Manchester Metropolitan University

https://e-space.mmu.ac.uk

Finding the polygon hull of a network without conditions on the starting vertex

Ahcène Bounceur[1] | **Madani Bezoui[2,3]** | **Mohammad Hammoudeh[4]** |
Loïc Lagadec[5] | **Reinhardt Euler[1]**

[1]Lab-STICC UMR CNRS 6285, Université de Bretagne Occidentale, Brest, France

[2]University M'hamed Bougara of Boumerdes, Boumerdes, Algeria

[3]LaRoMad, University of Science and Technology Houari Boumediene, Bab Ezzouar, Algeria

[4]Manchester Metropolitan University, Manchester, UK

[5]ENSTA, Brest, France

Correspondence
Ahcène Bounceur, Lab-STICC UMR CNRS 6285, Université de Bretagne Occidentale, 29238 Brest, France.
Email: Ahcene.Bounceur@univ-brest.fr

Abstract

Many real-life problems arising within the fields of wireless communication, image processing, combinatorial optimization, etc, can be modeled by means of Euclidean graphs. In the case of wireless sensor networks, the overall topology of the graph is not known because sensor nodes are often randomly deployed. One of the significant problems in this field is the search for boundary nodes. This problem is important in cases such as the surveillance of an area of interest, image contour reconstruction, graph matching problems, routing or clustering data, etc. In the literature, many algorithms are proposed to solve this problem, a recent one of which is the least polar-angle connected node (LPCN) algorithm and its distributed version D-LPCN, which are both based on the concept of a polar angle visit. An inconvenience of these algorithms is the determination of the starting vertex. In effect, the point with the minimum x-coordinate is a possible starting point, but it has to be known at the beginning, which considerably increases the algorithms' complexity. In this article, we propose a new method called RRLPCN (reset and restart with least polar-angle connected node), which is based on the LPCN algorithm to find the boundary vertices of a Euclidean graph. The main idea is to start the LPCN algorithm from an arbitrary vertex, and whenever it finds a vertex with an x-coordinate smaller than that of the starting one, LPCN is reset and restarted from this new vertex. The algorithm stops as soon as it visits the same edge for the second time in the same direction. In addition to finding the boundary vertices, RRLPCN also finds the vertex with minimum x-coordinate, which is the last starting point of our algorithm. The distributed version of the proposed algorithm, called D-RRLPCN, is then applied to boundary node detection in the wireless sensor network. It has been implemented using real sensor nodes (Arduino/XBee and TelosB). The simulation results have shown our algorithm to be very effective in comparison to other algorithms.

1 | INTRODUCTION

Since the 1970s, the algorithmic search for a "*border*" (or "boundary") circumscribing the elements of a set of points in the plane has attracted many researchers. One prominent example for such a border is the *"convex hull,"* which could be

illustrated by using a minimum size string to surround a set of nails planted in a plane. In 1971, Graham[1] was the first to propose an algorithm to find the convex hull of a set of points, which is based on the scan of the point set. One year later, Jarvis[2] proposed his version based on the use of polar angles. After the work of these two pioneers, many other approaches have emerged: the Quickhull algorithm[3] in 1977, Andrew's algorithm[4] in 1979, Kallay's algorithm[5] in 1984, Chan's algorithm[6] in 1996, etc.

In some situations, the convex hull does not sufficiently reflect the geometric properties of a data set. The *"concave hull,"* called here *"polygon hull,"* is more accurate for geometric analysis. This polygon hull approach is a more advanced method designed to capture the precise shape of the data set's surface. However, the construction of such a hull is not straightforward. Most algorithms require the knowledge of a starting point, supposed to belong to the border.

The problem of finding the polygon hull of a connected Euclidean graph can be formulated as follows: Given an undirected Euclidean graph $G = (V, E)$, where $V = \{v_0, v_1, \ldots, v_{n-1}\}$ is the set of vertices of G and E its set of edges, we are looking for a closed cycle of minimum length in G such that all vertices are either on or surrounded by that cycle. This cycle defines an area which is also called a polygon hull, and it can be described by a sequence of vertices $\mathbb{B}_V$ and a set of edges $\mathbb{B}_E$ as follows:

$$\mathbb{B}_V = (v_0, v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_h) \tag{1}$$

$$\mathbb{B}_E = \{\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{h-1}, v_h\}\} \tag{2}$$

such that

$$v_0 = v_h.$$

The search for a polygon hull of a Euclidean graph arises as an essential question in many fields such as Wireless Sensor Networks (WSN), statistics, data analysis, image processing, fingerprint matching tests, biological networks, etc. Unfortunately, there exist only a few methods to solve such a problem. In recent papers,[7,8] we have proposed an algorithm called least polar-angle connected node (LPCN) and its distributed version called D-LPCN.[8] The principle of these algorithms is to start from the point with minimum x-coordinate and look for a neighbor that forms the minimum polar angle. This process is repeated until a polar angle is traversed for the second time. Despite the good performance of this algorithm and its optimality, demonstrated in the work of Lalem et al,[7] it requires an a priori knowledge of the starting vertex, which considerably increases its complexity.

In this paper, we propose a new method called reset and restart. It is an extension of the LPCN algorithm giving a new algorithm called RRLPCN, which allows to find the polygon hull of a Euclidean graph without any condition on the starting vertex. We also present a distributed version of this algorithm, called D-RRLPCN, together with an application in the field of wireless sensor networks (WSNs).

A WSN consists of autonomous sensor nodes distributed in space to cooperatively monitor physical or environmental conditions such as temperature, sound, vibration, pressure, motion, etc. Typical WSN applications may require the random deployment of sensor nodes over a large target area. Besides, military surveillance systems may also require the detection of activities around the boundaries of the area under surveillance. Thus, the system should be able to detect and identify any object entering or leaving the monitored area. Therefore, developing mechanisms to identify network boundary nodes is a significant and challenging problem.

Moreover, RRLPCN is developed and validated theoretically using the CupCarbon and Tossim simulators. The distributed version of our proposed algorithm (D-RRLPCN) is applied to the boundary node detection problem in a wireless sensor network. The simulation results show that D-RRLPCN consumes less energy compared to some existing algorithms. After validation by simulation, a real implementation is conducted using two types of wireless sensor nodes, TelosB and Arduino Xbee.

The remainder of this paper is organized as follows. Section 2 presents related work from the fields of polygon hull determination and boundary node detection in wireless sensor networks. A mathematical formulation of the problem of polygon hull finding is proposed in Section 3. In Section 4, the LPCN algorithm and the Reset and Restart method are presented. Moreover, the integration of Reset and Restart into LPCN is described. In Section 4.3 the distributed version of RRLPCN is presented. The convergence of our method is demonstrated in Section 5 in which simulation results and an implementation into real sensor nodes are shown. Section 6 briefly describes the use of our approach for two other applications, and Section 7 concludes the paper.

2 | RELATED WORK

In this section, we present a state of the art on algorithms to find convex, concave, or polygon hulls of a set of points in the plane and of Euclidean graphs as well as on algorithms to detect boundary nodes of wireless sensor networks. In this paper, we do not deal with security and mobility problems. For this, it is suggested to refer to existing methods.[9-13]

2.1 | Finding hulls of a set of points in the plane

Chaudhuri et al[14] have proposed an approach to find the concave hull of a plane point set S. This technique is called perceptual boundary extraction, and it is based on a new definition, called $s - shape$, which is simply the union of lattice cells containing all the points of S. To obtain a polygon boundary, another $r - shape$ is defined, where r is obtained from s. For points $p, q \in S$, the edge $\{p, q\}$ is selected if and only if the boundaries of the disks centered around p and q intersect at a point which is on the boundary of the union of all the disks. The $r - shape$ of S is then the union of the selected edges.

Garai et al[15] have proposed an algorithm that starts with a convex hull of the given points and reaches the final limit in two stages: division and fusion. By splitting, one or more sides of the convex hull will be removed, and new sides are added to maintain the inherent convexity. To obtain a smooth polygon border, two or more sides are merged into one.

Adriano and Santos[16] have suggested an algorithm based on nearest neighbors. It assumes that the currently selected point is related to its k nearest neighbors. Then, it selects the point forming the least polar angle with the current point just as in Jarvis' algorithm.[2] A polygon hull is not obtained for any set of points, and the value of k must be adapted to each case.

Gheibi et al[17] have presented a shape reconstruction algorithm that finds a polygon hull. The algorithm starts from the convex hull of the initial pattern and gradually "*concaves*" it to obtain a polygon output. Thus, at any step, each selected edge will be replaced by two new edges constructed so that the shape remains a polygon. This process is repeated until the stop condition is met.

Park and Oh[18] have proposed a four-step algorithm. In the first step, a set of convex hull edges is selected and a threshold value η is determined. In the second step, a decision distance between these closest internal points and the edge points is calculated. Third, if $\frac{\text{length of edge}}{\text{decision distance}} > \eta$, the search process is executed. Finally, the second and third steps are repeated until there is no more internal point to find.

The regularized geometric hull algorithm proposed by Korner et al[19] is used principally for biomedical image segmentation. It converts the points of a data set into a set of triangles. Each triangle having its maximal edge length greater than a given parameter ζ will be excluded. The value of ζ regularizes the convexity or concavity of the geometric hull.

The algorithm proposed by Methirumangalath et al[20] starts by constructing the Delaunay graph G.[21] Afterwards, it creates a priority queue of the outside edges of G in decreasing order of edge lengths. Then, it removes, repeatedly, the outside edge of the external triangle from the head of the priority queue and the current graph G but only if it satisfies the defined circle constraint and if G without the removed edge is regular. Once the outside edge is removed from G, the adjacent sides of the external triangle are added to the priority queue by maintaining the decreasing order of the edge lengths. This process is repeated until there is no possibility of removing any outside edges from the graph G.

The main idea of the algorithm of Braune et al[22] is to start from the convex hull of the entire data set. The convex hull is iteratively shrunk by replacing edges that are too long by new edges that fit the data more accurately and then to recursively split the convex hull path into two separate closed paths as soon as this process converges to one point.

2.2 | Finding boundary vertices of a Euclidean graph

To the best of our knowledge, LPCN, introduced by Bounceur et al,[7,23,24] is the only algorithm that can find the polygon hull of a Euclidean graph. This algorithm is inspired by Jarvis' algorithm designed to find the convex hull of a set of points. The main idea of this algorithm is to start from the point with minimum x-coordinate and to select a neighbor that forms the minimum polar angle. This step will be repeated by every selected point with respect to its neighbors, and the process stops as soon as a selected point is chosen a second time. Moreover, LPCN is presented in Section 3.2. A distributed version has been published in the work of Saoudi et al.[8]

2.3 | Boundary node detection in wireless sensor networks

Several solutions have been proposed to detect the boundary nodes in a wireless sensor network. These solutions can be classified into three categories.

Geometrical approach: The algorithms of this category assume that the sensor nodes know their exact location within the network.

Martincic and Schwiebert[25] have presented an algorithm that requires each node to know the positions and communication links of its two-hop neighborhood. Using this information, the node decides whether it is enveloped by a circle of other nodes. If this occurs, the node is located in the interior of the network.

The algorithm of Deogun et al[26] only requires a node to be able to determine the distances to its direct neighbors. The node selects four of its closest neighbors that are far from each other. Then, it checks whether three of these neighbors are surrounded or not.

The approach proposed by Fang et al[27] needs to know the position of the nodes. Using a Delaunay graph and local searches, holes are identified. Zhang et al[28] apply localized Voronoi polygons for which each node has to collect the positions of its direct neighbors.

The work introduced by Shirsat and Bhargava[29] only requires a node to know an anticlockwise order of its neighbors. Each node checks for empty cones and cordless paths in the connectivity graph of its two-hop neighborhood.

Luthy et al[30] have presented an algorithm that draws an exact image to decide whether the boundary of the nodes′ communication range is entirely covered by the communication ranges of its neighbors.

Sahoo et al[31] have proposed the sequential boundary node selection algorithm and the distributed boundary node selection algorithm. The first algorithm starts from the sink. Then, it uses the right-hand rule to select boundary nodes sequentially. The second algorithm defines extreme nodes as boundary nodes and then connects them to form cycles enclosing boundaries. An extreme node is defined as a node that has either maximum or minimum values in its coordinates compared to those of its one-hop neighbors.

The algorithm of Lara-Alvarez et al[32] is based on *Ircod* sensors. These sensors determine a cyclic order neighbor and choose the first neighbor in this order. Then, the right hand without crossing rule is used to determine the complete boundary.

Zhao et al[33] have proposed two distributed algorithms based on computational geometry, distributed sector cover scanning and directional walk. The first is used to identify the nodes on the hole borders and the outer boundary. The second is used to locate the coverage holes based on the boundary nodes identified with distributed sector cover scanning and which they enclose.

Saoudi et al[8] have presented a distributed version of the LPCN algorithm introduced in the work of Lalem et al.[7] In each iteration, the current boundary node chooses the nearest polar angle given by the previously chosen node and its neighbors, and the first node has to be a boundary node. This node can be automatically determined using the minimum finding algorithm. The algorithm works with any connected network, given as planar or not, and it can determine for a disconnected network all the boundaries of the different connected components.

Statistical approach: This type of algorithm tries to exploit statistical properties such as node degrees to detect boundary nodes. As long as nodes are evenly distributed, this approach works quite well since boundary nodes usually have fewer neighbors than interior nodes. However, as soon as node degrees fluctuate noticeably, most statistical approaches produce misclassifications. Besides, these algorithms often require unrealistically high average node degrees.

Prominent statistical approaches are Fekete et al[34,35] and Bi et al.[36] Fekete et al first analyze the node degree distribution in a theoretical study. Their implementation in a second work requires data being collected over the entire network to compute a histogram of node degrees. Using the histogram, they determine a threshold value by which each node can classify itself as an inner node or a boundary node. In the approach proposed by Bi et al, nodes need information only of their neighborhood localization. Each node compares its node degree with the average node degrees of its two-hop neighbors to decide whether it is on the network boundary.

Topological approach: Algorithms using this approach concentrate on information given by the connectivity graph and try to infer boundaries from its topological structure. They often require nodes to gather information about a large neighborhood or entail sophisticated algorithms with high computational cost.

Funke[37] and Funke and Klein[38] have described algorithms that construct isocontours and check whether their contours are broken. If a node detects that a contour is broken, it classifies the corresponding contour endpoints as boundary nodes. The first algorithm requires that the whole network is flooded, starting from some seed nodes. The second algorithm is distributed based on six-hop neighborhoods.

The methods proposed by Ghrist and Muhammad[39] and de Silva and Ghrist[40] detect holes by utilizing algebraic homology theory. They are centralized and rely on restrictive assumptions on the communication model.

The algorithm of Kroller et al[41] identifies complex combinatorial structures called flowers. Such flowers exist with high probability under some hypotheses on the communication model. The algorithm requires that every node knows its eight-hop neighborhood.

Wang et al[42] have introduced an algorithm that works well even in networks with low average node degree. It involves multiple steps, some of which require that the whole network is flooded.

Dong et al[43] have described a distributed algorithm based on topological transformations of the connectivity graph. It is, in particular, aimed at locating small holes.

The approach presented by Li and Hunter[44] needs to determine all pairs of nodes with overlapping sensing ranges (eg, by comparing sensing results). The graph induced by this information is used to detect holes in the network. Boundary nodes are determined by finding circles in their one-hop neighborhood. Using knowledge of their k-hop neighborhood, holes of up to $4k + 2$ ($k = 1, 2, \ldots$) hops in perimeter are found.

Dinh,[45] Schieferdecker et al[46], and Chu and Ssu[47] have introduced algorithms that exploit the same basic information. Each node constructs a graph induced by its neighbors at the exact two-hop distance. It is checked whether these graphs form closed circles. This is done by verifying the connectivity of subgraphs in Dinh's case, by tree construction and analysis in the approach of Chu and Ssu, who further provide a correctness proof for their method.

Saukh et al[48] have introduced an algorithm that tries to identify distinct patterns in the neighborhood of a node. Under specific conditions, they can guarantee that all nodes that are classified as interior nodes lie inside the network. The algorithm is distributed, and every node needs information only on its k-hop neighborhood. The radius k depends on the node density. For low density, $k = 6$ is used.

Sitanayah et al[49] have presented a heuristic based on finding boundary cycles inside a network. The principal idea of this approach is to start from a first cycle, which will be increased gradually by inserting other vertices until it hits the boundary of the network. A vertex with the highest degree will be chosen as the center of the cycle. In case that other cycles are found, they will be merged. This approach can work with low-density wireless sensor networks.

Yan et al[50] focus on detecting small coverage holes. They assume the communication range of each node to be two times its sensing range and the nodes know or can easily determine whether they are located on the outer boundary of the network. Their theoretical reasoning implies the topological Cech and Rips complexes. In their distributed implementation, each node needs to find a Hamilton cycle in its two-hop neighborhood.

In another work, Dong et al[43] also focus on discovering small coverage holes. They make the same assumptions as Yan et al[50] regarding communication ranges and periphery awareness.

Khan et al[51] devise a topology-based method for hole detection, in which the connectivity between nodes and their x-hop neighbors is used without the location of nodes.

In this paper, we propose an algorithm to find the polygon hull in a connected Euclidean graph based on LPCN, which allows to start from any point of the graph. A distributed version of the algorithm is proposed in view of its application to boundary node detection in wireless sensor networks.

3 | BASIC CONCEPTS

Before we begin the presentation of our proposed algorithm, let us give some definitions that will help the reader to better understand our method.

Let $G = (V, E)$ be a connected Euclidean graph in the plane, where V and E are, respectively, the set of vertices and edges of G. Unless otherwise specified, these definitions apply throughout this document.

3.1 | Definitions

Definition 1. A boundary vertex of G is a vertex whose deletion will modify the geometric area formed by the vertices of G.

Definition 2. If v is a boundary vertex of G, then there is no subset of V forming a polygon and containing v in its interior.
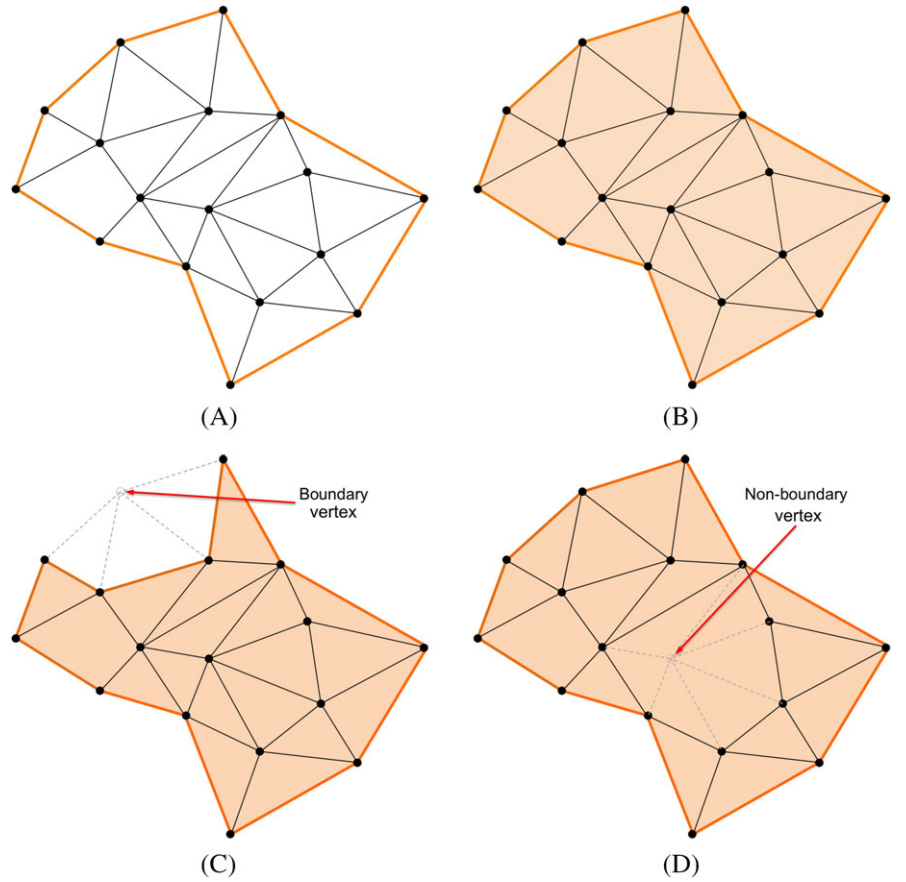
FIGURE 1 Boundary and nonboundary vertices

We can illustrate Definition 1 in Figure 1A, which shows a graph with two kinds of vertices. The boundary vertices are those that belong to the orange edges, and the other ones are nonboundary vertices. The orange area of Figure 1B shows the geometrical area formed by the graph. Figure 1C shows a case where one boundary vertex is removed. As we can see, the new shape of the graph is different from the original one. This means that the deleted vertex is a boundary vertex. However, if we remove a nonboundary vertex, as shown by Figure 1D, the new geometrical area formed by the vertices of the graph is the same as the original one, which means that the deleted vertex is a nonboundary one.

3.2 | Least polar-angle connected node Algorithm (LPCN)

In this section, we review the LPCN algorithm used to find the polygon hull of a Euclidean graph $G = (V, E)$. The algorithm starts from a vertex of G with minimum x-coordinate and determines the vertices $v_0, v_1, \ldots, v_{h-1}$ of the polygon hull. The main step of the algorithm is the selection of the neighbors of the current vertex, which forms the least-polar angle. The process stops when the angle corresponding to the starting vertex is visited twice. The LPCN pseudocode is described in Algorithm 1, where $\mathbb{B}_V$ and $\mathbb{B}_E$ are the sets of, respectively, boundary vertices and boundary edges, defined by Equation 1 and Equation 2. In this algorithm, $N(p)$ represents the set of the neighbors of the point p.

4 | BOUNDARY DETECTION WITHOUT CONDITION ON THE STARTING VERTEX

4.1 | Reset and restart technique

In this section, we describe a new method called reset and restart, which will be combined with the centralized algorithm LPCN in order to find the boundary vertices of a connected Euclidean graph by starting from any vertex. As a by-product, this method also allows to find the vertex with the minimum x-coordinate in the graph. However, if the considered system

Algorithm 1 Least polar-angle connected node (LPCN)

1: **procedure** $LPCN(V, E)$
2:     $P_0 \leftarrow P_c \leftarrow$ a point having the minimum x-coordinate
3:     $P_p \leftarrow$ a fictitious point situated to the left of $P(c)$
4:     $\mathbb{B}_V \leftarrow \{P_c\}$
5:     $\mathbb{B}_E \leftarrow \emptyset$
6:     $once \leftarrow true$
7:     **repeat**
8:         $\mathbb{A} \leftarrow \{P \in N(P_c) / \mathbb{B}_E \cap \{\{P_c, P\}\} = \emptyset\}$
9:         $P_{\min} \leftarrow \underset{P \in \mathbb{A}}{\operatorname{argmin}} \{\varphi(P_p, P_c, P)\}$
10:        $\mathbb{B}_V \leftarrow \mathbb{B}_V \cup \{P_{\min}\}$
11:       $\mathbb{B}_E \leftarrow \mathbb{B}_E \cup \{\{P_c, P_{\min}\}\}$
12:       $P_p \leftarrow P_c$
13:       $P_c \leftarrow P_{\min}$
14:       **if** $(once = true)$ **then**
15:          $once \leftarrow false$
16:          $P_{\text{first}} \leftarrow P_{\min}$
17:       **end if**
18:     **until** $((P_c = P_0) \ and \ (P_{\min} = P_{\text{first}}))$
19:     **return** $\mathbb{B}_V, \mathbb{B}_E$
20: **end procedure**

is distributed, then the reset and restart concept must be combined with the D-LPCN algorithm. In this paper, we assume that the nodes/vertices of the graph are not mobile. However, the mobility is considered as future work.

To illustrate the Reset and restart technique, let us take the graph shown in Figure 2, where the vertex with minimum x-coordinate is linked by a chain to the starting vertex. Suppose that the algorithm starts from vertex 1 as shown by Figure 2A. First, we will mark it and set the value of $x_{\min} = x_1$, the x-coordinate of vertex 1, and go to vertex 2, as shown by Figure 2B and select it. Since the x-coordinate of vertex 2 is greater than $x_{\min}$, we select vertex 3 and compare its x-coordinate to $x_{\min}$ (see Figure 2C). Again, the x-coordinate of vertex 3 is greater than $x_{\min}$. Now, vertex 4 is selected (see Figure 2D). Since the x-coordinate of vertex 4 is less than $x_{\min}$, "Reset" is launched, which consists first in unselecting all the selected vertices (Figure 2E). After that, "Restart" executes LPCN with the last selected vertex 4 and updates $x_{\min}$ to the value x_4, the x-coordinate of vertex 4 (Figure 2F). The, by executing LPCN, we select vertex 5 (Figure 2G). Again, since the x-coordinate of vertex 5 is less than $x_{\min}$ (Figure 2H), we will execute the reset and restart process (Figure 2I). Now, if we run LPCN from vertex 5, we will find the polygon hull of the graph because the algorithm will be run until it selects vertex 5 for the second time, which is the stop condition of the algorithm.

4.2 | Reset and restart with least polar-angle connected node (RRLPCN)

The major disadvantage of LPCN is the need for an a priori knowledge of the vertex of G with minimum x-coordinate. Unfortunately, this information is often unknown, and its search increases the complexity of the algorithm. This is the main motivation for the proposed improvement of LPCN using the reset and restart technique. The complexity of the proposed algorithm is the same as that of the LPCN algorithm repeated each time the starting process is executed. This means that $O(RRLPCN) = w \times O(LPCN) = w \times O(dn_b + n)$, where n is the number of nodes, n_b is the number of the boundary nodes, and d is the maximum degree of the network/graph.

The main steps of the new algorithm are described as follows.

1. In the *Restart* step, the starting vertex will select that vertex, which forms the minimum polar angle between the starting vertex, a fictitious vertex, which is an imaginary vertex situated on its left and its neighbors, as shown by Figure 3. In this figure, the edge between the starting vertex and the selected one is colored green.
2. After the selection of the next vertex, we have two possible cases. Either the x-coordinate of this vertex is greater or equal to $x_{\min}$ or it is less than $x_{\min}$.

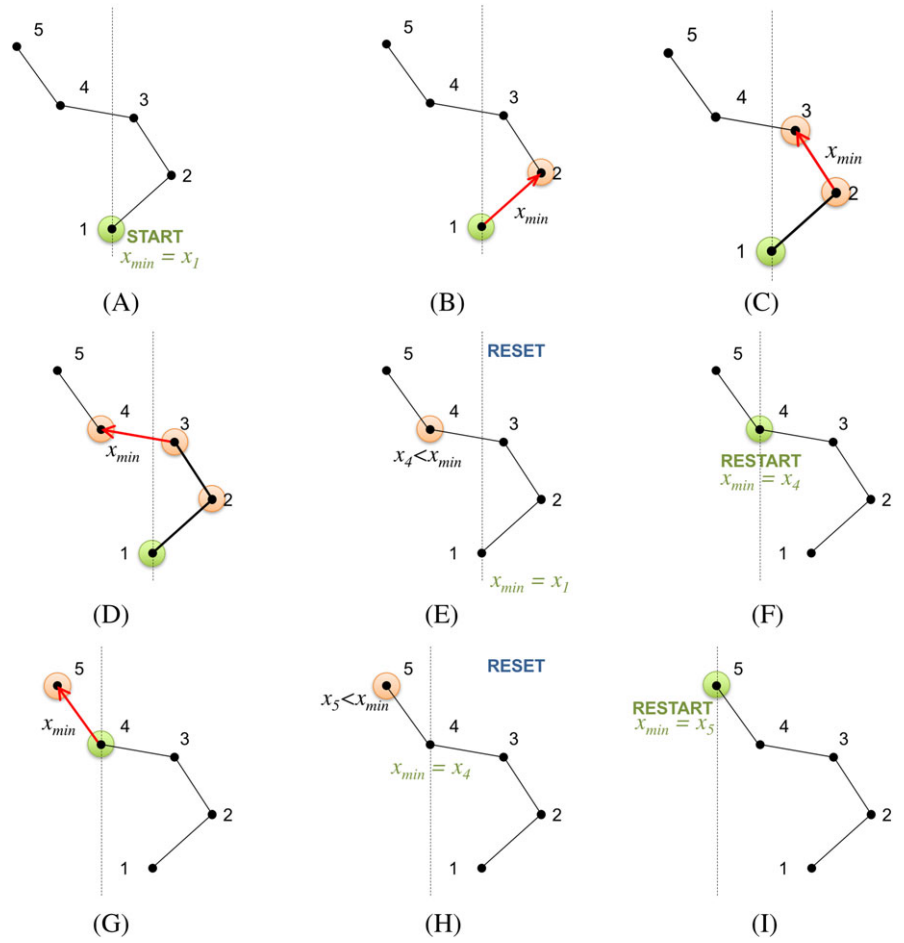   (a) In the first case, we continue the execution of LPCN and select a subsequent vertex.
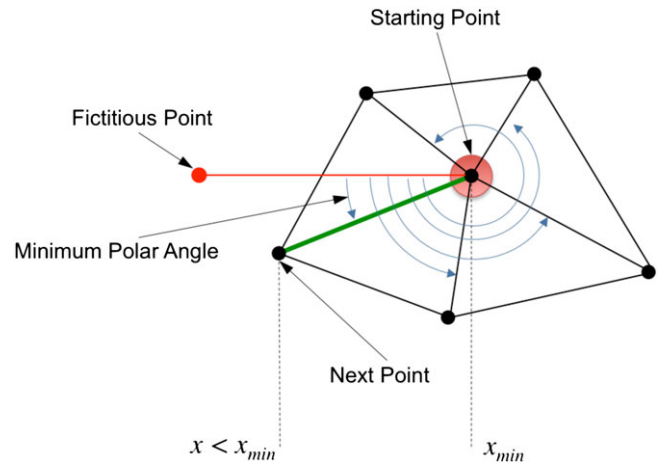
FIGURE 2 *Reset and restart* technique

FIGURE 3 Minimum polar angle

(b) In the second case, we run *Reset* and set $x_{\min}$ equal to the current vertex x-coordinate and we *Restart* the algorithm from this vertex.

Figure 4 shows an example where all the x-coordinates of the selected vertices are greater than $x_{\min}$.

We can conclude that finding the boundary vertices matches with finding a vertex with minimum x-coordinate. Thus, Algorithm 2 shows the pseudocode of RRLPCN to find the boundary vertices by starting from any vertex of a network. Moreover, a flowchart is presented in Figure 5, which resumes the main steps of the introduced method.
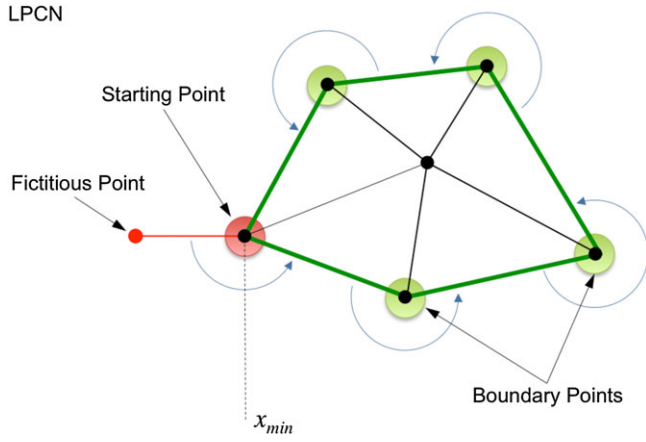
FIGURE 4   An example for polygon hull finding

Algorithm 2 Reset and restart least polar-angle connected node (RRLPCN)

1: **procedure** $RRLPCN(V, E)$
2:     $P_c \leftarrow$ an arbitrary vertex chosen from V.
3:     $\mathbb{B}_V \leftarrow [P_c]$
4:     $P_{\text{first}} \leftarrow P_c$
5:     $P_p \leftarrow$ a fictious vertex situated to the left of P_f
6:     $X_{\text{first}} \leftarrow$ the x-coordinate of P_{first}
7:     **repeat**
8:         $\mathbb{A} \leftarrow \emptyset$
9:         $P_v \leftarrow \underset{P_j \in N(P_c) \ \& \ P_j \notin \mathbb{A}}{\operatorname{argmin}} \{\varphi(P_p, P_c, P_j)\}$
10:        $X_v \leftarrow$ the x-coordinate of P_v
11:        **if** $X_v < X_{\text{first}}$ **then**
12:            *Reset*: $\mathbb{B}_V \leftarrow \emptyset$; $\mathbb{B}_E \leftarrow \emptyset$
13:            *Restart*: $P_c \leftarrow P_v$; go to step 3;
14:        **end if**
15:        **if** $\mathbb{B}_\mathbb{E} \cap \{P_c, P_v\} \neq \emptyset$ **then**
16:            $\mathbb{A} \leftarrow \mathbb{A} \cup \{P_v\}$
17:            Go to 9
18:        **end if**
19:        $\mathbb{B}_V \leftarrow \mathbb{B}_V \cup \{P_v\}$
20:        $\mathbb{B}_E \leftarrow \mathbb{B}_E \cup \{P_c, P_v\}$
21:        $P_p \leftarrow P_c$
22:        $P_c \leftarrow P_v$
23:     **until** $P_v = P_f$
24:     **return** $\mathbb{B}_V, \mathbb{B}_E$
25: **end procedure**

Figure 6 shows the execution of Algorithm 2 on a graph of 10 vertices.

The algorithm starts from the red vertex (see part (A)), which is set as a starting vertex and with an x-coordinate equal to $x_{\min}$. Then, it computes the next vertex among its neighbors, which is the neighbor forming the minimum polar angle between a fictitious vertex, itself (the red vertex) and its neighbors. In our example, the next vertex is the green vertex (see part (B)). Since the x-coordinate of the green vertex is less than the x-coordinate of the red vertex, the red vertex is *Reset*; the new starting vertex is the green vertex whose x-coordinate is set to the new $x_{\min}$ (see part (C)). In the same way, the new starting vertex computes the next vertex (see part (D)). Since the x-coordinate of the next vertex is less than $x_{\min}$, the next vertex is the new starting vertex, its x-coordinate is set as $x_{\min}$, and the previously selected vertex is *Reset* (part (F)). Using the same principle, the new starting vertex *Restart* the process and computes the next vertex since, this time, the x-coordinate of the next vertex is greater than $x_{\min}$. This process continues without *Restart*, until reaching the
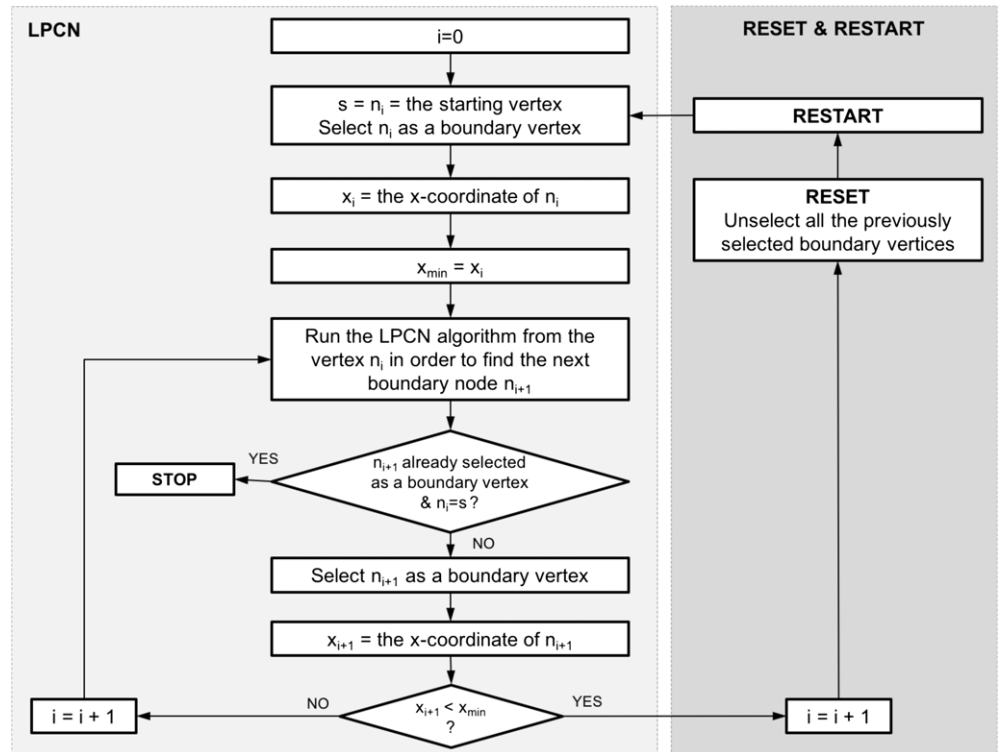
FIGURE 5 A flowchart of reset and restart combined with the least polar-angle connected node algorithm (RRLPCN)
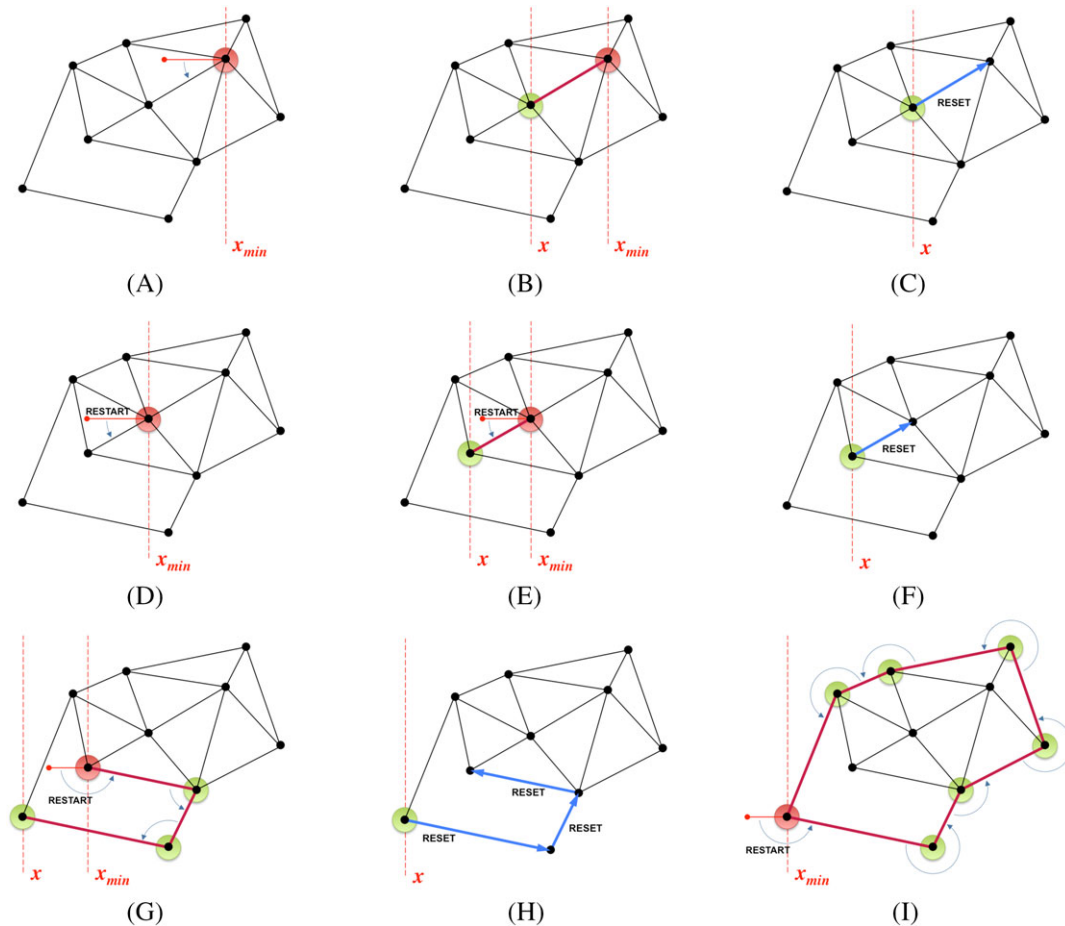

FIGURE 6 An example for reset and restart combined with the least polar-angle connected node algorithm

Message	Description
AC	Request coordinates
CS	Send coordinates
SN	Select a node
DN	Reset node

TABLE 1 Different messages and their description

Function	Description
getId()	Returns the node identifier
getCoord()	Returns the node coordinates (x, y)
getNumberOfNeighbors()	Returns $out - nbrs_i$, the number of neighbors of the node
send(m, adr)	Sends a message m to the node having address adr, if $adr = *$ it sends a message to all *outgoing neighbors*
reset()	Reset node

TABLE 2 Functions used by D-RRLPCN

most left green vertex (part (G)). Since the x-coordinate of this vertex is less than $x_{\min}$, all previously selected vertices (part (H)) are *Reset*, and the process is *Restart* from this vertex. The algorithm selects all boundary vertices until arriving at the starting vertex, which computes the next vertex as the same vertex selected in the previous round, and the stop condition is satisfied, ie, all the boundary vertices are selected (part (I)).

4.3 | Reset and restart combined with the distributed least polar-angle connected node algorithm (D-RRLPCN)

Boundary node detection is an essential problem in wireless sensor network applications. It is necessary for keeping track of the coverage range and events entering or leaving the region and for tracking communication with the external environment. It can be used for extracting further information about the structure and robustness of the network, which is employed for routing, controlling, and management purposes.

Suppose we have a wireless sensor network with all nodes deployed randomly. There can be two kinds of nodes: those that are on the boundary and the interior ones. We assume that each node can get its localization within the deployment zone. Many existing methods can be used for this information.[52-54]

In order to define a wireless sensor network formally, we model it as a Euclidean graph $G = (V, E)$. We use the letter n to denote $|V|$, the number of nodes in the network. For each node i of G, we use the notation $nbrs_i$ to denote the *"number of neighbors"* of i, that is, those nodes located in the communication range of i.

We also define a message alphabet, described in Table 1, and the functions used in the algorithm in Table 2.

Algorithm 3 exhibits the main steps performed by each sensor node for the identification of the boundary nodes of a wireless sensor network. These steps can be explained by the flowchart of Figure 7. In this case, the nodes are completely independent, and the x-coordinate of the starting node is sent to the next boundary node in each iteration. If a node receives an x-coordinate that is smaller than its own x-coordinate, then this node will send to the previously found boundary nodes a message to reset them as nonboundary nodes. Then, it will restart the process of finding the boundary nodes.

5 | VALIDATION OF THE PROPOSED ALGORITHMS

5.1 | Analytical validation

In order to validate the proposed algorithm regarding optimality and convergence, we will present, in the following, some mathematical concepts that will allow us to define the final version of the proposed algorithm.

Proposition 1. *After the application of LPCN starting from a nonboundary vertex $v_1 \in V$, with X_{v_1} its x-coordinate and the degree of v_1 being equal or greater than one, the subgraph visited by LPCN is a cycle containing at least one vertex $v^0 \in V$ with*

$$X_{v^0} < X_{v_1}, \tag{3}$$

where X_{v^0} is the x-coordinate of v_0.

Algorithm 3 Distributed reset and restart least polar-angle connected node (D-RRLPCN)

```
 1: boundary = false;
 2: once = true;
 3: cid = getId();
 4: cx = getX();
 5: cy = getY();
 6: i = 0;
 7: n = getNumberOfNeighbors();
 8: nid = 0;
 9: nid_ref = -1;
10: phi_min = 10;
11: first = id of the starting node;
12: prevList=();
13: repeat
14:     if (cid==first and once) then
15:         boundary = true;
16:         once = false;
17:         xmin = cx;
18:         px = cx-1;
19:         py = cy;
20:         data = cid+"|"+"AC";
21:         send(data, *);
22:     end if
23:     id = read();
24:     type = read();
25:     if (type=="AC") then
26:         data = cid+"|"+"CS"+"|"+cx+"|"+cy;
27:         send(data, id);
28:     end if
29:     if (type=="CS") then
30:         id = read();
31:         nx = read();
32:         ny = read();
33:         phi = angle(px, py, cx, cy, nx, ny);
34:         if (phi<phi_min) then
35:             phi_min = phi;
36:             nid = id;
37:         end if
38:         i = i + 1;
39:         if (i==n) then
40:             if (nid==nid_ref and cid==first) then
41:                 STOP
42:             else
43:                 if (nid_ref<0) then
44:                     nid_ref = nid
45:                 end if
46:             end if
47:             i = 0;
48:             data = cid+"|"+"SN"+"'"|"+cx+"|"+cy+"|"+xmin;
49:             send(data, nid);
50:         end if
51:     end if
```

```
52:     if (type=="SN") then
53:         boundary = true;
54:         n = getNumberOfNeighbors();
55:         nid = 0;
56:         phi_min = 10;
57:         px = read();
58:         py = read();
59:         xmin = read();
60:         if (cx<xmin) then
61:             delay(t)
62:             once = true;
63:             first = cid;
64:             data = cid+"|"+"DN"+"|"+first;
65:             send(data, id);
66:         else
67:             prevList.add(id);
68:             data = cid+"|"+"AC";
69:             send(data, *);
70:         end if
71:     end if
```

```
72:     if (type=="DN") then
73:         boundary = false;
74:         first = read();
75:         if (prevList.size()>=0) then
76:             data = cid+"|"+"DN"+"|"+first;
77:             previous = prevList.getLast();
78:             prevList.removeLast();
79:             if previous.hasPrevious()) then
80:                 send(data, previous);
81:             end if
82:         end if
83:     end if
84: until false
```

Proof. Suppose the opposite, ie, v_1 is a nonboundary vertex and there exists no vertex visited by LPCN that has a smaller x-coordinate. This implies that X_{v_1} is the minimal x-coordinate of the resulting subgraph G'. However, G' is a boundary of G and $v_1 \in G'$, which means that v_1 is a boundary vertex. This is a contradiction to the fact that v_1 is a non-boundary vertex. □

Theorem 1. *Except for the finally calculated angle of the starting point, the RRLPCN algorithm never calculates the same polar angle more than once.*

Proof. We assume that the algorithm starts from vertex P_{first} and that, at iteration i, we can, without loss of generality, represent the current set of boundary nodes $\mathbb{B}_V^i = \{P_{\text{first}}, \ldots, P_i\}$ by a vertex *PiBV*. On the other hand, we assume that angles are calculated only in anticlockwise order. Hence, any angle, as given by two specific edges, will be calculated only once. Otherwise, if an angle is calculated a second time, then the algorithm must pass a second time through the vertex *PiBV*. This means that we pass a second time through the starting vertex P_{first}, which represents the stop condition of LPCN (see line 12 of Algorithm 2). This leads to a contradiction, and therefore, it is not possible to calculate an angle more than once. □

Theorem 2. *The RRLPCN algorithm never runs the Restart Step (step-14) from the same point.*
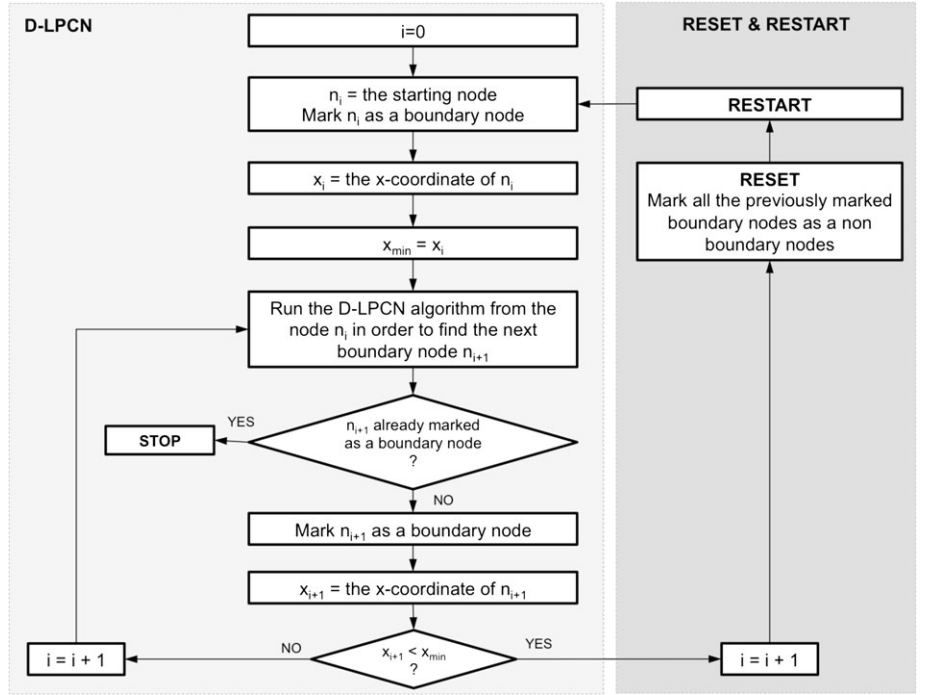
FIGURE 7 A flowchart of reset and restart combined with the distributed least polar-angle connected node algorithm

Proof. Assume that at iteration k we will start *RRLPCN* with a vertex P^k that has already been visited. Let $\mathbb{B}_V^{k-1}$ be the set of the obtained vertices. Since the algorithm restarts from the same vertex, $X_P^k = \underset{v \in \mathbb{B}_V^{k-1}}{argmin}\ X_v$, which means that p_k is a boundary vertex according to Theorem 1. Which is the stop condition of LPCN. □

Corollary 1. *The RRLPCN algorithm converges in a finite number of steps.*

Theorem 3 (See the work of Lalem et al[7]). *The polygon hull found by LPCN is of minimum cardinality.*

Lemma 1. *The polygon hull found by RRLPCN is of minimum cardinality.*

Proof. When the algorithm executes the last iteration, it stops and finishes with a vertex v_f (stop condition satisfied). This means that there is no point $v_i \in V$ such that $x_{v_i} < x_{v_f}$. In other words, $\forall v_i \in V\ :\ x_{v_i} < x_{v_f}$; consequently, v_f is a vertex with minimum x-coordinate in the graph. Starting LPCN from this vertex will give us the polygon hull of the graph according to Theorem 3. □

5.2 | Application to wireless sensor networks

5.2.1 | Simulation results

To validate and test our distributed algorithm, we have used the two simulation environments CupCarbon[55] and TinyOS[56] Tossim.[57] The first one allowed us to develop our protocol and to make debugging, and the second one was used to test the final version and to analyze the performances.

CupCarbon[55] is a Smart City and Internet of Things simulator. Its objective is to design, visualize, debug, and validate distributed algorithms for monitoring, environmental data collection and to create ecological scenarios, generally within educational and scientific projects. CupCarbon proposes two simulation environments. The first is a multiagent environment, which enables the creation of the mobility scenarios and the generation of events such as fires and gas as well as the simulation of mobiles such as vehicles and flying objects.[58] The second simulation environment uses the discrete event simulation of wireless sensor networks taking into account the scenario created by the first environment.

Here, TOSSIM, the TinyOS Simulator, uses its hierarchical model by replacing lower level hardware components with software-emulated ones. This approach reduces the hole between the simulator and the real environment. By substituting low-level components, a high fidelity between reality and the simulation environment is achieved. One of the advantages
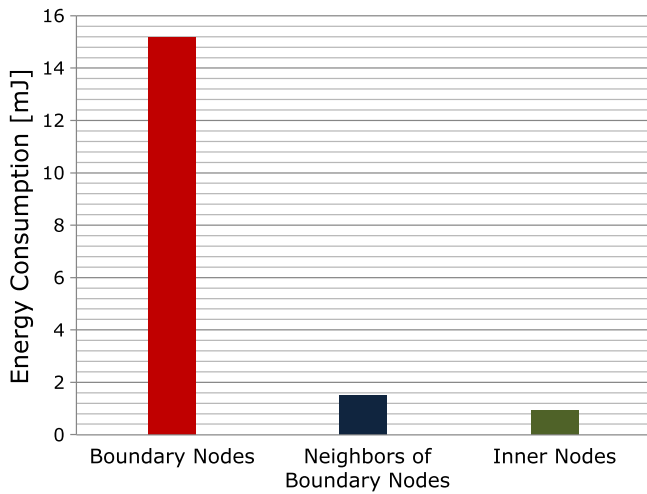
FIGURE 8 Energy consumption according to sensor node positions in reset and restart combined with the distributed least polar-angle connected node algorithm
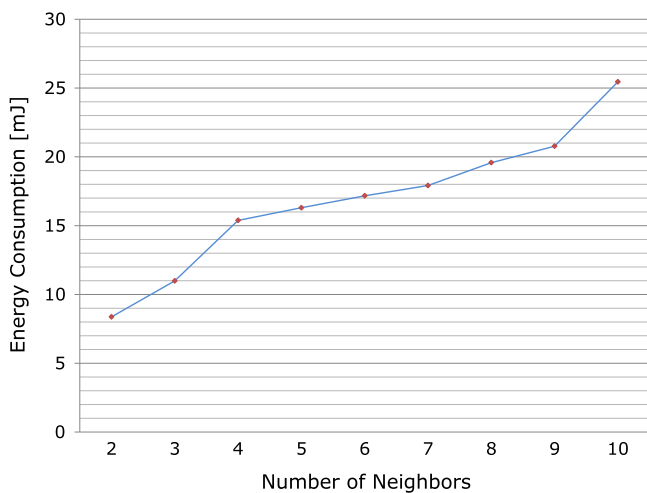

FIGURE 9 Energy consumption of boundary nodes according to the number of neighbors in reset and restart combined with the distributed least polar-angle connected node algorithm

of the simulation with TOSSIM is the reuse of the code since the code used for the simulation is the same that will be exported to the real sensor nodes.

Energy efficiency is one of the most critical factors for WSN applications. The lower the energy consumed by each node, the longer the network can perform its mission. For this reason, our simulations are based on this measure of performance. We calculate the average of energy consumed by each node in millijoule (mJ) according to simulation parameters.

Figure 8 represents the average consumption of nodes in mJ according to their position (boundary nodes, neighbors of boundary nodes, and inner nodes). The purpose of this histogram is to compare the consumption of the nodes in the network according to their positions. As can be noted, the boundary nodes consume more energy because they are the ones that participate most in the algorithm, followed by the neighbors of boundary nodes, and finally the inner nodes.

As we have seen in Figure 8, the boundary nodes consume more energy in comparison to other nodes. In Figure 9, the consumption of these nodes according to their number of neighbors is examined. There is a correlation between the energy consumption of these nodes and their number of neighbors. The more the number of neighbors increases, the more they consume energy. One of the solutions to reduce this consumption is the reduction of the network density.

It is possible to not activate all network nodes at the same time, which will increase the lifetime of the network and reduce the consumption of the boundary nodes.

In order to prove the efficiency of D-RRLPCN regarding energy consumption, we have compared it with D-LPCN and three other algorithms (D-LPCN, min finding, and D-LPCN+min finding).

Figure 10 shows a comparison histogram of the average of energy consumption of nodes depending on their position in the network (boundary nodes, neighbors of boundary nodes, and inner nodes) for the two algorithms, D-RRLPCN and D-LPCN. The graph allows us to deduce that whatever the position of the node in the network, D-RRLPCN consumes less energy than D-LPCN.

FIGURE 10 Comparison of distributed least-polar angle connected node (D-LPCN) and reset and restart combined with the distributed least polar-angle connected node algorithm (D-RRLPCN) with respect to energy consumption according to sensor node positions
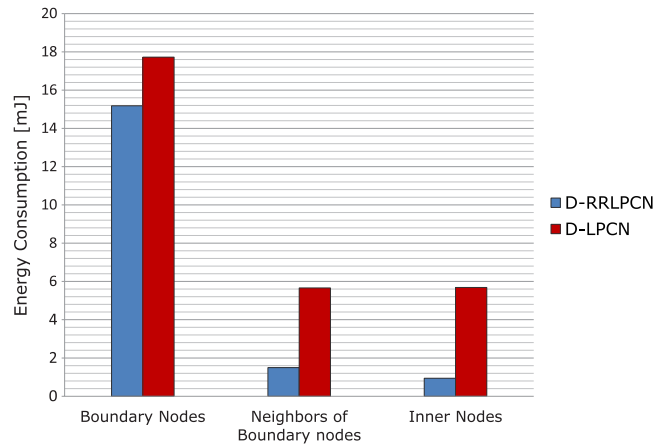

FIGURE 11 Comparison of the distributed least-polar angle connected node (D-LPCN), min finding, D-LPCN+min finding, and reset and restart combined with the distributed least polar-angle connected node algorithm (D-RRLPCN) with respect to energy consumption of boundary node according to number of neighbors
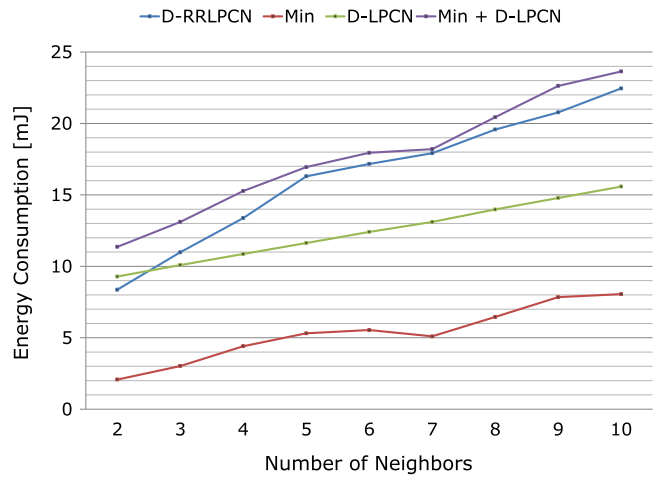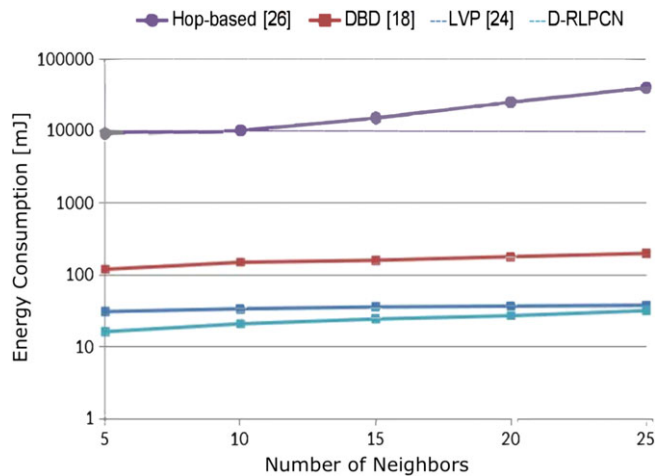

FIGURE 12 Comparison of reset and restart combined with the distributed least polar-angle connected node algorithm (D-RRLPCN), hop-based, distributed boundary detection (DBD), and located Voronoï polygons (LVP) with respect to energy consumption of boundary nodes according to number of neighbors

The D-LPCN algorithm consists of two phases. The first phase allows to find the starting node which is the node with minimum *x*-coordinate (in this paper we call this part "min finding"). The second phase allows to start from the starting sensor node found in the first phase to obtain a boundary of the network.

Figure 11 shows the energy consumption of the boundary nodes depending on their number of neighbors, for D-RRLPCN, the min finding, D-LPCN without the first part (min finding), and the last graph representing the sum of the two phases of D-LPCN. Considering the two phases of D-LPCN, D-RRLPCN is more efficient regardless of the number of neighbors.
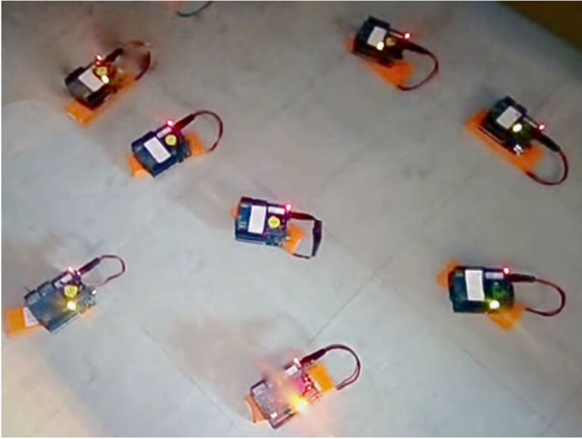
FIGURE 13 Execution of reset and restart combined with the distributed least polar-angle connected node algorithm using real Arduino/XBee sensor nodes

In Figure 12, a comparison with existing methods is given. We have compared the energy consumption of three algorithms, namely hop-based,[59] distributed boundary detection (DBD),[50] and located Voronoï polygons (LVP)[60] with that of D-RRLPCN. The energy consumption of boundary nodes is calculated according to their number of neighbors.

5.2.2 | Implementation in real sensor nodes

Building a wireless sensor network system needs development and integration of many hardware and software components. Figure 13 shows the overall architecture of the wireless sensor network system that we have developed. The system includes a number of distributed wireless sensor nodes, where each sensor node is a combination of a microcontroller, an XBee radio transceiver, and a battery.

→ *Design of Sensor Node*

In this research, we have studied sensor nodes using Arduino and Digi XBee modules. Arduino is an open-source computer hardware and software company, project, and user community that designs and produces devices for building digital applications and interactive objects that can sense and control the physical world. These devices are based on a family of microcontroller board designs produced primarily by Smart Projects in Italy and by numerous other vendors, using different 8-bit Atmel AVR microcontrollers or 32-bit Atmel ARM processors. In this work, we have used Arduino Uno R3 based on the Atmel Atmega328 microcontroller.

For wireless communication, we have used the XBee module from Digi. Xbee is basically used for this purpose. It offers the IEEE 802.15.4 connectivity in the 2.4 GHz ISM band. There is a "pro" version, which provides large communication range, denoted as XBP24, and a "normal" version, denoted as XB24. The term "series 1" is employed for the 802.15.4 version and "series 2" for the ZigBee version. The XBee module uses a UART (serial interface) to communicate with the main board. The advantage is simplicity and the possibility to reuse many serial tools. For our implementation, we have used the Xbee series 1. The indoor communication range of this module is 30 m, whereas the outdoor range is nearly 100 m. With low power consumption and data rates of up to 250 kbps, Xbee devices are particularly suitable for fast prototyping of wireless sensor network applications. It is possible to build a simple star-structured network or a complex mesh network using these devices. Table 3 summarizes the technical characteristics of the used sensor nodes.

TABLE 3 Technical characteristics of sensor nodes

Devices	Specifications	Values
Arduino UNO R3	Microcontroller	ATmega328P
	Flash memory	32 KB
	Clock speed	16 MHz
	SRAM	2 KB
	EEPROM	1 KB
Xbee XB24	Frequency band	2.4 GHz
	RF data rate	250 kbps
	Indoor/Urban range	30 m
	Outdoor/RF line-of-sight range	100 m
	Transmit current	45 mA
	Receive current	50 mA
Battery	Capacity	2200 mAh

Figure 13 shows a wireless sensor network with eight Arduino/Xbee sensor nodes deployed randomly. The sensor nodes that have a green led turned on are the boundary nodes detected by the D-RRLPCN algorithm.

6 | OTHER APPLICATIONS

There are several other problems and domains where we can apply the algorithms proposed in this paper (RRLPCN and D-RRLPCN). We can cite two examples.

Drawing object contours in images:
The algorithm combined with an algorithm to characterize the selected zone can be used to draw object contours in images. The user has to click on a pixel of the zone to study, and starting from this point, RRLPCN can detect the contour of the selected zone (object). This technique can be applied to identify the shape of a tumor, for example. Moreover, RRLPCN can be very useful in the medical field. Indeed, detecting the boundary of a tumor with high precision is a very crucial task for the medical practitioner. With RRLPCN, this task can be done by selecting any pixel of the radio representing this tumor.

Defining the boundary of a two-dimensional multirobot system:
A multirobot system can be described as a set of robots operating in the same environment. In these systems, we can use a boundary as a formal practical definition of what is inside and outside the network. Knowing the boundary would allow us to estimate the perimeter of the configuration. For a surveillance application,[61] robots on the boundary can specialize in target monitoring and notify the network when a target has entered or left the tracking area. At each moment, one of the robots can launch the D-RRLPCN algorithm, and the boundary robots will be detected.

7 | CONCLUSION

In this article, we have proposed a new technique called reset and restart, which, in combination with LPCN and D-LPCN, allowed us to introduce two new algorithms: RRLPCN and D-RRLPCN. Both algorithms now find the polygon hull of a Euclidean graph without any assumption on the starting node where the second is the distributed version of the first and particularly suited for wireless sensor networks. We have also validated the proposed method theoretically and computationally.

The D-RRLPCN algorithm has been developed and simulated using the two simulation environments, CupCarbon and TOSSIM. The results show that it consumes less energy than D-LPCN and hop-based, DBD, and LVP. The algorithm has been implemented in two real sensor platforms, TelosB and Arduino Xbee. For future work, we suggest an extension of the RRLPCN algorithm to 3D space and a proposition of a distributed version for boundary node detection in 3D wireless sensor networks.

ORCID

Ahcène Bounceur https://orcid.org/0000-0002-0043-7742
Mohammad Hammoudeh https://orcid.org/0000-0003-1058-0996

REFERENCES

1. Graham RL. An efficient algorithm for determining the convex hull of a finite planar set. *Inf Process Lett*. 1972;1(4):132-133.
2. Jarvis RA. On the identification of the convex hull of a finite set of points in the plane. *Inf Process Lett*. 1973;2(1):18-21.
3. Eddy WF. A new convex hull algorithm for planar sets. *ACM Trans Math Softw*. 1977;3(4):398-403.
4. Andrew AM. Another efficient algorithm for convex hulls in two dimensions. *Inf Process Lett*. 1979;9(5):216-219.
5. Kallay M. The complexity of incremental convex hull algorithms in $\mathbb{R}^d$. *Inf Process Lett*. 1984;19(4):197.
6. Chan TM. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete Comput Geom*. 1996;16(4):361-368.
7. Lalem F, Bounceur A, Bezoui M, et al. LPCN: least polar-angle connected node algorithm to find a polygon hull in a connected Euclidean graph. *J Netw Comput Appl*. 2017;93:38-50.

8. Saoudi M, Lalem F, Bounceur A, et al. D-LPCN: a distributed least polar-angle connected node algorithm for finding the boundary of a wireless sensor network. *Ad Hoc Netw*. 2017;56:56-71.

9. Baker T, Aldawsari B, Tawfik H, Reid D, Ngoko Y. GreeDi: an energy efficient routing algorithm for big data on cloud. *Ad Hoc Netw*. 2015;35:83-96.

10. Baker T, Mackay M, Shaheed A, Aldawsari B. Security-oriented cloud platform for SOA-based SCADA. In: Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2015); 2015; Shenzhen, China.

11. Ghafir I, Saleem J, Hammoudeh M, et al. Security threats to critical infrastructure: the human factor. *J Supercomput*. 2018;74(10):4986-5002.

12. Hussain AJ, Marcinonyte DM, Iqbal F, Tawfik H, Baker T, Al-Jumeily D. Smart home systems security. In: Proceedings of the 20th IEEE International Conference on High Performance Computing and Communications; 16th IEEE International Conference on Smart City; 4th IEEE International Conference on Data Science and Systems (HPCC/SmartCity/DSS 2018); 2018; Exeter, UK.

13. Aldabbas O, Abuarqoub A, Hammoudeh M, Raza U, Bounceur A. Unmanned ground vehicle for data collection in wireless sensor networks: mobility-aware sink selection. *Open Autom Control Syst J*. 2016;8:35-46.

14. Chaudhuri AR, Chaudhuri BB, Parui SK. A novel approach to computation of the shape of a dot pattern and extraction of its perceptual border. *Comput Vis Image Underst*. 1997;68(3):257-275.

15. Garai G, Chaudhuri BB. A split and merge procedure for polygonal border detection of dot pattern. *Image Vis Comput*. 1999;17(1):75-82.

16. Moreira A, Santos MY. Concave hull: a k-nearest neighbours approach for the computation of the region occupied by a set of points. In: Proceedings of the International Conference on Computer Graphics Theory and Applications; 2007; Barcelona, Spain.

17. Gheibi A, Davoodi M, Javad A, et al. Polygonal shape reconstruction in the plane. *IET Comput Vis*. 2011;5(2):97-106.

18. Park J-S, Oh S-J. A new concave hull algorithm and concaveness measure for *n*-dimensional datasets. *J Inf Sci Eng*. 2012;28(3):587-600.

19. Körner M, Krishna MV, Süsse H, Ortmann W, Denzler J. Regularized geometric hulls for bio-medical image segmentation. *Ann BMVA*. 2015:1-12.

20. Methirumangalath S, Parakkat AD, Muthuganapathy R. A unified approach towards reconstruction of a planar point set. *Comput Graph*. 2015;51:90-97.

21. Toussaint GT. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*. 1980;12(4):261-268.

22. Braune C, Dankel M, Kruse R. Obtaining shape descriptors from a concave hull-based clustering algorithm. In: Proceedings of the International Symposium on Intelligent Data Analysis; 2016; Stockholm, Sweden.

23. Bounceur A, Bezoui M, Euler R. *Boundaries and Hulls of Euclidean Graphs: From Theory to Practice*. Boca Raton, FL: CRC Press, Taylor & Francis; 2018.

24. Bounceur A, Euler R, Benzerbadj A, et al. Finding a polygon hull in wireless sensor networks. In: Proceedings of the European Conference on Operational Research (EURO); 2015; Glasgow, UK.

25. Martincic F, Schwiebert L. Distributed perimeter detection in wireless sensor networks. 2004. http://newslab.cs.wayne.edu/perimeter.pdf

26. Deogun JS, Das S, Hamza HS, Goddard S. An algorithm for boundary discovery in wireless sensor networks. In: Proceedings of the International Conference on High-Performance Computing; 2005; Goa, India.

27. Fang Q, Gao J, Guibas LJ. Locating and bypassing holes in sensor networks. *Mob Netw Appl*. 2006;11(2):187-200.

28. Zhang C, Zhang Y, Fang Y. Localized algorithms for coverage boundary detection in wireless sensor networks. *Wireless Networks*. 2009;15(1):3-20.

29. Shirsat A, Bhargava B. Local geometric algorithm for hole boundary detection in sensor networks. *Secur Commun Netw*. 2011;4(9):1003-1012.

30. Luthy K, Grant E, Deshpande N, Henderson TC. Perimeter detection in wireless sensor networks. *Robot Auton Syst*. 2012;60(2):266-277.

31. Sahoo PK, Sheu J-P, Hsieh K-Y. Target tracking and boundary node selection algorithms of wireless sensor networks for internet services. *Information Sciences*. 2013;230:21-38.

32. Lara-Alvarez C, Flores JJ, Wang C-C. Detecting the boundary of sensor networks from limited cyclic information. *Int J Distributed Sens Netw*. 2015;11(7):401-438.

33. Zhao L-H, Liu W, Lei H, Zhang R, Tan Q. Detecting boundary nodes and coverage holes in wireless sensor networks. *Mob Inf Syst*. 2016;2016:1-16.

34. Fekete SP, Kaufmann M, Kröller A, Lehmann K. A new approach for boundary recognition in geometric sensor networks. arXiv preprint CS/0508006. 2005.

35. Fekete SP, Kröller A, Pfisterer D, Fischer S, Buschmann C. Neighborhood-based topology recognition in sensor networks. In: Proceedings of the International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics; 2004; Turku, Finland.

36. Bi K, Tu K, Gu N, Dong WL, Liu X. Topological hole detection in sensor networks with cooperative neighbors. In: Proceedings of the 2006 International Conference on Systems and Networks Communications (ICSNC'06); 2006; Tahiti, French Polynesia.

37. Funke S. Topological hole detection in wireless sensor networks and its applications. In: Proceedings of the 2005 Joint Workshop on Foundations of Mobile Computing (DIALM-POMC '05); 2005; Cologne, Germany.

38. Funke S, Klein C. Hole detection or: "How much geometry hides in connectivity?" In: Proceedings of the 22nd Annual Symposium on Computational Geometry (SCG '06); 2006; Sedona, AZ.

39. Ghrist R, Muhammad A. Coverage and hole-detection in sensor networks via homology. In: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN '05); 2005; Los Angeles, CA.

40. de Silva V, Ghrist R. Coordinate-free coverage in sensor networks with controlled boundaries via homology. *Int J Robotics Res*. 2006;25(12):1205-1222.

41. Kröller A, Fekete S, Pfisterer D, Fischer S. Deterministic boundary recognition and topology extraction for large sensor networks. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm (SODA '06); 2006; Miami, FL.

42. Wang Y, Gao J, Mitchell JSB. Boundary recognition in sensor networks by topological methods. In: Proceedings of the 12th Annual International Conference on Mobile Computing and Networking (MobiCom '06); 2006; Los Angeles, CA.

43. Dong D, Liao X, Liu K, Liu Y, Xu W. Distributed coverage in wireless ad hoc and sensor networks by topological graph approaches. *IEEE Trans Comput*. 2012;61(10):1417-1428.

44. Li X, Hunter DK. Distributed coordinate-free algorithm for full sensing coverage. *Int J Sens Netw*. 2009;5(3):153-163.

45. Dinh TL. Topological boundary detection in wireless sensor networks. *J Inf Process Syst*. 2009;5(3):145-150.

46. Schieferdecker D, Völker M, Wagner D. Efficient algorithms for distributed detection of holes and boundaries in wireless networks. In: Proceedings of the International Symposium on Experimental Algorithms; 2011; Crete, Greece.

47. Chu W-C, Ssu K-F. Decentralized boundary detection without location information in wireless sensor networks. In: Proceedings of the Wireless Communications and Networking Conference (WCNC 2012); 2012; Paris, France.

48. Saukh O, Sauter R, Gauger M, Marrón PJ. On boundary recognition without location information in wireless sensor networks. *ACM Trans Sens Netw*. 2010;6(3). Article No. 20.

49. Sitanayah L, Datta A, Cardell-Oliver R. Heuristic algorithm for finding boundary cycles in location-free low density wireless sensor networks. *Computer Networks*. 2010;54(10):1630-1645.

50. Yan F, Martins P, Decreusefond L. Connectivity-based distributed coverage hole detection in wireless sensor networks. In: Proceedings of the Global Telecommunications Conference (GLOBECOM 2011); 2011; Houston, TX.

51. Khan IM, Jabeur N, Zeadally S. Hop-based approach for holes and boundary detection in wireless sensor networks. *IET Wirel Sens Syst*. 2012;2(4):328-337.

52. Langendoen K, Reijers N. Distributed localization in wireless sensor networks: a quantitative comparison. *Computer Networks*. 2003;43(4):499-518.

53. Mao G. *Localization Algorithms and Strategies for Wireless Sensor Networks: Monitoring and Surveillance Techniques for Target Tracking: Monitoring and Surveillance Techniques for Target Tracking*. Hershey, PA: IGI Global; 2009.

54. Zhang X, Tepedelenlioglu C, Banavar M, Spanias A. Node localization in wireless sensor networks. *Synth Lect Commun*. 2016;9(1):1-62.

55. Mehdi K, Lounis M, Bounceur A, Kechadi T. CupCarbon: a multi-agent and discrete event wireless sensor network design and simulation tool. In: Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques; 2014; Lisbon, Portugal.

56. Levis P, Madden S, Polastre J, et al. TinyOS: an operating system for sensor networks. In: *Ambient Intelligence*. Berlin, Germany: Springer; 2005:115-148.

57. Levis P, Lee N, Welsh M, Culler D. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03); 2003; New York, NY.

58. Lounis M, Mehdi K, Bounceur A. A CupCarbon tool for simulating destructive insect movements. In: Proceedings of the 1st IEEE International Conference on Information and Communication Technologies for Disaster Management (ICT-DM'14); 2014; Algiers, Algeria.

59. Hammoudeh M, Al-Fayez F, Lloyd H, et al. A wireless sensor network border monitoring system: deployment issues and routing protocols. *IEEE Sens J*. 2017;17(8):2572-2582.

60. Abdelkader K. *Méthodes Analytiques Pour la Couverture dans un Réseau de Capteurs sans Fil* [PhD thesis]. Béjaia, Algeria: Université Abderrahmane Mira de Béjaia; 2010.

61. Benzerbadj A, Bouabdellah K, Bounceur A, Hammoudeh M. Surveillance of sensitive fenced areas using duty-cycled wireless sensor networks with asymmetrical links. *J Netw Comput Appl*. 2018;112:41-52.