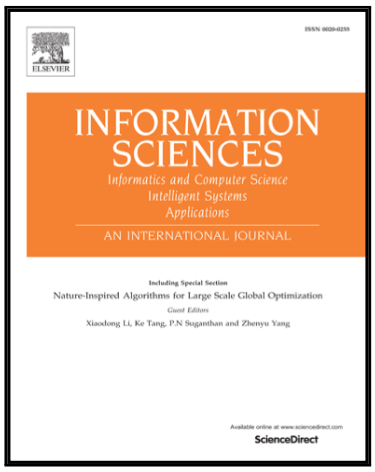


An Efficient Genetic Algorithm for Maximizing Area Coverage in Wireless Sensor Networks

Nguyen Thi Hanh, Huynh Thi Thanh Binh, Nguyen Xuan Hoai, Marimuthu Swami Palaniswami

PII: S0020-0255(19)30182-3
DOI: https://doi.org/10.1016/j.ins.2019.02.059
Reference: INS 14331

To appear in: *Information Sciences*

Received date: 10 August 2018
Revised date: 22 February 2019
Accepted date: 23 February 2019

Please cite this article as: Nguyen Thi Hanh, Huynh Thi Thanh Binh, Nguyen Xuan Hoai, Marimuthu Swami Palaniswami, An Efficient Genetic Algorithm for Maximizing Area Coverage in Wireless Sensor Networks, *Information Sciences* (2019), doi: https://doi.org/10.1016/j.ins.2019.02.059

An Efficient Genetic Algorithm for Maximizing Area Coverage in Wireless Sensor Networks

Nguyen Thi Hanh[a,b], Huynh Thi Thanh Binh[a,*], Nguyen Xuan Hoai[c], Marimuthu Swami Palaniswami[d]

[a]*Hanoi University of Science and Technology, Vietnam*
[b]*Phuong Dong University, Vietnam*
[c]*AI Academy, Vietnam*
[d]*University of Melbourne, Australia*

Abstract

Wireless Sensor Networks collect and transfer environmental data from a predefined region to a base station to be processed and analyzed. A major problem when designing these networks is deploying sensors such that their area coverage is maximized. Given a number of sensors with heterogeneous sensing ranges, the problem of coverage maximization is known to be NP-hard. As such, prevailing methods often rely on metaheuristic techniques while employing approximated fitness functions, resulting in modest solution quality and stability. This paper proposes a novel and efficient metaheuristic in the form of a genetic algorithm, which overcomes several weaknesses of existing metaheuristics, along with an exact method for calculating the fitness function for this problem. The proposed genetic algorithm includes a heuristic population initialization procedure, the proposed exact integral area calculation for the fitness function, and a combination of Laplace Crossover and Arithmetic Crossover Method operators. Experiments have been conducted to compare the proposed algorithm with five state-of-the-art methods on a wide range of problem instances. The results show that our algorithm delivers the best performance in terms of solution quality and stability on a majority of the tested instances.

Keywords:
`Area Coverage Problem, Genetic Algorithm, Integral Calculation, Wireless Sensor Networks, Monte Carlo.`

*Corresponding author. Huynh Thi Thanh Binh, Hanoi University of Science and Technology, Vietnam.
Email addresses: `hanhnt@dhpd.edu.vn` (Nguyen Thi Hanh), `binhht@soict.hust.edu.vn` (Huynh Thi Thanh Binh), `nxhoai@aiacademy.edu.vn` (Nguyen Xuan Hoai), `palani@unimelb.edu.au` (Marimuthu Swami Palaniswami)

1. Introduction

Advancements in wireless technology have had broad impacts over the last few decades, including but not limited to the improvement of the communication range and scalability of wireless networks. Among branches of wireless networks, Wireless Sensor Networks (WSNs) have received significant attention from the academic community due to their wide range of applications, particularly in Internet of Things (IoT) technology. As a result, communication in large scale WSNs has seen significant improvements in recent years, enabling their deployment in regions of complex terrain or hard-to-reach areas.

WSNs usually consist of one or more base stations and a set of sensor nodes tracking changes in environmental factors such as temperature, water salinity and humidity. Each sensor has a known sensing range, within which environmental data is continuously collected. This data is then sent either directly or via temporary nodes to the base station within the limits of the sensor's communication range [21, 26].

Information collected by sensors is useful for anticipating and giving warnings about unusual occurrences within the monitoring area. Therefore, WSNs support a variety of applications in environmental studies, healthcare, military, industry, agriculture, and IoT [3, 19, 15, 4, 23, 8, 2, 22].

However, the limited range of sensor nodes is considered to be one of the primary weaknesses of WSNs, resulting in unmonitored areas in the surveillance region. For the sake of greater coverage, there is a tendency to increase the number of sensor nodes, which requires higher infrastructure costs. Practically, sensors with different sensing ranges can be deployed in the same network to adapt to the specific size and shape of the surveillance region and optimize the cost of deployment. Thus, some existing models that assume a fixed sensing range for all sensors may not address the requirements of real life applications.

Researchers have recently approached the problem of maximizing area coverage in WSNs with multiple sensing ranges in different ways, e.g. applying particle swarm optimization algorithms and evolutionary algorithms. This problem can be stated concisely as follows: Given a number of sensors with heterogeneous sensing ranges, how can we deploy the sensors in a specific region such that the total area coverage is maximized?

The WSN coverage problem was first formulated and proven to be NP-Hard by Yourim Yoon et al. [25] who concurrently proposed four genetic algorithms for its solution, known as PGA, MGA, OPTGA, and OPTHGA. The solutions obtained by these algorithms were highly promising with OPTHGA providing the best results among the four. However, these algorithms have high computational complexity. To overcome this drawback, Dinh Thi Ha Ly et al. [14] proposed an algorithm called IGA, which replaced the Monte Carlo technique in [25] with a new technique, as well as introducing a new heuristic and dynamic mutation operator. This improvement led to a great reduction in computational complexity and an increase in solution quality in comparison with OPTHGA [14].

However, as in [11] Nguyen Thi Hanh et. al. pointed out that IGA has a slow rate of convergence. To improve the convergence speed, the authors proposed the Democratic Particle Swarm Optimization (DPSO) algorithm which was proven to converge nearly five times faster than IGA while obtaining solutions of similar quality. In [6], Improved Cuckoo

Search (ICS) and Chaotic Flower Pollination Algorithm (CFPA) were proposed by Huynh Thi Thanh Binh et. al. to achieve even better computational time.

However, we have noticed that the performance of the algorithms in [25, 14, 11, 6] are to some extent unstable. In particular, their results were evaluated by the approximate area coverage that sensors cast on the surveillance region. This area coverage varied greatly in different runs and sometimes even exceeded the theoretical upper bound. We assume that the problems leading to this performance instability and unreliability lie within the fitness function calculation procedure. Therefore, in this paper, we propose a novel genetic algorithm called MIGA to overcome the aforementioned weaknesses. In particular, the new contributions of our genetic algorithm are as follows:

- A heuristic algorithm for population initialization.
- A more stable and reliable fitness function whose computed area coverage does not exceed the theoretical upper bound.
- A combination of two different crossover operators.
- A local search using Virtual Force Algorithm (VFA) for further improvement of the final results.

The rest of the paper is organized as follows. We formulate the problem in Section 2. Section 3 covers related work. We describe our proposed algorithms in Section 4. Section 5 provides extensive experimental results. Finally, Section 6 concludes the paper and gives insight for future works.

2. Problem Formulation

This paper considers the problem of maximizing the area coverage in heterogeneous WSNs as formulated in [25]. Initially, a two dimensional surveillance region A sized W x H and a fixed number of sensors are given. There are k different types of sensors in the whole sensor node set, in which a sensor type i possesses a sensing range r_i. The goal of this problem is to find an optimal placement scheme for the set of sensor nodes so that the area coverage cast on A, denoted coA, is maximized. Formally, this problem can be stated as follows:

Input:

- W, H: the respective width and height of the $2D$ surveillance region A.
- k: the number of sensor types.
- n: the number of sensors.

- n_i: the number of sensors for type i $(i = 1, 2, ..., k)$, such that:

$$\sum_{i=1}^{k} n_i = n \tag{1}$$

- r_i: the sensing radius of sensor type i $(i = 1, 2, ..., k)$

Output: The placement scheme for each sensor node.

Objective: Maximize the area coverage of n sensors cast on A (denoted coA):

$$coA = area\left(\left(\bigcup_{i=1}^{k}\bigcup_{j=1}^{n_i} c_{r_i}\left(x_{i_j}, y_{i_j}\right)\right)\bigcap A\right) \rightarrow \max \tag{2}$$

with $c_{r_i}\left(x_{i_j}, y_{i_j}\right)$ as the circle centered at $\left(x_{i_j}, y_{i_j}\right)$ with radius r_i; and $area(X)$ as the area of region X.

In Expression (2), $\left(\bigcup_{i=1}^{k}\bigcup_{j=1}^{n_i} c_{r_i}\left(x_{i_j}, y_{i_j}\right)\right)$ refers to the total area coverage by all n sensors. Since we only care about coverage in the surveillance region A, the effective area coverage is the intersection of the total area coverage with A. The objective is to maximize this effective area coverage.

In this paper, we use the Boolean disk coverage model [25]. We denote $d\left(s, x\right)$ as the Euclidean distance between a sensor s and a point x, and the constant r_s as the sensing range of s. The coverage function of the model is given by

$$f\left(d(s, x)\right) = \begin{cases} 1, \text{if } d\left(s, x\right) \leq r_s \\ 0, \text{otherwise} \end{cases} \tag{3}$$

The next section presents an overview of works related to this problem.

3. Related Work

Research works in [19, 25] suggest that the coverage in WSNs plays an important part in evaluating the quality of such systems. However, the optimization objectives in terms of coverage vary depending on different applications of wireless sensor networks. Based on surveys on this topic in [19, 10, 16, 12], the coverage problems in WSNs can be divided into three categories corresponding to one or more specific applications of WSNs, namely, area coverage, target coverage, and barrier coverage. This paper focuses on solving the area coverage problem.

This problem was first tackled in 2002 [12] by using a potential-field based approach where each sensor node is pushed away by both obstacles and other sensor nodes, forcing the network to spread itself throughout the surveillance region. This process executes repeatedly until no further improvement of the nodes' placement is achieved, assuring the optimality of the final solution.

Another formulation for the coverage maximization problem was later introduced in [16] and [23]. Given a $2D$ surveillance region A, the goal of this model is to find the smallest number of sensor nodes that offer total area coverage of A. To tackle this problem, two algorithms were proposed including deterministic node placement and random node deployment.

However, in practice, the area coverage of sensor nodes strictly depends on the manufacturers. Therefore, considering only one type of sensor node is impractical when working on such problems. Taking this argument into consideration, a new model for the area coverage problem was introduced by Yourim Yoon et al. in [25]. This research investigated sensor nodes having heterogeneous sensing ranges and aimed at finding an optimal placement scheme for a fixed number of sensors. The objective for this model is to have the largest area covered by the sensor node set. In particular, the sensing coverage of each sensor node is represented by a perfect circular disk centered at its location; each possessing a radius equal to the sensing range of the corresponding sensor type. This problem formulation was proven to be an NP-hard combinatorial optimization problem, thus it is usually tackled by using heuristics and metaheuristics. Genetic algorithms (GA) are a family of metaheuristic models inspired by evolution [20]. Recently, the authors in [5], [1] proposed continuous genetic algorithms to solve systems of singular boundary value problems (BVPs) and proved their convergence and stability. Yourim Yoon et al. in [25] applied GA to solve the coverage maximization problem with four different versions of a genetic algorithm called PGA, MGA, OPTGA and OPTHGA. In these GAs, an individual is encoded as an array of sensor node coordinates; each member of the array represents the corresponding sensor's location in the surveillance region A.

To evaluate the quality (fitness) of individuals, the area coverage cast on A (CoA) by the encoded sensor nodes is measured using the Monte Carlo sampling method. This technique scatters L randomly generated points into the surveillance region A where the approximate area coverage is calculated as the ratio between the points covered by the sensor node set and L. Because L is usually very large (for reliable Monte Carlo sampling based estimation) while the number of sensors n is relatively small, Monte Carlo sampling has a high complexity of $O(nL)$ with $L >> n$. Therefore, the proposed genetic algorithms perform poorly in terms of computational time despite producing good results. Experiments showed that, the calculation time on large problem instances took up to 45 minutes [19].

In order to improve the quality of solutions and reduce the computational time of OTPHGA, Dinh Thi Ha Ly et al. [14] proposed an improved genetic algorithm called IGA that replaces Monte Carlo sampling with a new fitness function, whose complexity is $O(n^2)$. In their work, a new concept of $Olap$ was introduced, representing the approximate overlapping area between sensor nodes or between a sensor node and the surveillance region's boundaries. Additionally, to improve the solution quality, they replaced the static Gaussian mutation applied in OTPHGA with a dynamic one and implemented a new heuristic initialization to substitute the random scheme proposed in [25]. Experimental results showed that IGA achieved better computational time compared to that in [25] while also improving solution quality by approximately 0.5%.

However, it has been pointed out in [11] that IGA delivers relatively slow convergence

speed when compared to the Particle Swarm Optimization (PSO) and Democratic PSO (DPSO) algorithms. Overall, PSO offered solutions with better quality but required longer execution time than IGA. Meanwhile, DPSO executed much faster while producing area coverage that is about 1% smaller than IGA. Experimental results showed that DPSO outperformed IGA in terms of convergence speed, leading to a significant improvement in computational time. Specifically, while DPSO converges quickly at around the 50^{th} generation, the convergence speed of IGA is steady and fairly slow [11].

More recently in [6], two new metaheuristics, Improved Cuckoo Search (ICS) and Chaotic Flower Pollination Algorithm (CFPA), were proposed to improve solution quality using less computational time. ICS achieved relatively small computational time based on the functionalization of important parameters including α (constant related to the scale of the problem), p_a (probability of discovering Cuckoo's eggs) and λ (parameter related to the dimension of the problem). On the other hand, CFPA takes advantage of flower constancy, which allows similar flowers to exchange pollens in order to maintain a high convergence rate.

In summary, ICS produces the best solutions in term of area coverage for problem instances of small sizes while requiring the shortest computational time. On the other hand, IGA and PSO are the best algorithms in terms of balancing solution quality and execution time. However, the solutions delivered by all five algorithms have yet to reach the upper bounds in problem instances of large sizes. Furthermore, for smaller problem instances it is noticeable that some solutions have a corresponding area coverage even higher than the upper bound. The reason for this unreliability, we believe, is that the Monte Carlo sampling method used for calculating the approximate area coverage utilizes random generation of sampling points, leading to a slight inaccuracy in its outputs. This paper proposes a novel and modified version of IGA to overcome the weaknesses of existing methods.

4. Proposed Algorithm

To improve the quality of solutions, our objective is to develop a novel genetic algorithm based on IGA [14], which is currently the best genetic algorithm for the area coverage problem. Our modified IGA algorithm, henceforth known as MIGA, approaches the problem with a new individual representation, a different heuristic initialization, a combination of Laplace Crossover (LX) and Arithmetic Crossover Method ($AMXO$) operators, and a local search (VFA). Secondly, to increase the reliability of results, an exact integral area calculation is applied as the fitness function in replacement for the *Olap* technique [14] which does not fully reflect the area coverage of a sensor node set. Because the problem is approached in such a way that the fitness function works effectively with low computational complexity, MIGA is expected to perform better than IGA both in terms of solution quality and computational time. These improvements are detailed further in the following sections.

4.1. Individual Representation

In IGA and OTPHGA [25, 14], individual representations are complicated and fragmented since genes are grouped by their type, dividing the solution into k sections where k equals the number of sensor types.

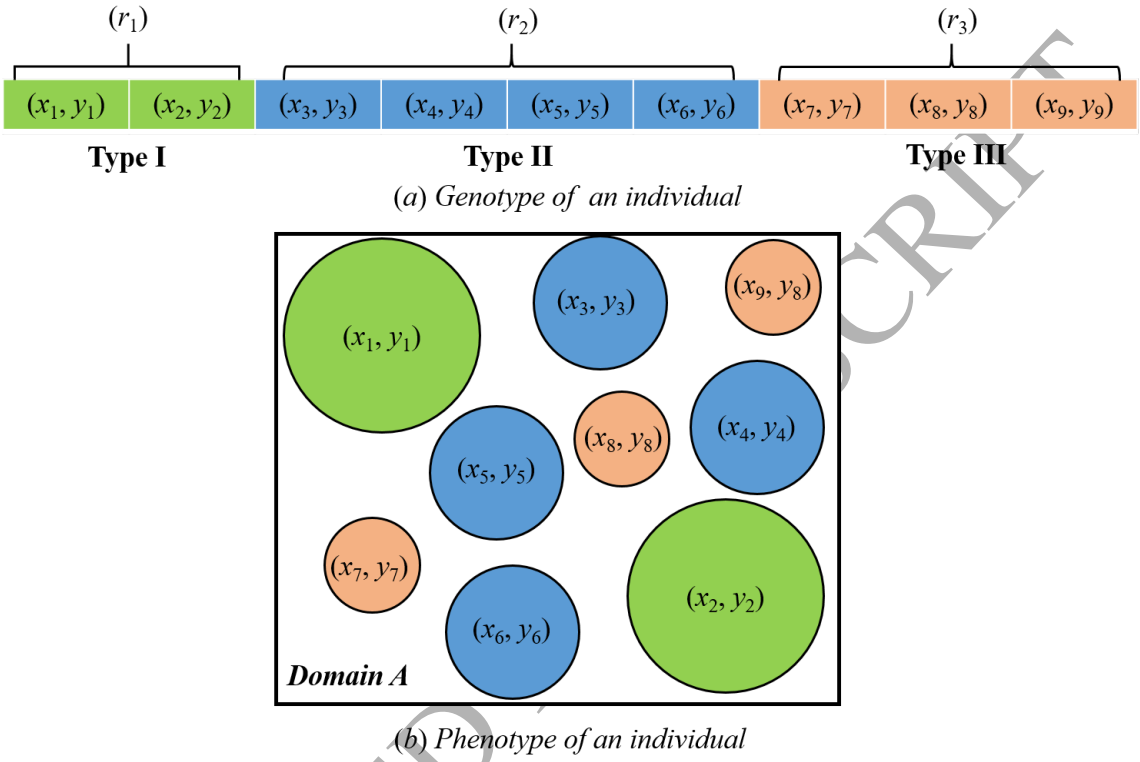


Figure 1: Individual representation of MIGA (a) illustrate genotypes, (b) illustrate phenotype.

Fig 1 shows an example of the individual representation, with $n = 9$ sensors, $k = 3$ sensor types, in which $n_1 = 2$ sensors of type I, $n_2 = 4$ sensors of type II and $n_3 = 3$ sensors of type III with ranges of r_1, r_2, and r_3 respectively.

4.2. Population Initialization

In general, a heuristic population initialization provides better solutions in terms of quality compared to random initialization [18, 7]. Therefore, in MIGA a population includes N individuals where each individual will be generated from the heuristic initialization within the surveillance region A such that their sensing range touches each other and/or the boundaries in order to minimize the overlapping areas. The initialization process is described as follows:

Step 1: A sensor is placed into the surveillance region A so that it is in contact with two of the four boundaries.

Step 2: Place another sensor in the surveillance region A to satisfy one of the following three conditions:

- The sensor is in contact with two of the four boundaries in the surveillance region A.
- The sensor is exposed to at least two sensors that have been deployed in the surveillance region A.
- The sensor is exposed to a boundary and a sensor that have been deployed in the surveillance region A.

Denote:

- C_{old} is the sensor deployed in the surveillance region A with coordinates $(x_{C_{old}}, y_{C_{old}})$ and radius $r_{C_{old}}$.
- C_{new} is a sensor being considered for deployment in the surveillance region A with coordinates $(x_{C_{new}}, y_{C_{new}})$ and radius $r_{C_{new}}$.
- $Distance(C_i, C_j)$ quantifies the Euclidean distance between two sensors C_i and C_j.
- $\triangle$ specifies the coverage intersection of sensor C_{new}.

The location of C_{new} to be placed inside the surveillance region A must satisfy all three conditions as follows, where $\triangle = 0$ on initialization:

$$Distance\,(C_{new}, C_{old}) + \triangle \geq W + \triangle \tag{4}$$

$$-\triangle \leq x_{Cnew} - r_{C_{new}}; x_{C_{new}} + r_{C_{new}} \leq W + \triangle \tag{5}$$

$$-\triangle \leq y_{C_{new}} - r_{C_{new}}; y_{C_{new}} + r_{C_{new}} \leq H + \triangle \tag{6}$$

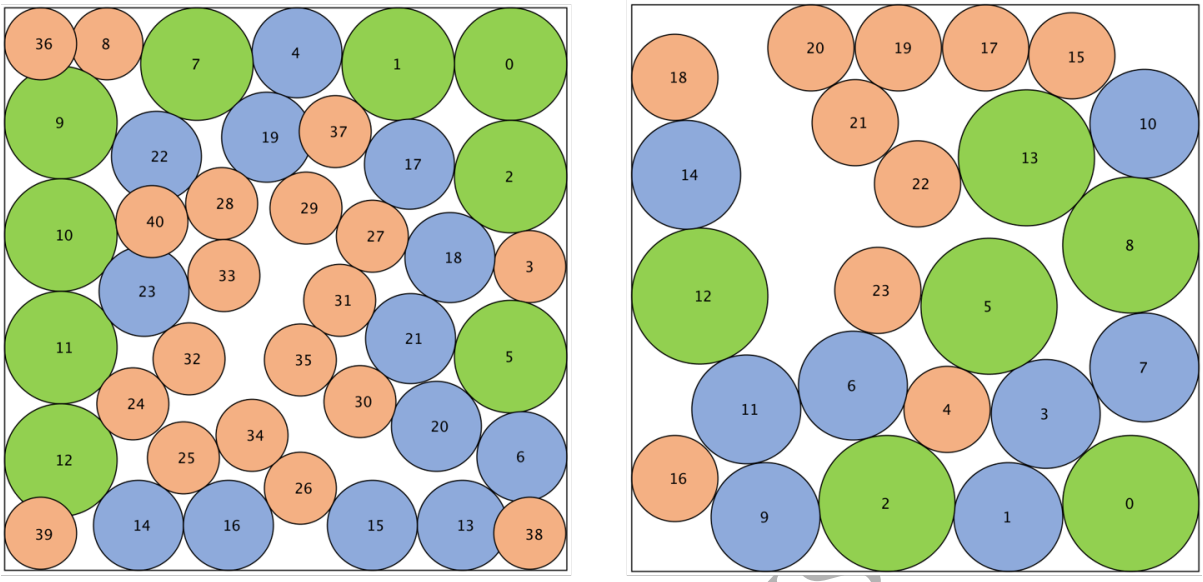
C_{set} represents the set of possible positions that C_{new} can be placed which satisfies the three conditions (4) (5) (6). If C_{set} is not empty, C_{new} will be deployed at any location in C_{set}. If C_{set} is empty, increase $\triangle$ by an amount of ξ, $(\triangle + \xi)$ and updates C_{Set}. The process is repeated until the position of C_{new} is found.

Repeat step 2 until all sensors are deployed in the surveillance region A. After the initialization of an individual stops, if $\triangle = 0$, the proposed initialization gives an optimal solution. Fig 2 illustrates two examples of optimal and non-optimal solutions.

4.3. Genetic Operators

4.3.1. Crossover Operator

In our new genetic algorithm, two basic crossover operators named the Laplace Crossover (LX) and the Arithmetic Crossover Method $(AMXO)$ as presented in [17, 9] will be performed on two parents $P_1 = \{(x_1, y_1), i = \overline{1,n}\}$ and $P_2 = \{(x_1', y_1'), i = \overline{1,n}\}$. The crossover is executed only on the coordinates of the parents, leaving the radius unchanged, resulting in two offspring called $Z_1 = \{(u_1, v_1), i = \overline{1,n}\}$ and $Z_2 = \{(u_1', v_1'), i = \overline{1,n}\}$ as shown in Fig 3. One of these two operators are fairly chosen when the crossover process is activated,



(*a*). A non-optimal solution with overlaps between some sensors in domain *A*. Because,

(*b*). An optimal solution with no overlap between any sensors in domain *A*.

Figure 2: Individuals produced by the proposed heuristic population initialization: (a) A non-optimal solution, (b) An optimal solution with no overlaps.

meaning that the probabilities of choosing between LX and $AMXO$ operators are equal. The reason for replacing $BLX\alpha$ in IGA [14] is that the offspring generated from P_1 and P_2 using $BLX\alpha$ strictly follows the genetic information from its parents without customizable configuration of parameters. Experiments suggest using both LX and $AMXO$ produces better results in term of solution quality.

The process of generating Z_1 and Z_2 using the two crossover operators LX and $AMXO$ are as follows:

Laplace Crossover ($\boldsymbol{LX}$) [17]: In LX crossover, two parents P_1 and P_2 produce two offspring. The offspring are given by the equations (7), (8) and (9).

$$Z_1 = \begin{cases} u_i = x_i + \beta \left| x_i - x_i^{'} \right|, i = \overline{1,n} \\ v_i = y_i + \beta \left| y_i - y_i^{'} \right|, i = \overline{1,n} \end{cases} \tag{7}$$

$$Z_2 = \begin{cases} u_i^{'} = x_i^{'} + \beta \left| x_i^{'} - x_i \right|, i = \overline{1,n} \\ v_i^{'} = y_i^{'} + \beta \left| y_i^{'} - y_i \right|, i = \overline{1,n} \end{cases} \tag{8}$$

where

$$\beta = \begin{cases} a + bln\left(\alpha\right) \left(\alpha > \frac{1}{2}\right) \\ a - bln\left(\alpha\right) \left(\alpha \leq \frac{1}{2}\right) \end{cases} \tag{9}$$

and $\alpha \in [0,1]$ is a uniformly distributed random number; $a = 0$; $b = 0.5$.

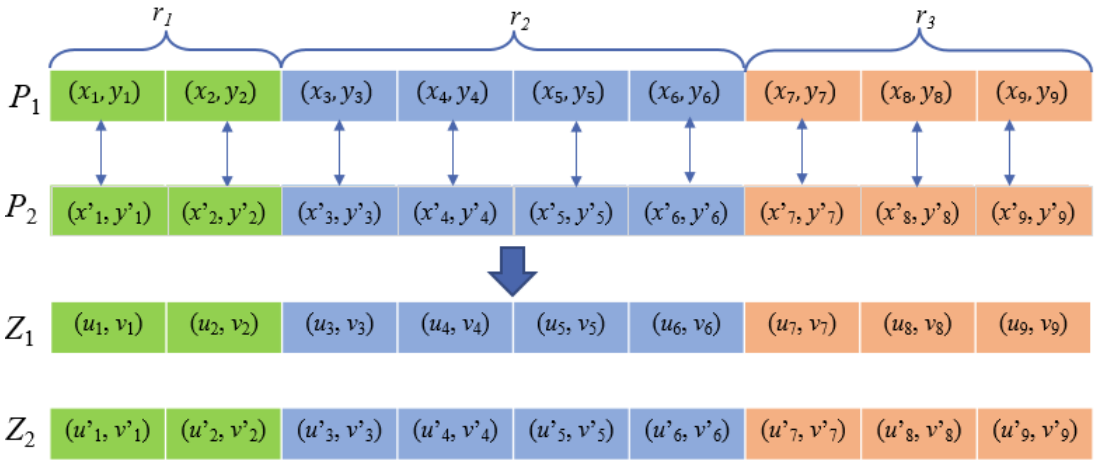


Figure 3: The process of generating offspring Z_1 and Z_2 by performing crossover between P_1 and P_2.

Arithmetic Crossover Method (*AMXO*) [17]: In *AMXO* crossover, two offspring Z_1 and Z_2 generated from parents P_1 and P_2 . The offspring are given by the equations (10) and (11).

$$Z_1 = \begin{cases} u_i = \alpha_i x_i + (1-\alpha_i)\, x_i^{'}, i = \overline{1,n} \\ v_i = \alpha_i y_i + (1-\alpha_i)\, y_i^{'}, i = \overline{1,n} \end{cases} \tag{10}$$

$$Z_2 = \begin{cases} u_i^{'} = \alpha_i x_i^{'} + (1-\alpha_i)\, x_i, i = \overline{1,n} \\ v_i^{'} = \alpha_i y_i^{'} + (1-\alpha_i)\, y_i, i = \overline{1,n} \end{cases} \tag{11}$$

where $\alpha_i \in [0,1]\,, i = \overline{1,n}$ are uniform random numbers.

4.3.2. Mutation Operator

The offspring $Z = \{(u_1, v_1), (u_2, v_2), ..., (u_n, v_n)\}$ produced by crossing-over the parents will be mutated by dynamic Gauss [24] as given by equations (12) and (13).

$$u_i^{'} = u_i + N\left(0, \left|x_i - x_i^{'}\right|^2\right) \tag{12}$$

$$v_i^{'} = v_i + N\left(0, \left|y_i - y_i^{'}\right|^2\right) \tag{13}$$

where $N\left(0, \left|x_i - x_i^{'}\right|^2\right)$ and $N\left(0, \left|y_i - y_i^{'}\right|^2\right)$ are Gaussian distributions. Fig 4 illustrates the process of the mutation operator.

4.3.3. Individual Adjustment

Newly generated individuals are adjusted such that all genes in an individual are sorted by their locations in the surveillance region A as follows. Given two circles $C_1\,(x_1, y_1)$ and $C_2\,(x_2, y_2)$, C_1 goes first in the gene arrangement of an individual when $y_1 \leq y_2$ and

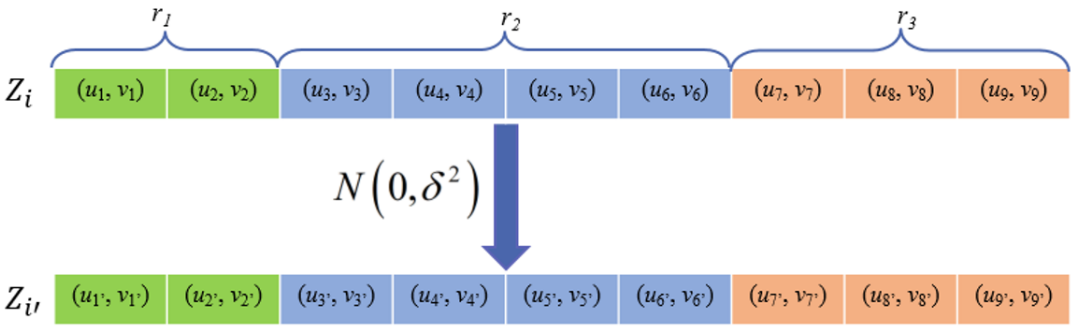
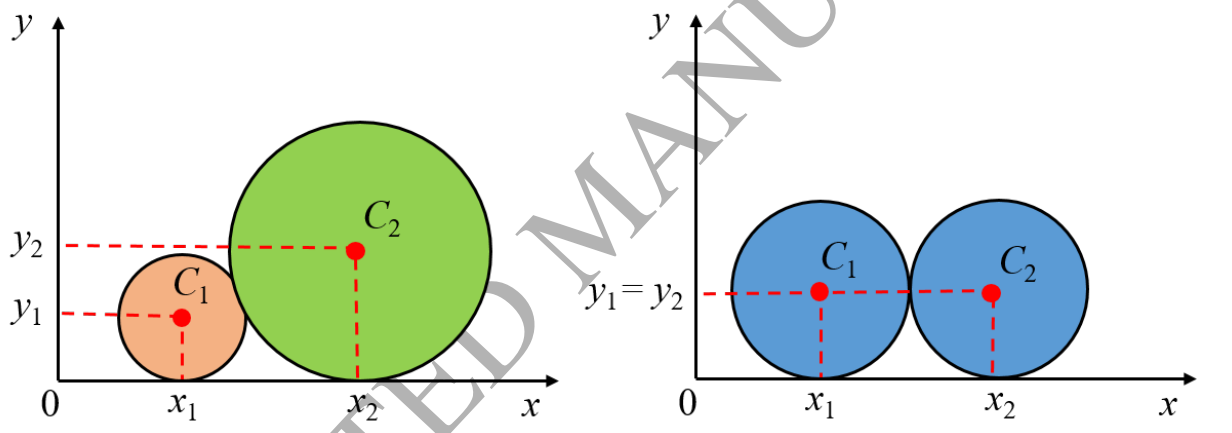


Figure 4: The process of mutation over the individual using dynamic Gauss function.

$x_1 \leq x_2$. Fig 5 illustrates several cases where C_1 goes before C_2 in the encoded solution. This arrangement reduces the number of potentially intersecting circles within one individual, thus lowering the calculation time of the fitness function in the next stage.



(a) The case in which ($y_1 < y_2$) and ($x_1 < x_2$) **(b) The case in which ($y_1 = y_2$) and ($x_1 < x_2$)**

Figure 5: Two cases of the individual adjustment where C_1 is placed before C_2 in the solution representation.

4.4. Fitness Function

As previously stated, some solutions produced by IGA [14], ICS, CFPA [6] and OT-PHGA [25] are higher than the theoretical upper bound because of instability in the Monte Carlo technique used for calculating the area coverage. Dinh [14] has greatly reduced the complexity of the fitness function with the proposal of the *Olap* technique. However, the fitness function calculated by using this technique does not fully reflect the area coverage of individuals, which results in a stability trade-off for decreasing the complexity.

In this paper, we show that the fitness function can be exactly calculated using an integral method [13] for computing the area coverage corresponding to a given number of sensor nodes. This ensures the stability of the solution and removes abnormal behaviors such

as the final area coverage going beyond the upper bound as seen in [14] and any approach that employs Monte Carlo method for approximate calculation.

Theorem 1. *An exact algorithm for calculating the fitness function exists with a time complexity of* $O(n^3 logn)$.

We present the proof for Theorem 1 in the Appendix section.

4.5. VFA Optimization on The Best Solution

After execution of MIGA reaches the generation limit or the results cannot be improved, the best solution in the population is further optimized by local search using Virtual Force Algorithm (VFA) [27].

The VFA algorithm aims to maximize the area coverage of a given number of sensors over the monitoring field. Each sensor node is considered a force source that pushes or pulls others away or towards it depending on the distance between it and other sensor nodes. When the ranges of two sensors intersect, they push each other away. Meanwhile, non-overlapping sensors are pulled together. By balancing the forces on the whole system, these behaviors ensure acceptable sensor density on the monitoring field while reducing the overlapping areas.

The adjustment process will be terminated when the coverage improvement is less than 0.1 area unit, meaning that it has reached the optimal area coverage arrangement for the given sensor node set.

5. Experiments and Results

5.1. Problem Instances

In our experiments, the 15 problem instances from [25, 14] were reused as listed in the Table 1 using the following parameters:

- The dimensions of the surveillance region A: 100 x 100.
- Three sensor types n_1, n_2, n_3 with sensing radii r_1, r_2, r_3 respectively where $r_2 = 0.8$ x r_1 and $r_3 = 0.8$ x r_2.
- The sensor tightness ratio α is the ratio between the total coverage achieved by the whole sensor node set and the area of the surveillance region A.

We compare MIGA with five state-of-the-art metaheuristics for the experimental instances defined in Table 1, namely, IGA, PSO, DPSO, ICS, and CFPA. The comparison takes into account three important aspects of a heuristic/metaheuristic: accuracy (average area coverage), efficiency (computational time), and stability (area coverage standard deviation). Because the experimental results in [25, 14] were computed using the Monte Carlo method, they cannot be directly compared to that of MIGA as presented in this study. Therefore, we had to recompute the final area coverage using our proposed fitness function.

Table 1: Experimental Instances.

Instances	r_1	n_1	r_2	n_2	r_3	n_3	n	α
s1-07	14.00	5	11.20	5	8.96	7	17	0.68
s2-07	12.00	6	9.60	8	7.68	10	24	0.69
s3-07	10.00	8	8.00	12	6.40	16	36	0.70
s4-07	8.00	12	6.40	18	5.12	27	57	0.70
s5-07	6.00	22	4.80	32	3.84	47	101	0.70
s1-08	14.00	5	11.20	6	8.96	10	21	0.80
s2-08	12.00	6	9.60	9	7.68	14	29	0.79
s3-08	10.00	9	8.00	13	6.40	19	41	0.79
s4-08	8.00	14	6.40	20	5.12	29	63	0.78
s5-08	6.00	25	4.80	36	3.84	55	116	0.80
s1-09	14.00	6	11.20	7	8.96	10	23	0.90
s2-09	12.00	7	9.60	11	7.68	14	32	0.89
s3-09	10.00	11	8.00	14	6.40	21	46	0.90
s4-09	8.00	16	6.40	23	5.12	34	73	0.90
s5-09	6.00	28	4.80	41	3.84	61	130	0.90

5.2. Parameter Settings

Table 2 shows the parameter settings for the GAs. For PSO, DPSO, ICS, and CFPA, the parameter values are the same as those in the corresponding publications. All algorithms were implemented in Java and run on a machine with Intel Core i5 2.4 GHz, RAM 8GB 1600 MHz DDR3.

5.3. Computational Results and Discussions

In this subsection, we report on three experiments. In the first experiment, we test the usefulness of each newly proposed component in MIGA (*MIGA Component Analyses*). The second experiment aims to validate the combination usage of two crossovers, *AMXO* and *LX* (*Effectiveness of the Combined Crossover in MIGA*). The third experiment is for comparing the performances of MIGA, IGA, PSO, DPSO, ICS and CFPA (*Performance Comparison*). The implementations are analyzed as follows.

5.3.1. MIGA Component Analyses

As MIGA is designed based on four main improvements over the state-of-art IGA, namely, a new initialization, a new fitness calculation scheme, new genetic operators, and improvement of the final solution using VFA, it raises the question as to whether the synergy of all additional components are necessary.

In the first experiment, four versions of MIGA were implemented such that each of them possesses only three out of the four mentioned improvements while the other one is kept similar to IGA's, which are MIGA1, MIGA2, MIGA3 and MIGA4. In each version, we

Table 2: The Experiment's Parameters.

Parameter	**Value**
The number of executions on one instance	30
Population size	50
The number of individual generated from the heuristic initialization	50
Generation limitation	1000
Crossover rate AMXO	0.9
Crossorver rate LX	0.1
Mutation rate	*1/n*

Table 3: The Combination of Different Components in Each Version of MIGA in The Second Experiment.

No.	**Intialization**		**Fitness**		**Genetic Operator**		**VFA**
	IGA	**MIGA**	**IGA**	**MIGA**	**IGA**	**MIGA**	
MIGA1	**x**			**x**		**x**	**x**
MIGA2		**x**	**x**			**x**	**x**
MIGA3		**x**		**x**	**x**		**x**
MIGA4		**x**		**x**		**x**	
MIGA		**x**		**x**		**x**	**x**

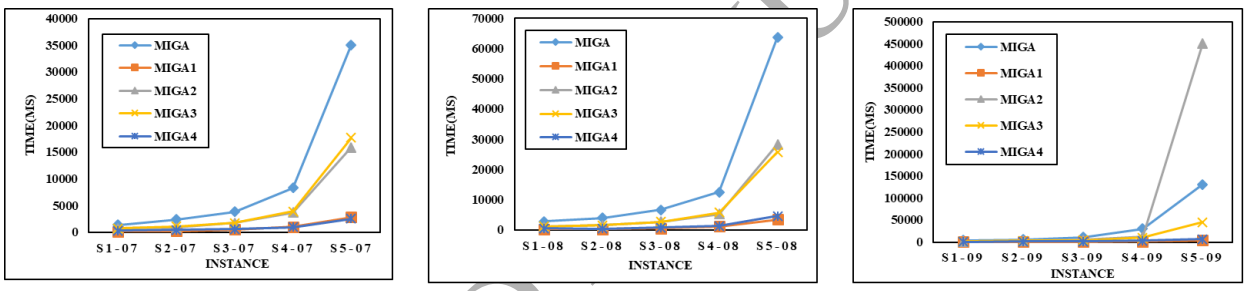
replaced one of the components used in MIGA by the corresponding one used in IGA as shown in Table 3.

In detail, MIGA1, MIGA2, MIGA3 and MIGA4 are used respectively for examining the effects of new methods of initialization, new fitness function, genetic operators and VFA. Four algorithms were tested using the same problem instances and system settings. The performance of MIGA1, MIGA2, MIGA3 and MIGA4 are compared to MIGA with respect to average area coverage, standard deviation and computational time (averaged over 30 runs) which are shown in Table 4 and Fig 6. The best values of average area coverage, computational time and standard deviation are marked in bold.

Results for each of the four versions of MIGA are depicted in Table 4 and Fig 6. First, in terms of average area coverage, MIGA achieved the best or equal best performance on 10 out of 15 instances, while the corresponding results for MIGA1, MIGA2, MIGA3, MIGA4, are 0/15, 10/15, 6/15, and 4/15 instances respectively. These results show that the new initialization, new genetic operators, and the usage of VFA are all essential for the genetic algorithms to find good area coverage. Excluding any of these three improvements will significantly degrade MIGA's performance. By contrast, the de-randomization of the fitness function (i.e to use the deterministic and exact coverage calculation for fitness) does not have much effect on the solution quality with respect to the average area coverage.

Table 4: Average Area Coverage (Avg) and Standard Deviation (SD) of MIGA1, MIGA2, MIGA3 and MIGA4 Compared to MIGA over 15 Instances.

Instance	MIGA		MIGA1		MIGA2		MIGA3		MIGA4		Upper
	Avg	SD	Avg	SD	Avg	SD	Avg	SD	Avg	SD	bound
s1_07	**6814.7**	*0*	6802.5	50.88	**6814.7**	4.53	**6814.7**	2.49	**6814.7**	2.15	6841.7
s2_07	**6883.6**	*0*	6866.3	73.79	**6883.6**	5.64	**6883.6**	1.53	**6883.6**	1.49	6883.6
s3_07	**6984.9**	*0*	6940.4	96.51	**6984.9**	9.45	**6984.9**	2.25	**6984.9**	9.96	6984.9
s4_07	**6952.6**	*0*	6926.1	2.49	**6952.6**	6.08	**6952.6**	6.86	**6952.6**	2.49	6952.6
s5_07	**6981.6**	*0*	6968.3	9.96	**6981.6**	9.56	**6981.6**	1.25	6935.3	1.49	6981.6
s1_08	**7955.0**	*1.99*	7760.3	282.04	7955	25.6	7939.3	119.22	7884.9	4.66	7965.4
s2_08	7909	*9.96*	7723.2	204.07	**7909.8**	11.01	7907.4	41.83	7795.0	1.55	7914.3
s3_08	**7884.2**	*2.98*	7698.9	255.84	7883.6	4.42	7883.8	6.26	7807.3	0.75	7886.2
s4_08	**7776.0**	*0*	7642	159.78	**7776.0**	2.71	7775.9	2.85	7694.4	0.57	7776.8
s5_08	7977.5	*2.98*	7752.6	166.54	**7977.6**	3.81	7977.4	4.36	7923.9	0.93	7981.1
s1_09	**8691.9**	*1.99*	8338.9	398.13	8676.7	104.35	8583.9	604.82	8540.4	26.48	8975.2
s2_09	8707.5	*1.99*	8260.8	334.38	**8708.8**	101.18	8689.9	265.43	8400.1	17.02	8945.7
s3_09	8756.3	*3.22*	8232.6	213.11	**8756.6**	68.91	8741.6	250.71	8413.7	16.93	8972.9
s4_09	**8766.2**	*9.96*	8208.8	174.84	8764.8	45.57	8769	59.39	8415.2	11.35	8976.7
s5_09	8786.6	*1.99*	8103.2	213.82	8785.4	29.99	**8788.1**	48.44	8575.1	7.07	8960.2



(*a*). Computational time with datasets covers about 70% domain *A*. **(*b*). Computational time with datasets covers about 80% domain *A*.** **(*c*). Computational time with datasets covers about 90% domain *A*.**

Figure 6: Average computational time between algorithms MIGA, MIGA1, MIGA2, MIGA3 and MIGA4 over 15 instances.

Secondly, regarding performance stability, it can be seen that MIGA's solutions have the smallest standard deviation on all 15/15 instances. The solution standard deviation for all genetic algorithms is shown in Table 4. Comparing MIGA and MIGA2 (MIGA2 used fitness function of IGA as shown in Table 3), it can be seen that MIGA achieved much smaller solution standard deviation (i.e maintains much more stable performance) especially on problems of large sizes (s1-09 to s5-09). This shows that the new fitness function indeed helps to solve the problem of performance instability of the old genetic algorithms (mainly due to the approximate and/or stochastic nature of the fitness functions).

Third, in terms of computational time, MIGA1 and MIGA4 have the best performance on 15/15 problem instances respectively. These figures are shown in Fig 6. This is understandable as the use of VFA and especially the new initialization scheme significantly increase the running time of the genetic algorithms. However, this increase in computational time (for

MIGA, MIGA2, MIGA3) is a worthwhile trade-off for the performance, as without either VFA or the new initialization scheme, the genetic algorithms (MIGA1, MIGA4) performed poorly on the test problem instances.

Overall, the experimental results show that all newly proposed components in MIGA are essential for achieving good coverage and performance stability (low solution standard deviation) even at the cost of higher computational time.

5.3.2. Effectiveness of the Combined Crossover in MIGA

In this experiment, we validate the effectiveness of the combined crossover in MIGA. Firstly, we implement MIGA with a single crossover only. Table 5 compares the performance of MIGA with three standalone operators: BLX_α [14, 25], LX, and $AMXO$ on the 15 problem instances. Their scores on 70% area coverage instances (s1-07 to s5-07) are identical. Similarly, LX and $AMXO$ could not outperform BLX_α in the next 10 instances (s1-08 to s5-08) and (s1-09 to s5-09). To be more precise, MIGA which employs $AMXO$ is only better than MIGA using BLX_α over 6/10 instances. LX produces even worse solutions than BLX_α in all instances.

Our hypothesis is that appropriately combining the two crossover operators LX and $AMXO$ can significantly improve the diversity of created offspring and eventually lead to more optimal solutions. One issue is selecting the proportion of offspring originating from each operator. MIGA with $AMXO$ only facilitates slightly better solutions than with BLX_α, whereas MIGA with LX does not outperform BLX_α. Therefore, the percentage of offspring from $AMXO$ should be considerably higher than that of LX. We also perform a grid search to find the best combination (see Table 6), where the best identified ratio is 90% for $AMXO$ and 10% for LX. The combined crossover is able to outperform MIGA with BLX_α in all challenging instances (s1-08 to s5-08) and (s1-09 to s5-09) and obtains the best solution in 13 out of 15 benchmark instances.

We observe from LX's formulation given by the equations(7), (8) and (9) in that the changes to the resultant offspring are dependent on the difference between its parents. This means that LX can produce drastically different offspring, especially in early generations. We theorize that since LX is a more unstable operator, using it in conjunction with $AMXO$ can enhance the search space for better solutions, whereas using only LX results in an unstable search that is not quite as effective.

5.3.3. Performance Comparison

In the third experiment, we compare the performance of MIGA, IGA, PSO, DPSO, ICS and CFPA. For each problem instance, each system was run 30 times. We recorded the average area coverage of obtained solutions (bigger is better), the standard deviation (smaller is better), and average computation time.

Table 7 depicts the results of IGA, PSO, DPSO, ICS, CFPA and MIGA, where the best values of average area coverage (Avg) and standard deviation (SD) are highlighted. Fig 8 illustrates the average computational time for all 6 algorithms on the datasets of 70%, 80%, 90% area coverage.

Table 5: Average Area Coverage Between Four Crossovers BLX_α, $AMXO$, LX, and MIGA ($AMXO$, LX) with Upper Bound Over 15 Instances.

Instance	MIGA(BLX_α)	MIGA($AMXO$)	MIGA(LX)	MIGA	Upper Bound
s1_07	**6814.65**	**6814.65**	**6814.65**	**6814.65**	6814.65
s2_07	**6883.56**	**6883.56**	**6883.56**	**6883.56**	6883.56
s3_07	**6984.89**	**6984.89**	**6984.89**	**6984.89**	6984.89
s4_07	**6952.56**	**6952.56**	**6952.56**	**6952.56**	6952.56
s5_07	**6981.63**	**6981.63**	**6981.63**	**6981.63**	6981.63
s1_08	7939.30	7953.06	7911.61	**7955.02**	7965.37
s2_08	7907.40	**7909.60**	7649.46	7908.98	7914.28
s3_08	7883.80	7883.99	7875.71	**7884.15**	7886.15
s4_08	7775.87	7775.15	7774.61	**7776.72**	7776.75
s5_08	7977.36	7977.28	7968.42	**7977.53**	7981.05
s1_09	8583.89	8681.15	8449.51	**8691.89**	8975.20
s2_09	8689.89	8708.39	8507.49	**8708.49**	8945.73
s3_09	8741.57	8755.78	8570.15	**8756.30**	8972.89
s4_09	8769.00	8765.14	8564.35	**8766.17**	8976.69
s5_09	8785.40	**8787.39**	8597.04	8786.63	8960.20

Table 6: Average Area Coverage Obtained by MIGA with Different Crossover Combination Rates

Crossover Combination Rates		Area Coverage Obtained by MIGA				
		Instance (s1-07)	Instance (s1-08)	Instance (s4-08)	Instance (s1-09)	Instance (s5-09)
AMXO	LX	MIGA	MIGA	MIGA	MIGA	MIGA
0.1	0.9	6814.65	7906.65	7775.00	8442.01	8594.94
0.2	0.8	6814.65	7905.51	7774.60	8440.88	8593.44
0.3	0.7	6814.65	7913.15	7774.90	8448.61	8598.63
0.4	0.6	6814.65	7911.56	7775.13	8549.96	8599.17
0.5	0.5	6814.65	7908.20	7775.50	8539.60	8602.40
0.6	0.4	6814.65	7914.18	7774.95	8548.64	8604.71
0.7	0.3	6814.65	7912.22	7775.05	8652.08	8697.75
0.8	0.2	6814.65	7935.92	7775.14	8651.94	8709.54
0.9	*0.1*	**6814.65**	**7955.56**	**7775.54**	**8691.04**	**8785.21**

It can be seen from Table 7 and Fig 8 that MIGA offers excellent results throughout the experiment and outperforms IGA, PSO, DPSO, ICS and CFPA in terms of solution quality in 13/15 instances. In comparison with the solution produced by ICS (the second best algorithm), that of MIGA is improved up to nearly 1.5%, which is fairly good considering the fact that ICS's solutions were nearly optimal.

For small instances (s1-07 to s5-07), the area coverage achieved reaches the upper bound with insignificant deviation, showing that MIGA consistently delivers an optimal placement scheme for the given sensor node set as in depicted in Table 7. For instances of medium size (s1-08 to s5-08), MIGA also produces better results that almost meet the upper bound. In particular, experimental results on these instances fall short about 0.1% under the upper bound. In large instances (s1-09 to s5-09), there is still about 3% area coverage remaining to reach the upper bound, however all solutions are improved compared to the other algorithms.

As can be seen in Table 7 the standard deviations regarding the average area coverage of MIGA over 30 runs are much less than that of the others over 15/15 instances. This is understandable since BLX_α allows the offspring to be located within the square formed by two parental genes location, thus resulting in lower stability compared to LX and $AMXO$ crossover operators. Furthermore, MIGA uses an exact integral area calculation for the fitness function, which contributes to a more stable algorithm than those with approximate fitness functions.

We also conducted experiments on the convergence of MIGA. From Fig.7, MIGA converges around the 300^{th} generation. However, for fair judgment between algorithms using genetic algorithms (IGA and MIGA) we have set the number of generations for MIGA to be 1000 like that of IGA.

With respect to computational time, as shown in Fig 8, the experimental results show that the calculation time of MIGA is on average three times longer than what ICS achieves. This is explainable as ICS explores the search space more efficiently thanks to the simplicity of each random walk and high rate of convergence. However, this difference is acceptable considering the trade-off between the accuracy and computational time.

The improvements we've shown for MIGA can be attributed to several factors:

1. The use of an exact fitness function enables more stable evaluations during each generation.
2. Our proposed initialization heuristic creates solutions where sensors are more "spread out" as opposed to the heuristic proposed in [14], [11] and [6], which stacks sensors linearly on the plane. This allows more optimal initial solutions and a better search space for GA.
3. Our combination of two crossover operators creates more diverse populations that help find high quality solutions that are harder for BLX_α to locate.

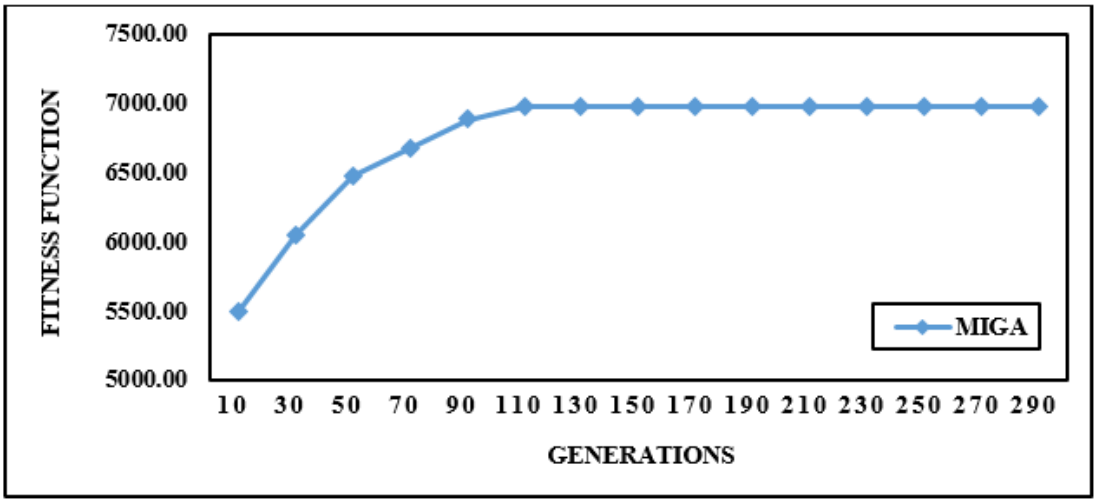
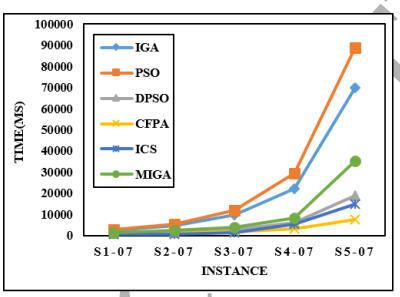
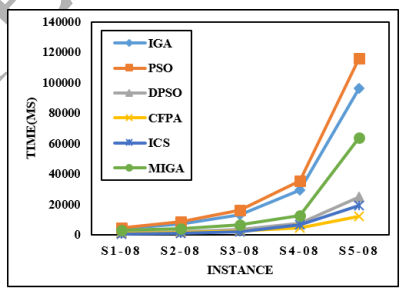


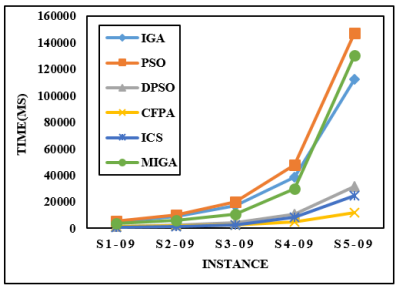
Figure 7: Convergence of MIGA in instance s5-07.



(*a*). Computational time with datasets covers about 70% domain *A*.



(*b*). Computational time with datasets covers about 80% domain *A*.



(*c*). Computational time with datasets covers about 90% domain *A*.

Figure 8: Average computational time between algorithms IGA, PSO, DPSO, ICS, CFPA and MIGA over 15 instances.

Table 7: Average Area Coverage (Avg), Standard Deviation (SD) of MIGA, IGA, PSO, DPSO, CFPA, and ICS on 15 Instances.

	IGA		PSO		DPSO		CFPA		ICS		MIGA		Upper
Instance	**Avg**	**SD**	**Avg**	**SD**	**Avg**	**SD**	**Avg**	**SD**	**Avg**	**SD**	**Avg**	**SD**	**bound**
s1_07	6814.2	2.43	6814.6	0.21	6814.58	*0*	6813.3	3.90	6815.0	3.87	**6814.7**	*0*	6814.7
s2_07	6883.6	*0*	6883.6	*0*	6883.52	*0*	6884.7	4.82	6883.3	5.09	**6883.6**	*0*	6883.6
s3_07	6984.5	1.6	6984.9	*0*	6984.89	*0*	6986.1	4.25	6984.9	5.21	**6984.9**	*0*	6984.9
s4_07	6952.6	0.01	6952.6	*0*	6952.56	*0*	6952.4	4.96	6952.8	3.86	**6952.6**	*0*	6952.6
s5_07	6981.6	0.01	6981.6	*0*	6981.63	*0*	6981.1	3.91	6981.4	3.12	**6981.6**	*0*	6981.6
s1_08	7897.1	36.67	7886.9	48.8	7907.74	30.47	7925.4	23.27	7902.5	47.91	**7955**	*4.66*	7965.4
s2_08	7867.6	37.75	7878.5	25.4	7882.68	16.77	7886.6	20.15	7888.7	13.18	**7909.0**	*1.55*	7914.3
s3_08	7865.3	14.49	7870.4	10.6	7870.23	12.33	7875.1	9.74	7876.5	9.60	**7884.2**	*0.75*	7886.2
s4_08	7771.2	6.1	7773.5	4.09	7775.39	1.87	**7777.0**	5.11	7776.3	3.30	7775.7	*0.57*	7776.8
s5_08	7973.5	3.89	7975.3	3.2	7977.41	1.82	7979	5.15	**7981.9**	2.95	7977.5	*0.93*	7981.1
s1_09	8568.8	72.23	8576.5	56.6	8616.37	46.31	8637.2	50.66	8595.5	55.32	**8708.9**	*26.5*	8975.2
s2_09	8605.7	33.27	8645.8	33	8618.81	45.2	8623.4	36.25	8629.4	40.69	**8707.5**	*17*	8945.7
s3_09	8660.0	37.05	8677	28.9	8673.39	44.16	8675.5	46.98	8704.4	23.95	**8756.3**	*16.9*	8972.9
s4_09	8698.6	36.07	8727.3	31	8718.42	30.08	8707.5	33.63	8742.2	22.54	**8766.2**	*11.4*	8976.7
s5_09	8733.6	21.57	8757.3	15.3	8736.17	36.51	8723	42.29	8774.1	12.67	**8786.6**	*7.07*	8960.2

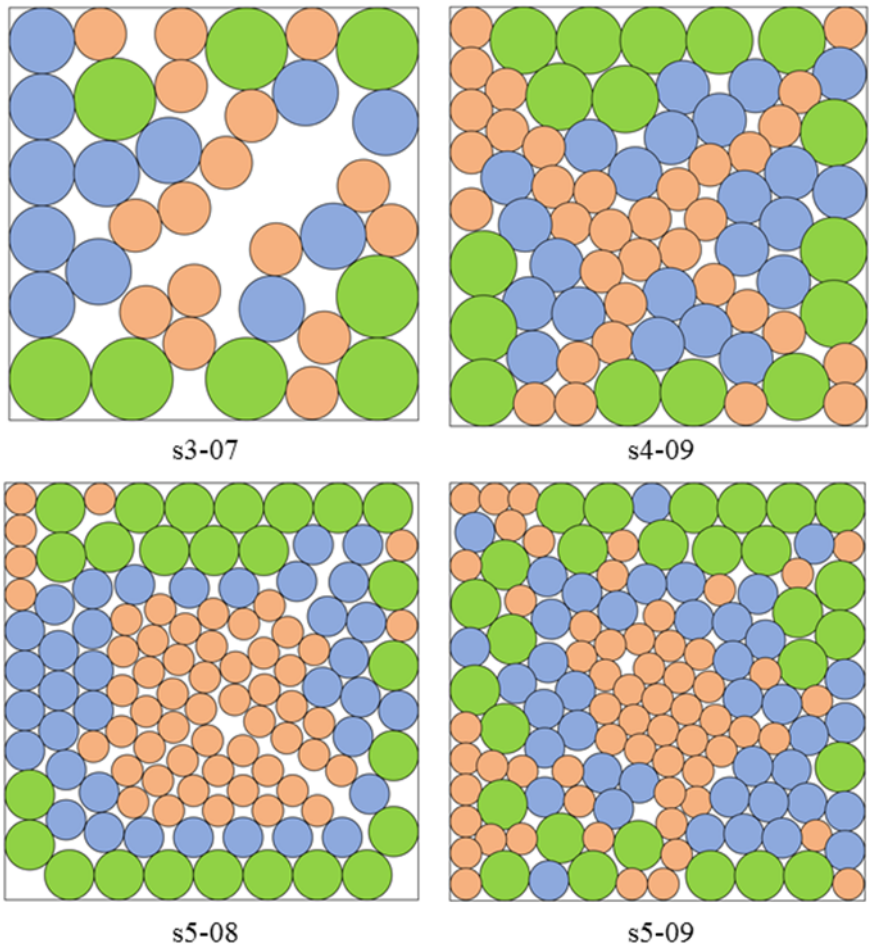


Figure 9: Solution derived by IGA and MIGA algorithms on test instances s3-0.7, s4-0.9, s5-08 and s5-0.9.

6. Conclusion

In this paper, we proposed a novel genetic algorithm called MIGA to solve the problem of maximizing area coverage in a WSN with heterogeneous sensing ranges. MIGA includes a new heuristic initialization, a new fitness function based on an exact integral area calculation and a combinatorial use of *LX* and *AMXO* crossover operators. In addition, MIGA uses VFA as a local search for refining the final solution quality. Our method was tested on 15 benchmark problem instances and results were compared with those of five state-of-the-art methods. The experimental results show that our method outperformed all other compared methods in terms of solution quality and stability, while keeping the computational time acceptable.

In addition to testing the performance of MIGA, we evaluated the contribution of each of its components to the overall performance by devising additional experimental scenarios using a leave-one-out approach. The results from this analysis demonstrated that every component in MIGA is essential for achieving optimal solution quality and stability.

The proposed combination of *LX* and *AMXO* crossover operators gives positive and promising results when implemented on the same sets of instances. It is also important to decide the rate of each operator being applied on the individuals. After extensive experiments, it can be observed that the combined operator achieves optimal results when *LX* is applied at a rate of 10% and *AMXO* at a rate of 90%.

These conclusions are meaningful for future work on this problem and can pave the way for advancements in the deployment of WSNs and the field of evolutionary omputing.

In the future, we are aiming to study the maximization of obstacle constrained area coverage in WSNs to adapt to realistic requirements. Furthermore, we will study area coverage problems related to the connectivity and lifetime of WSNs.

7. Acknowledgment

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number DFG 102.01-2016.03. The authors would like to thank Emerson Keenan from The University of Melbourne for his suggestions towards improving the quality of the manuscript.

References

[1] Abo-Hammour, Z., Arqub, O. A., Alsmadi, O., Momani, S., & Alsaedi, A. (2014). An optimization algorithm for solving systems of singular boundary value problems. *Applied Mathematics & Information Sciences*, *8*, 2809.

[2] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). A survey on sensor networks. *IEEE Communications Magazine*, *40*, 102–114. doi:10.1109/MCOM.2002.1024422.

[3] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer networks*, *38*, 393–422.

[4] Alemdar, H., & Ersoy, C. (2010). Wireless sensor networks for healthcare: A survey. *Comput. Netw.*, *54*, 2688–2710.

[5] Arqub, O. A., & Abo-Hammour, Z. (2014). Numerical solution of systems of second-order boundary value problems using continuous genetic algorithm. *Information sciences*, *279*, 396–415.

[6] Binh, H. T. T., Hanh, N. T., Dey, N. et al. (2016). Improved cuckoo search and chaotic flower pollination optimization algorithm for maximizing area coverage in wireless sensor networks. *Neural Computing and Applications*, (pp. 1–13).

[7] Burke, E. K., Newall, J. P., & Weare, R. F. (1998). Initialization strategies and diversity in evolutionary timetabling. *Evolutionary computation*, *6*, 81–103.

[8] Chong, C.-Y., & Kumar, S. P. (2003). Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, *91*, 1247–1256. doi:10.1109/JPROC.2003.814918.

[9] Deep, K., & Thakur, M. (2007). A new crossover operator for real coded genetic algorithms. *Applied mathematics and computation*, *188*, 895–911.

[10] Fan, G., & Jin, S. (2010). Coverage problem in wireless sensor network: A survey. *JNW*, *5*, 1033–1040.

[11] Hanh, N. T., Nam, N. H., & Binh, H. T. T. (2016). Particle swarm optimization algorithms for maximizing area coverage in wireless sensor networks. In *Proceedings of SAI Intelligent Systems Conference* (pp. 893–904). Springer.

[12] Howard, A., Matarić, M. J., & Sukhatme, G. S. (2002). Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In H. Asama, T. Arai, T. Fukuda, & T. Hasegawa (Eds.), *Distributed Autonomous Robotic Systems 5* (pp. 299–308). Tokyo: Springer Japan.

[13] Lovitt, W. V. (1924). *Linear Integral Equations*. McGraw-Hill.

[14] Ly, D. T. H., Hanh, N. T., Binh, H. T. T., & Nghia, N. D. (2015). An improved genetic algorithm for maximizing area coverage in wireless sensor networks. In *SoICT*.

[15] ur Rehman, A., Abbasi, A. Z., Islam, N., & Shaikh, Z. A. (2014). A review of wireless sensors and networks' applications in agriculture. *Computer Standards & Interfaces*, *36*, 263 – 270.

[16] Sangwan, A., & Singh, R. P. (2015). Survey on coverage problems in wireless sensor networks. *Wirel. Pers. Commun.*, *80*, 1475–1500.

[17] Sorsa, A., Peltokangas, R., & Leiviska, K. (2008). Real-coded genetic algorithms and nonlinear parameter identification. In *Intelligent Systems, 2008. IS'08. 4th International IEEE Conference* (pp. 10–42). IEEE volume 2.

[18] Talbi, E.-G. (2009). *Metaheuristics: from design to implementation* volume 74. John Wiley & Sons.

[19] Wang, B. (2011). Coverage problems in sensor networks: A survey. *ACM Comput. Surv.*, *43*, 32:1–32:53.

[20] Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and computing*, *4*, 65–85.

[21] Wilson, J. S. (2004). *Sensor technology handbook*. Elsevier.

[22] Xia, F., Yang, L. T., Wang, L., & Vinel, A. (2012). Internet of things. *Int. J. Commun. Syst.*, *25*, 1101–1102. URL: http://dx.doi.org/10.1002/dac.2417. doi:10.1002/dac.2417.

[23] Yinbiao, S., Lee, K., Lanctot, P., Jianbin, F., Hao, H., Chow, B., & Desbenoit, J. (2014). Internet of things: wireless sensor networks. *White Paper, International Electrotechnical Commission, http://www.iec. ch*, .

[24] Yoon, Y., & Kim, Y.-H. (2012). The roles of crossover and mutation in real-coded genetic algorithms. In S. Gao (Ed.), *Bio-Inspired Computational Algorithms and Their Applications* chapter 4. Rijeka: InTech. URL: https://doi.org/10.5772/38236. doi:10.5772/38236.

[25] Yoon, Y., & Kim, Y. H. (2013). An efficient genetic algorithm for maximum coverage deployment in wireless sensor networks. *IEEE Transactions on Cybernetics*, *43*, 1473–1483.

[26] Zheng, J., & Jamalipour, A. (2009). *Wireless sensor networks: a networking perspective*. John Wiley & Sons.

[27] Zou, Y., & Chakrabarty, K. (2003). Sensor deployment and target localization based on virtual forces. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)* (pp. 1293–1303 vol.2). volume 2.

Appendix

In this section, we present the proof for Theorem 1 by showing a method of exactly calculating the area covered by the sensors in the domain A and proving its complexity to be $O\left(n^3 logn\right)$. The method is implemented in two steps:

Step 1: Find all intersections of the circles and the contacting points between the circles with straight lines parallel to the vertical axis. $x-coordinates$ of the points are stored in an array x, representing the vertical lines that go through that $x-coordinates$. After that, the array is sorted in ascending order. Each pair of two consecutive array members form a vertical section on surveillance region A called a slot (see Fig 10). This step ensures that all slots are arranged in ascending order, which is useful for the next step.

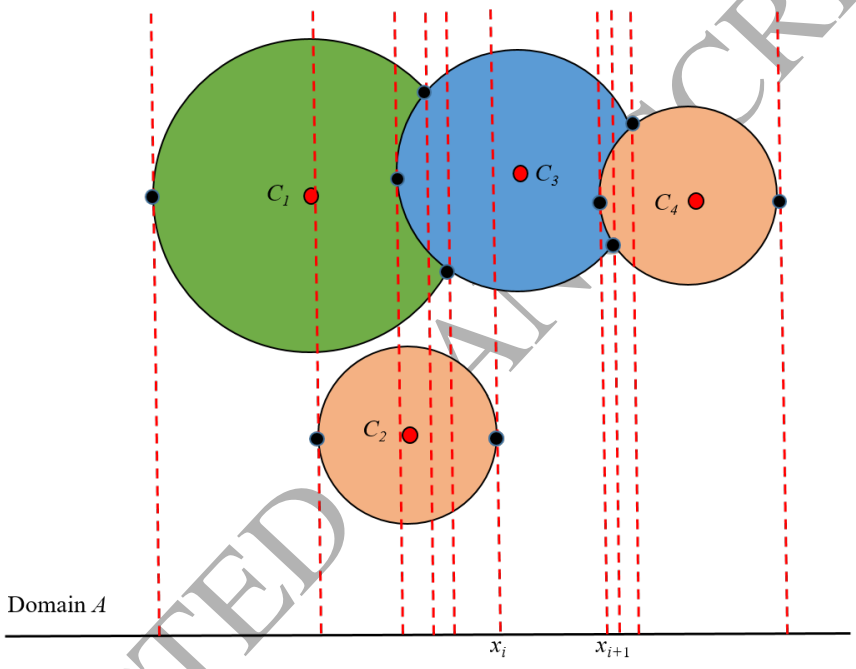


Figure 10: Demonstration for redundant step 1 of calculation. Dividing the surveillance region A into small pieces by vertical lines that contact the circles or go through the intersection points between circles called slots.

Step 2: The purpose of this step is to calculate the area coverage of all the sensors in each slot.

Consider each array member x_i of all intersections calculated after step 1; If $x_i < x_{i+1}$, calculate the area S_i covered by the given node set, which is equal to the area of circles redundant in the domain A limited by two straight lines $x = x_i$ and $x = x_{i+1}$; in other words, they are limited by a slot. Assume there are m slots $(S_i, i = \overline{1,m})$.

After calculating all component areas S_i, the total area coverage of the whole sensor node set on the surveillance region A is calculated as equation (14).

$$S = \sum_{i=1}^{m} S_i \tag{14}$$

Getting to this point, the objective is narrowed down to how to calculate the area of a circle under a narrow slot limited by 2 lines $x = a$ and $x = b$. To do this, we calculate the exact integral part of this area as follows.

First, we find all circles that are within the slot limited by two straight lines $x = a$ and $x = b$ (for simplicity, denote this space domain D from hereon).

The circle C_i overlays domain D if and only if $x_i - r_i \leq a \leq x_i + r_i$ and $x_i - r_i \leq b \leq x_i + r_i$; where (x_i, y_i) and r_i are the coordinates and radius of a sensor that the circle C_i represents.

The coverage that C_i casts on domain D is calculated as follow:

Case 1: When there is only one sensor C_i, area coverage of C_i in the domain D ($Area_i$) is calculated as given in equation (15), (16) and (17).

$$Area_i = \int_a^b g_i(x)dx - \int_a^b f_i(x)dx \tag{15}$$

$$Area_i = \int_a^b \left(y_i + \sqrt{r_i^2 - (x - x_i)^2} \right) dx - \int_a^b \left(y_i - \sqrt{r_i^2 - (x - x_i)^2} \right) dx \tag{16}$$

$$Area_i = 2 \int_a^b \sqrt{r_i^2 - (x - x_i)^2} dx \tag{17}$$

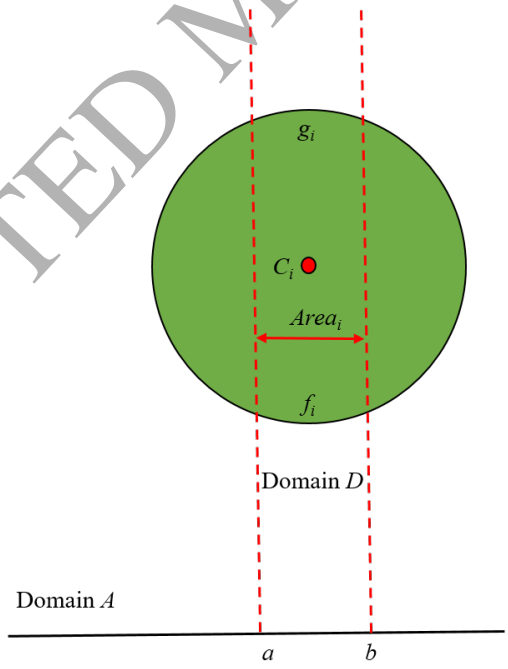


Figure 11: The relative positions of one circle in one slot in the domain D.

Case 2 : When there are many circles overlaying the domain D and they might even intersect with each other, it is inaccurate to calculate the total area coverage as the sum of areas cast by all circles on domain D. In order to provide an exact result of the coverage in case two or more circles intersect inside the domain D, we sort all the bounding lines and curves in ascending order with regard to the vertical direction and reject unnecessary or shadowed ones. The process is described as follows.
Assume there are t circles inside domain D (denote: $C_i, i = \overline{1,t}$).
Considering two intersecting circles C_i and $C_j, 1 \leq i, j \leq t$, their relative positions could only fall into one of two cases as illustrated in Fig 12 and Fig 13.
Case 2.1: In the first case, two circles C_i and C_j which do not intersect in the domain D as shown in Fig 12 are calculated as follows in equation (18).

$$Area_{ij} = Area_i + Area_j \tag{18}$$

where $Area_i$ and $Area_j$ are calculated as in equation (15), (16) and (17).

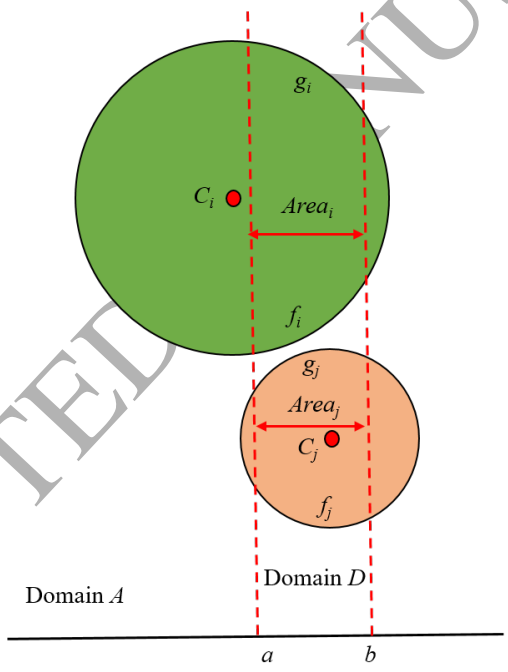


Figure 12: The relative positions of two circles in one slot. when two circles are separated from each other in the domain D.

Case 2.2: In the second case, two circles C_i and C_j intersecting in the domain D as shown in Fig 13 are calculated as follows in equation (19), (20) and (21).

$$Area_{ij} = \int_a^b g_i(x)dx - \int_a^b f_j(x)dx \tag{19}$$

$$Area_{ij} = \int_a^b \left(y_i + \sqrt{r_i^2 - (x - x_i)^2} \right) dx - \int_a^b \left(y_j - \sqrt{r_j^2 - (x - x_j)^2} \right) dx \tag{20}$$

$$Area_{ij} = (y_i - y_j)(b - a) + \int_a^b \sqrt{r_i^2 - (x - x_i)^2} dx + \int_a^b \sqrt{r_j^2 - (x - x_j)^2} dx \tag{21}$$

All the integral formulas given in this paper are calculated base on equation (22).

$$\int \sqrt{a^2 - x^2} dx = \frac{x\sqrt{a^2 - x^2}}{2} + \frac{a^2}{2} arcsin\frac{x}{a} + c \tag{22}$$

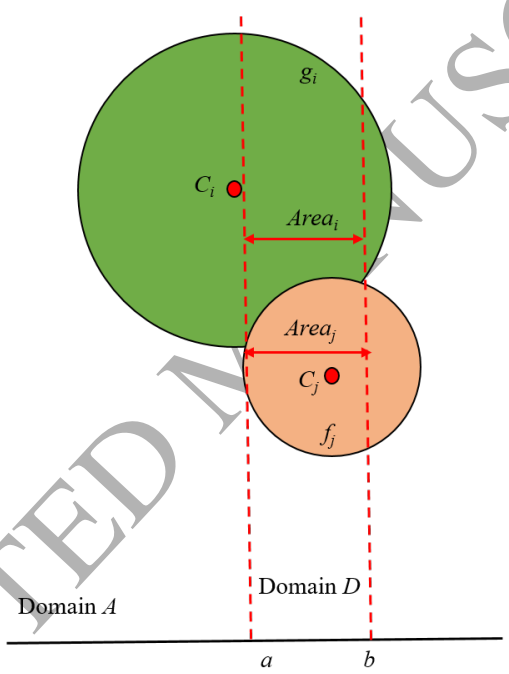


Figure 13: The relative positions of two circles in one slot. when there exists circle-circle intersection somewhere in the domain D, resulting in an overlapping region.

555

The process of this constructive proof is presented in the pseudo code of Algorithm 1.

Algorithm 1: Exact Coverage Calculation

Input : The area of the entire field: $A = W \times H$;
The list of sensor nodes
Output: The total area coverage S casted by all sensor nodes on the surveillance region A.

Begin
Function $area\,(c)$
$a \leftarrow \emptyset$
$index \leftarrow 0$
for *i=1 to n do* **do**
  $a\,(index) \leftarrow x_i - r_i$
  $index \leftarrow index + 1$
  $a\,(index) \leftarrow x_i + r_i$
  $index \leftarrow index + 1$
  for *i=1 to n do* **do**
    if c_i *intersect* c_j **then** break;
    ponits $\leftarrow$ *intersection points of* c_i and c_j;
    for *p in points* **do**
      $a\,(index) \leftarrow p.x$
      $index \leftarrow index + 1$
    end
  end
end
$Sort(a)$
$k \leftarrow$ size of array a
$s \leftarrow 0$
for *i=1 to k-1 do* **do**
  Calculate the area covered of the sensors in slot i^{th} $(Area_i)$
  $s \leftarrow s + Area_i$
end
return(s)

The complexity of Algorithm 1 is computed as follow. First, it computes the largest number of intersection points created by n sensors. The complexity of this stage is $O(n^2)$. Consequently, the number of slots is at most is n^2. For each slot, we need to compute the number of intersection points of n sensors and sort these points, thus the complexity of this step is $O(nlogn)$. Finally, the complexity of the whole Algorithm 1 is $O(n^3logn)$.

Biography

Nguyen Thi Hanh

Huynh Thi Thanh Binh

Nguyen Xuan Hoai

arimuthu Swami Palaniswami