2] A Cerpa and D Estrin. Ascent: Adaptive self-configuring sensor networks topologies. In *INFOCOM02 Proceedings of the Conference on Computer Communications*, Jun 2002.

[3] B Chen, K Jamieson, H Balakrishnan, and R Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *MobiCom02 Proceedings of the Seventh Annual Inernational Conference on Mobile Computing and Networking*, pages 85–96, Jul 2001.

[4] F. Dai and J. Wu. Distributed dominant pruning in ad hoc networks. *ICC'03*, 2003.

[5] A.K. Datta, M. Gradinariu, P. Linga, and P. Raipin-Parvey. Self* query region covery in sensor networks. Technical Report 1607, IRISA, Rennes, France (www.irisa.fr), 2004.

[6] EW Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17(11):643–644, Nov 1974.

[7] S Dolev. *Self-Stabilization*. MIT Press, 2000.

[8] D Estrin, R Govindan, J Heidemann, and S Kumar. Next century challenges: Scalable coordination in sensor networks. *Mobile Computing and Networking*, pages 263–270, 1999.

[9] H Gupta, SR Das, and Q Gu. Connected sensor cover: Self-organization of sensor networks for efficient query execution. In *MobiHoc03 Proceedings of the Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 189–200, 2003.

[10] VSA Kumar, S Arya, and H Ramesh. Hardness of set cover with intersection 1. In *ICALP00 Proceedings of the Twentyseventh International Colloquium on Automata, Languages and Programming*, pages 624–635, 2000.

[11] S Shakkottai, R Srikant, and N Shroff. Unreliable sensor grids: Coverage, connectivity and diameter. In *INFOCOM03 Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1073–1083, Apr 2003.

[12] D Tian and ND Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *WSNA02 Proceedings of the First Workshop on Sensor Networks and Applications*, pages 32–41, Sep 2002.

[13] X Wang, G Xing, Y Zhang, C Lu, R Pless, and C Gill. Integrated coverage and connectivity configuration in wireless sensor networks. In *ACM SenSys03 Proceedings of the First International Conference on Embedded Networked Sensor Systems*, pages 28–39, Nov 2003.

[14] M Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, Sep 1991.

[15] Y Xu, J Heidemann, and D Estrin. Geography-informed energy conservation for ad hoc routing. In *MobiCom02 Proceedings of the Seventh Annual Inernational Conference on Mobile Computing and Networking*, pages 70–84, 2001.

[16] F Ye, G Zhong, J Cheng, S Lu, and L Zhang. PEAS: A robust energy conserving protocol for long-lived sensor networks. In *ICDCS03 Proceedings of the 23rd International Conference on Distributed Computing Systems*, pages 1–10, 2003.

[17] H Zhang and A Arora. GS3: Scalable self-configuring and self-healing in wireless networks. In *PODC02 Proceedings of the Twentyfirst Annual ACM Symposium on Principles of Distributed Computing*, 2002.

[18] H Zhang and JC Hou. Maintaining sensing coverage and connectivity in large sensor networks. Technical Report UIUCDCS-R-2003-2351, University of Illinois at Urbana Champaign, Jun 2003.

Appendix

7.1 $\mathcal{DMSC}$ Proof of convergence

In this section, we will prove the convergence of Algorithm $\mathcal{DMSC}$ (presented in Section 4). The self* properties of Algorithm $\mathcal{DMSC}$ will be proven in Section 8. The outline of this section is as follows: We will first define a legitimacy predicate of Algorithm $\mathcal{DMSC}$ with respect to the specification of the proposed problem. Next it will be shown that the algorithm is guaranteed to arrive at a legitimate state regardless of the initial configuration or type of faults occurring in the system.

Definition 7.1 (Legitimacy Predicate $\mathcal{L}_{\mathcal{DMSC}}$). *The system is considered to be in a legitimate state (i.e., satisfies the legitimacy predicate $\mathcal{L}_{\mathcal{DMSC}}$) if the following conditions are true with respect to a query region:*
(i) All non-redundant boundary sensors are black. (ii) All non-redundant best candidate sensors are black. (iii) All other active sensors — exterior, boundary, and interior — are red.

The proof outline is as follows: We first show that starting from an arbitrary configuration, the boundary of R_Q will be covered. Then we establish the progress towards covering the whole region by proving that every *black* node creates two more *black* nodes to cover some other uncovered area of R_Q. This process is repeated until R_Q is completely covered.

Lemma 7.1. *Starting from any arbitrary configuration, the boundary of the input query R_Q will be covered.*

Proof. By contradiction. Assume that there is an area A which intersects the boundary of R_Q is not covered. A must contain at least an active sensor i. Then if Action $\mathcal{A}_1$ is enabled at i, i will turn *black*. Considering other sensors in A, A will eventually be covered. That is a contradiction. Let us assume that i is not enabled to execute Action $\mathcal{A}_1$. Then per guard of $\mathcal{A}_1$, there are two possibilities:

1. *(i)* Sensor i is *black*. So, considering other sensors (like i) in A, A is covered. That is a contradiction.
2. *(ii)* The predicate $Redundant(i)$ is true. By the definition of $Redundant(i)$, it follows that i is covered by *black* sensors. Again, considering other active sensors in A, we obtain that A is covered, hence the contradiction.

□

Lemma 7.2. *In any configuration, if all boundary and interior red nodes are redundant, then the region R_Q is completely covered.*

Proof. Assume the contradictory, i.e., although all boundary and interior *red* nodes satisfy *Redundant* predicate, R_Q is not completely covered yet. Consider an area A intersecting R_Q which is not covered. A must contain at least an active sensor i. The color of i cannot be *black* since A is assumed to be uncovered. So, i is red. Since A is not covered, i will not satisfy $Redundant(i)$. That contradicts the lemma hypothesis. □

Lemma 7.3. *Every black node covering a region of R_Q will eventually add two neighboring black nodes unless the new nodes are found to be redundant.*

Proof. 1. *(i)* Consider a *black* boundary node r. That is, r covers a region on the boundary of R_Q. The existence of at least one such node is implied by Lemma 7.1. Assume that r has two *black* neighbors, r_1 and r_2. This is a valid assumption because only one *black* sensor covering a region is very unlikely. Let I_1 be the area of intersection between the sensing regions of r_1 and r. Then in I_1, there must exist a node p that satisfies $Best\_candidate(p)$. So, unless p is redundant (i.e., satisfies the predicate $Redundant(p)$), it will execute Action $\mathcal{A}_3$ to turn *black*. Similarly, let I_2 be the intersection between the sensing regions of r and r_2. So, there must be a sensor q in I_2 which will satisfy $Best\_candidate(q)$, perform $\mathcal{A}_3$, and change its color to *black* if it is not a redundant node. Note that both p and q are interior nodes.

2. *(ii)* Now, consider a *black* interior node r covering a region in R_Q. By Lemma 7.2 and the above case, a node like r exists unless R_Q is completely covered. Following the same reasoning as in the above case, we can show that r will add two more *black* nodes unless the new nodes are marked to be redundant.

□

Lemma 7.4. *Starting from an arbitrary configuration, the input query region R_Q will eventually be completely covered by black nodes.*

Proof. Assume that $Covered\_Region_i$ and $Uncovered\_Region_i$ represent the current *covered* and *yet to be covered* region of R_Q, respectively. By Lemmas 7.2 and 7.3, there must exist at least one *black* node r in $Covered\_Region_i$, which will generate two more *black* sensors. These new *black* sensors will cover some portion of $Uncovered\_Region_i$, effectively reducing the area of $Uncovered\_Region_i$. Therefore, repeated application of Lemma 7.3 progressively reduces the area of $Uncovered\_Region_i$. Since $Uncovered\_Region_i$ is finite, eventually the system reaches a configuration which satisfies one of the following two conditions:

1. *(i)* $Uncovered\_Region_i$ becomes an empty set. That is, R_Q is completely covered.

2. *(ii)* $Uncovered\_Region_i$ is nonempty. By the lemma hypothesis, there are some *black* nodes in $Covered\_Region_i$. By Lemma 7.3, a *black* node will create two more *black* nodes. If the newly created nodes are redundant, then by Lemma 7.2, R_Q is already covered. But, that is a contradiction.

□

Theorem 7.1 (Convergence). *Starting from an arbitrary configuration, Algorithm $\mathcal{DMSC}$ reaches a configuration that satisfies the legitimacy predicate $\mathcal{L}_{\mathcal{DMSC}}$.*

Proof. By Lemma 7.4, R_Q will be eventually covered. Starting from this configuration, we now prove that the system will reach a configuration satisfying $\mathcal{L}_{\mathcal{DMSC}}$. In the following, we will consider the three conditions to be satisfied to satisfy $\mathcal{L}_{\mathcal{DMSC}}$.

(i) *All non-redundant boundary sensors are black.*
The proof follows from Action $\mathcal{A}_1$.

(ii) *All non-redundant best candidate sensors are black.*
The proof follows from Action $\mathcal{A}_3$

(iii) *All other active sensors — exterior, boundary, and interior — are red.*
Exterior nodes will turn *red* by applying Action $\mathcal{A}_2$. Other nodes if redundant will turn *red* by executing Action $\mathcal{A}_2$. The nodes which are not best candidates (if not already redundant) will eventually become redundant when they will be covered by *black* nodes, and will turn *red* (if not already *red*).

□

7.2 $\mathcal{IMSC}$ Proof of Convergence

The legitimate configuration definition for Algorithm $\mathcal{IMSC}$ is based on the notion of candidate.

Definition 7.2 (Legitimacy Predicate $\mathcal{L}_{\mathcal{IMSC}}$). *The system is considered to be in a legitimate state (i.e., satisfies the legitimacy predicate $\mathcal{L}_{\mathcal{IMSC}}$) if the following conditions are true with respect to a query region:*
(i) All candidate and link sensors are black.
(ii) All other active sensors — exterior, boundary, and interior — are red.

In the following, we prove that starting in an arbitrary configuration, the algorithm converges to a legitimate configuration satisfying Definition 7.2.

Lemma 7.5. *Starting from an arbitrary configuration, each node changes its color a finite number of times.*

Proof. Any exterior node or useless node executes at most one step. If the node is black then it turns red, otherwise it does not change its color.

A red interior node may execute first $\mathcal{A}_1$ and turns black then it may execute $\mathcal{A}_2$ and turns again red. In the following we show that the previous scenario can be repeated only a finite number of times. Such a scenario is possible if the node is part of a cycle that includes oscillating nodes. Assume the contrary. There is no cycle of oscillating nodes that contains the node. Hence we can construct a linear chain formed only with oscillating nodes. This chain will have at least one node outside the query region or at least one inside node that eventually will not oscillate. After a finite number of steps the chain eventually stabilizes. That is, if the oscillating node is part of an acyclic structure then after a finite number of steps the chain stabilizes.

Let $p_1 \dots p_l \dots p_1$ be the oscillating cycle. The cycle oscillates only if: all nodes execute rule $\mathcal{A}_1$, then all of them execute $\mathcal{A}_2$ and so on. This scenario is not possible because of the Candidate predicate. In a neighborhood only one node has its predicate verified, hence the symmetry is broken.

□

Lemma 7.6. *Starting from an arbitrary configuration, eventually, the input query region will be completely covered by black nodes.*

Proof. Let c be the initial illegitimate configuration. Each node in the query region executes a finite number of times Algorithm $\mathcal{IMSC}$ (Lemma 7.5). Hence there exists c' a configuration such that no node in c' is able to execute its algorithm. Assume that such configuration does not exist. Hence a node executes forever its algorithm. Let c'' be the configuration where all the nodes but one, p, are disabled.

Let *Cover* be the query region covered by black nodes in c''. If p is candidate or link node colored *red* in c'' . then p executes rule $\mathcal{A}_1$ and colors itself black. That is, after the p execution the query region covering is completed.

If p is a redundant node and not critical then p executes $\mathcal{A}_2$ and colors itself in red. Again, after the p's execution the query region covering is completed. □

Theorem 7.2 (Convergence). *Starting from an arbitrary configuration, Algorithm $\mathcal{IMSC}$ reaches a configuration that satisfies the legitimacy predicate $\mathcal{L}_{\mathcal{IMSC}}$.*

Proof. By Lemma 7.6, R_Q is eventually covered. In the following, we will consider the two conditions to be satisfied to satisfy $\mathcal{L}_{\mathcal{IMSC}}$.

(i) *All non-redundant boundary and interior sensors are black.*
The proof follows from Action $\mathcal{A}_1$.

(ii) *All other active sensors — exterior, boundary, and interior — are red.*
Exterior nodes will turn *red* by applying Action $\mathcal{A}_2$. Other nodes if redundant will turn *red* by executing Action $\mathcal{A}_2$.

□

8 Self-∗ Proofs

We want to conclude the proof of Algorithms $\mathcal{DMSC}$ and $\mathcal{IMSC}$ in this section by showing how our solutions satisfy some of the self-∗ properties. Algorithms $\mathcal{DMSC}$ and $\mathcal{IMSC}$ are distributed, self-configuring, self-healing, and scalable. Thus, the proposed self-∗ solution makes the goal of ubiquitous/pervasive computing a reality since two of the main requirements for this type of large ubiquitous sensor networks are low-power and self-configuring.

The next two lemmas prove that the two proposed algorithms satisfy the coverage and connectivity properties of the connected cover problem. These two results will be used later to prove some self-∗ properties.

Lemma 8.1 (Coverage). *In any legitimate configuration, the connected set cover $\mathcal{MCSC}_\mathcal{Q}$ computed by Algorithm $\mathcal{DMSC}$ or Algorithm $\mathcal{IMSC}$ completely covers the query region R_Q.*

Proof. By contradiction. Assume that there is an area A which intersects R_Q is uncovered by $\mathcal{MCSC}_\mathcal{Q}$. Since the network is assumed to be densely deployed, A must contain at least an active sensor i which is obviously *red*. Then according to the predicates $\mathcal{L}_{\mathcal{DMSC}}$ or $\mathcal{L}_{\mathcal{IMSC}}$, i must be a redundant sensor. By the definition of predicate $Redundant(i)$, i must be covered by some *black* nodes. Since i was chosen to be any node in the uncovered area A, we can claim that all active *red* sensors in A are covered by some *black* sensors. Therefore, A is covered, and we arrive at the contradiction. □

Lemma 8.2 (Connectivity). *In any legitimate configuration, the connected set cover $\mathcal{MCSC}_\mathcal{Q}$ computed by Algorithm $\mathcal{DMSC}$ or Algorithm $\mathcal{IMSC}$ forms a connected graph.*

Proof. By contradiction. Assume that there exist two node-disjoint connected components in the set $\mathcal{MCSC}_\mathcal{Q}$. All active sensors initially form a connected graph. So, the only way for the set being disconnected is by marking some active sensor (say i) as redundant such that not considering i as part of the final set $\mathcal{MCSC}_\mathcal{Q}$ disconnected i's neighborhood.

However, per predicate $BlackConNbrs(i)$, i is considered to be a redundant node only after ensuring the complete bidirectional connectivity of its neighborhood. That is, if i was marked

redundant and colored *red* by Action $\mathcal{A}_2$, all neighbors of i remained connected. In other words, if the set $\mathcal{MCSC}_Q$ was connected before $\mathcal{A}_2$ was executed, it would remain connected after the execution of the action as well. We reach the contradiction. □

Self-configuring and Self-healing Algorithms $\mathcal{DMSC}$ and $\mathcal{IMSC}$ are self-configuring in the sense that starting from any initial configuration, they configure themselves to form a network topology where all sensors (members of the connected sensor cover) are able to communicate with each other either directly or indirectly. We also proved that the given query region will eventually be covered starting from any arbitrary state. Hence these algorithms are self-configuring.

The proposed algorithms are self-healing under various perturbations, such as node joins, failures (due to crash and energy loss), state corruptions, and weakening of power. If a node fails, Algorithms $\mathcal{DMSC}$ and $\mathcal{IMSC}$ heal themselves in the following manner: If it is not a redundant node, then a part of the query region R_Q may become uncovered. In that situation, a subset of its (active) neighbors will take over by executing Action $\mathcal{A}_1$ or $\mathcal{A}_3$ in Algorithm $\mathcal{DMSC}$, and $\mathcal{A}_1$ in Algorithm $\mathcal{IMSC}$. A similar process may be initiated, if necessary, when a node's power weakens to the point that it affects the node's communication ability. On the other hand, if a node joins the network (after recovering, being repaired, or being re-energized in power), it would be considered as a redundant node since the query region is already covered by the existing nodes. So, the node joining event will not change the connected set cover. Arbitrary corruptions of state variables of the nodes are also dealt with in the solution — change of *Color* variable due to faults is fixed in a very simple manner.

Self-stabilization.

Theorem 8.1 ($\mathcal{L}_{\mathcal{DMSC}}$ and $\mathcal{L}_{\mathcal{IMSC}}$ satisfy specification). *Any system configuration satisfying the legitimacy predicates $\mathcal{L}_{\mathcal{DMSC}}$ or $\mathcal{L}_{\mathcal{IMSC}}$ (per Definitions 7.1 or 7.2, respectively) satisfies the specification of the minimal*[1] *connected sensor cover problem (per Definition 3.2).*

Proof. The coverage and connectivity properties are proved in Lemmas 8.1 and 8.2, respectively. The definitions of $\mathcal{L}_{\mathcal{DMSC}}$ and $\mathcal{L}_{\mathcal{IMSC}}$ imply that in a legitimate configuration no node can be removed from the coverage sets. Therefore, the connected cover set $\mathcal{MCSC}_Q$ computed at this point are minimal. □

Lemma 8.3 (Closure). *The legitimacy predicates $\mathcal{L}_{\mathcal{DMSC}}$ and $\mathcal{L}_{\mathcal{IMSC}}$ are closed.*

Proof. In any configuration satisfying $\mathcal{L}_{\mathcal{DMSC}}$ or $\mathcal{L}_{\mathcal{IMSC}}$, all actions of Algorithm $\mathcal{DMSC}$ or Algorithm $\mathcal{IMSC}$ respectively are disabled. Therefore, the algorithms are silent, and satisfy the closure property. □

Theorem 8.2. *Algorithms $\mathcal{DMSC}$ and $\mathcal{IMSC}$ are self-stabilizing.*

Proof. The proof follows from Theorem 8.1, Lemma 8.3, and Theorems 7.1 and 7.2. □

[1] A connected cover is minimal if it does not include another connected cover