Received June 2, 2020, accepted June 19, 2020, date of current version August 25, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3005986

Parallel Genetic Algorithm to Extend the Lifespan of Internet of Things in 5G Networks

YING ZHANG, (Member, IEEE), WEIHONG YU, XIAODONG CHEN, AND JIANHUI JIANG
School of Software Engineering, Tongji University, Shanghai 201804, China

Corresponding author: Jianhui Jiang (jhjiang@tongji.edu.cn)

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant 61432017, Grant 61772199, and Grant 61974105.

ABSTRACT With the development and popularization of 5G networks, the coverage problem of the Internet of Things (IoT) will encounter the massive-node problem. In this paper, we design a parallel genetic algorithm that divides the coverage problem of IoTs with massive nodes into many small problems and then solves these problems using Hadoop in parallel. First, the algorithm uses partitioning and grouping operations to degrade the scale of a large IoT and makes the coverage problem solvable. The algorithm then adopts the multi-objective programming-based genetic algorithm (MPGA) to solve the coverage problem. MPGA uses the fast non-dominated sorting to optimize the IoT coverage and node redundancy; it implements the preferential selection of non-critical nodes to maximize the length of the configuration sequence of working nodes. Finally, the parallel genetic algorithm uses uniform mutation and individual pruning to optimize the genetic algorithm internally and force its solving process to quickly converge toward feasible solutions. Experimental results confirm that the MPGA outperforms the existing algorithm on small IoTs in terms of coverage, the number of nodes, computing time, and the IoT lifespan. They also demonstrate that the parallel genetic algorithm successfully solves the coverage problem of IoTs with massive nodes and significantly extends the IoT lifespan.

INDEX TERMS 5G networks, Internet of Things, parallel computing, genetic algorithm.

I. INTRODUCTION

With the development and popularization of 5G networks, the applications of the Internet of Things (IoT) face new opportunities and challenges [1]–[6]. Sensor nodes (i.e., nodes) in an IoT often have no continuous power source, hence extending the lifespan of the IoT has always been a crucial issue. One possible way to solve the problem is to overlay more sensor nodes in the monitoring area and allow these nodes to be active or sleep alternately. A configuration of working nodes in the IoT lasts one timeframe, and then other nodes in the next configuration turn to be active for another timeframe. With the consumption of working nodes, the configuration continues to form a sequence until the IoT exhausts most of the sensor nodes, and the remaining nodes cannot meet the lower bound of the IoT coverage. Then the IoT lifespan ends up. Hence, the lifespan of an IoT is equal to the length of the configuration sequence of working nodes.

In 5G networks, calculating the optimum configuration sequence [7], [8] to extend the IoT lifespan will encounter the massive-node problem [9]. Fig. 1 shows a schematic diagram of 5G network systems. 5G networks use a short-range and high-frequency radio for communication to achieve a high transmission speed. As a result, the 5G network times the number of base stations compared with 4G networks. In 4G networks, each base station contains a network access server that is responsible for wireless devices accessing the backbone network. The network access server often works as the gateway to manage the local IoT. However, to reduce construction costs, the 5G network simplifies the network access server in the base station [9]. Instead, the network uses the data center to take charge of these network access servers. Hence the data center has to manage a large IoT that consists of many local IoTs and contains massive nodes. Meanwhile, the 5G network also promotes the popularity of the IoT and result in more IoT devices (i.e., nodes). According to the GSMA report, IoT connections (i.e., nodes) will reach almost 25 billion globally by 2025 [10].

The associate editor coordinating the review of this manuscript and approving it for publication was Adnan M. Abu-Mahfouz.

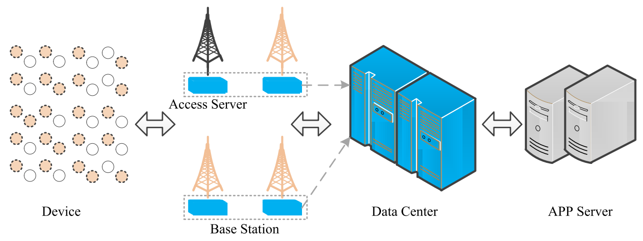

FIGURE 1. The framework of 5G networks.

Then, in some hotspots, the data center will manage millions of nodes.

The IoT coverage problem that is a selection problem for coverage-centric active nodes is already an NP-complete problem [11], and solving the problem in massive-node scenarios is often beyond the solving capacity of the existing algorithms. Usually, these algorithms need to reserve a series of possible solutions in the solving process to search for the global optimum solution. In massive-node scenarios, the number of possible solutions required in the solving process is huge. The algorithm will fail due to the inability to complete the calculation after a very long period.

Three requirements exist for the algorithm that is capable of solving the IoT coverage problem in massive-node scenarios. First, the algorithm should be able to degrade the scale of the problem and ensure to complete the computing operation within a limited period. Furthermore, solving the IoT coverage problem is a multi-objective programming problem. Hence the algorithm should take both network coverage and node redundancy into account and consider the impact of the current configuration of working nodes on the following configuration. Finally, the algorithm requires internal optimization so that the solving process can quickly evolve towards feasible solutions.

This paper presents a parallel genetic algorithm (PGA) using Hadoop to calculate the optimal configuration sequence for IoTs with massive nodes and finally extend the IoT lifespan. The key contributions are as follows:

- The parallel algorithm divides the coverage problem of IoTs with massive nodes into many small problems, degrades the problem scale, and then solves them using Hadoop [12] in parallel.
- The algorithm adopts the multi-objective programming-based genetic algorithm (MPGA) to solve the coverage problem. MPGA uses the fast non-dominated sorting [13] to optimize the IoT coverage and node redundancy; it implements the preferential selection of non-critical nodes to maximize the length of the configuration sequence of working nodes.
- The parallel algorithm uses uniform mutation and pruning operations to optimize the genetic algorithm internally and force its solving process to quickly converge toward feasible solutions.

The rest of the paper is organized as follows. Section II describes the background and related works for the coverage problem. Section III presents the framework of the parallel genetic algorithm. Section IV, V, and VI describe the detailed implementations of the algorithm. Experimental results are presented in Section VII. Finally, we conclude the paper in Section VIII.

II. BACKGROUND, RELATED WORKS AND MOTIVATION OF THIS WORK

A. IoT COVERAGE MODEL

The network coverage is a fundamental problem for constructing an IoT, and its value needs to be maximized or at least exceed a given threshold. In this way, the IoT can avoid a blind communication area and guarantee the quality of service (QoS) [14]–[16]. This work considers the certainty coverage where the monitoring area is determined, and the coverage of that area must exceed a given threshold [17]. At the same time, this work also uses the strategy that activates part nodes to cover the monitoring area as working nodes and turns off the remaining nodes into a low-power sleep state.

This work uses the coverage model in [17] to formulate the IoT coverage problem. Suppose an IoT works on a 2D monitoring area A, this area is constituted by $m \times n$ grids, and each grid is 1m $\times$ 1m. Let s_i be the i^{th} sensor node. Let num is the count of these nodes in the IoT, and $S = (s_1, s_2, \ldots, s_i, \ldots, s_{num})$ be the set of sensor nodes. Suppose the position of each node is known, and (x_i, y_i) stands for the coordinate of the s_i node. Suppose the perception range of a node is a circular region, the rand $\{x_i, y_i, r\}$ stands the effective perception circle of the s_i node where the node $s_i(x_i, y_i)$ is the center and r is the radius.

Assume that the communication radius r_c (i.e., the maximum communication distance between sensor nodes) is at least two times of the perception radius r, namely $r_c >= 2r$. In this way, if the sensor nodes can cover the monitoring area, the IoT can maintain its connectivity, and researchers no longer need to consider connectivity. Let $P_{cov}(x, y, s_i)$ be the condition whether the sensor node $s_i(x_i, y_i)$ covers the grid (x, y), as shown in Equation 1 [17].

$$P_{\text{cov}}(x, y, s_i) = \begin{cases} 1, & (x - x_i)^2 + (y - y_i)^2 \le r^2 \\ 0, & else \end{cases} \quad (1)$$

$$P_{\text{cov}}(x, y, C_j) = \begin{cases} 1, & \exists s_i \in C_j, P_{\text{cov}}(x, y, s_i) = 1 \\ 0, & else \end{cases} \quad (2)$$

Furthermore, let $P_{cov}(x, y, C_j)$ be the condition whether the j^{th} configuration of working nodes covers the grid (x, y), where C_j is a subset of the node set S and denotes these working nodes. Its value is calculated by Equation 2 [17].

Once a node s_i belongs to the j^{th} configuration of working nodes, and the node s_i satisfies Equation 1, the IoT covers the monitoring grid (x, y). Let $A_{area}(C_j)$ be the grid count covered by the configuration C_j, as shown in Equation 3 [17].

$$A_{area}(C_j) = \sum_{x=1}^{m} \sum_{y=1}^{n} P_{\text{cov}}(x, y, C_j) \quad (3)$$

So far, the existing model has not dealt with the redundancy of working nodes.

B. RELATED WORKS

Researchers have published [17]–[34] many techniques for the IoT coverage problem, but none of these techniques is proposed for the IoTs with massive nodes in 5G networks. Some researchers have proposed a series of meta-heuristic algorithms to maximize the IoT coverage, as shown in Table 1 [18]. A hybrid algorithm [19] uses simulated annealing technology to minimize the number of nodes for optimal IoT deployment. The EENPA algorithm [20] applies an energy-efficient node placement technology to cover the monitoring area and save network energy. The ABC algorithm [21] adopts artificial bee colony technology to increase IoT coverage. The PSO algorithm [22] integrates two-particle swarm optimizers to maximize coverage and extend the IoT lifespan in the 3D industrial area with barriers. The HS algorithm [23] uses a harmony search to maximize network coverage with minimal cost. The MADA-WOA algorithm [24] employs the whale optimization technique to optimize the dynamic deployment of sensor nodes. The Firefly algorithm [25] uses a firefly optimization technique to improve the coverage of mobile IoT. The CM-IA algorithm [26] applies an immune algorithm to improve the coverage of mobile IoT and reduce redundant area between nodes. Genetic algorithms are suitable for solving discrete problems and can find globally optimal solutions, so they are widely used for IoT coverage problems. For example, GA [27], MGA-RNP [28], and GA [29] optimize the number of relay nodes for K-connectivity IoTs, maximize network efficiency using a multi-objective fitness function, and realize K-coverage and K-connectivity IoT with minimum nodes, respectively. However, directly applying existing algorithms to the IoT coverage problem in massive-node scenarios will fail to solve the globally optimal solution because the large number of possible solutions required in the solving process prevents these algorithms from completing calculations within a limited period.

TABLE 1. **The meta-heuristic algorithms for IoT coverage [18].**

Name	Technology	Advantage
A hybrid algorithm[19]	Simulated annealing	Minimize the number of nodes for area and barrier coverage of IoTs
EENPA[20]	---	Achieve area coverage with minimum nodes
ABC[21]	Artificial bee colony	Increase coverage for monitoring area
PSO[22]	Swarm optimizers	Maximize network coverage and prolong lifespan of 3D area with obstacles
HS[23]	Harmony search	Maximize area coverage with minimum nodes
MADA-WOA[24]	Whale optimization	Maximize area coverage with minimum nodes
Firefly[25]	Firefly optimization	Increase area coverage of mobile IoT
CM-IA[26]	Immune algorithm	Increase area coverage of mobile IoT
GA[27]	Genetic algorithm	Achieve K-connectivity with less number of relay nodes
MGA-RNP[28]	Genetic algorithm	Maximize network efficiency using a multi-objective fitness
GA[29]	Genetic algorithm	Achieve K-coverage and K-connectivity IoT with minimum nodes

Some researchers have also developed many scheduling-based algorithms to extend the IoT lifespan. Calculating the optimum configuration sequence to extend the IoT lifespan emerges as a multi-objective programming problem. The algorithms should take both network coverage and node redundancy into account and maximum the length of the working nodes' configuration sequence as another goal. The rotation based heuristic cover algorithm [30] divides sensor node clusters into disjoint sets. It activates these sets one by one to extend the IoT lifespan, but it is time-consuming for large IoTs. The node self-scheduling algorithm [31] alternately schedules nodes to be sleep or active and maximizes the IoT coverage. The probing environment and adaptive sleeping algorithm [32] demands a working node to be sleep if another working node exists within the current node's detection range, thus reducing redundant nodes. However, the unbalanced energy consumption will prematurely exhaust some nodes and reduce the IoT lifespan. The energy-efficient coordination algorithm [33] constructs a backbone network to determine working nodes, but the backbone nodes will also prematurely exhaust their energy. The lightweight deployment-aware scheduling (LDAS) [34] algorithm demands the working node to sense other nodes within the detection region. Once the number of working nodes exceeds a threshold, these nodes are sleep to save energy. However, the algorithm causes severe redundancy of working nodes. The HCCVGA algorithm [7] schedules sensor nodes to be active/sleep to satisfy connectivity and coverage requirements well.

The IGA-BAC algorithm [17] emerges as one of the advanced methods to implement multi-objective programming for the IoT coverage problem. The algorithm adopts a single objective function (i.e., fitness function) for the IoT coverage and the used nodes. It then combines the improved genetic algorithm and the binary ant colony (IGA-BAC) algorithm to solve the problem [17]. However, degrading multi-objective programming into linear programming is not suitable for the IoT coverage problem. That is because the current two goals are mutually exclusive, and the appropriate single objective function corresponding to the optimum solution is usually unknown. Besides, none of these researches consider the impact of the current configuration of working nodes on the following configuration, and maximum the length of the configuration sequence as a goal for multi-objective programming.

C. MOTIVATION

In 5G networks, the algorithm for solving the IoT coverage problem will encounter the massive-node problem [35]–[37] and may have a cliff-like decline in its performance. For example, Fig. 2 presents the coverage that the genetic algorithm (GA) achieves when the different count of sensor nodes exist in an IoT. The experiments work on a 100m × 100m monitoring area, set the perception radius as 10m, and initialize the number of individuals in a generation (i.e., possible solutions in the solving process) as 60. The coverage firstly rises when the number of nodes increases due to the presence of more nodes in the IoT. When the number of nodes is 1000, the coverage reaches its peak, and the calculation time only takes several minutes. Then, the coverage falls when the number of nodes reaches 1500. When the number reaches 2000, the genetic algorithm fails as none of the individuals that meet the pruning conditions exists after several generations. If the algorithm dramatically increases the number of individuals in a generation, it takes tens of hours to get some solutions. However, in a larger IoT, the algorithm still fails as it cannot complete the calculation after a very long period. In massive-node scenarios, feasible solutions often occupy an extremely low ratio among all possible solutions. Many individuals with advantages in IoT coverage or node redundancy cannot meet the pruning conditions. Hence, the algorithm requires a huge number of individuals reserved in the solving process to search for globally optimal solutions. Meanwhile, the algorithm requires many generations to complete the evolution of initial individuals to feasible solutions. Therefore, the genetic algorithm is either early terminated as there is no individual remaining after individual pruning if the individuals in a generation are not large enough, or fails to complete its calculation within a limited period if the individuals in a generation are huge.

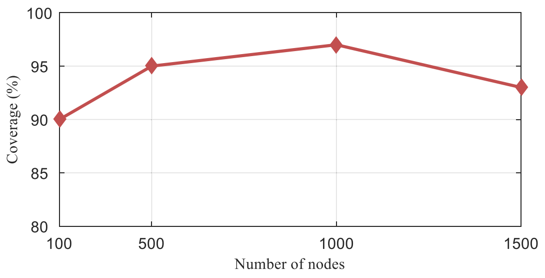

FIGURE 2. The coverage of the GA for different number of nodes.

Solving the coverage problem in parallel is feasible for IoTs with massive nodes in 5G networks. First, the perception area of a sensor node is much smaller than the monitoring area of an IoT, that is, whether the node is active only affects the local zone rather than the entire IoT. Hence partitioning the IoT into many zones (i.e., sub-IoTs) and solving their coverage problems in parallel are feasible. Second, the IoT has many redundant nodes in the scenarios of over-deploying and alternately activating nodes. Thus the solving algorithm can obtain feasible solutions from a group with a small number of nodes. Therefore, grouping nodes in the IoT is also feasible for the coverage problem.

III. PARALLEL GENETIC ALGORITHM

This work proposes a parallel genetic algorithm using Hadoop to extend the lifespan of IoTs with massive nodes in 5G networks. Fig. 3 presents an example of the parallel genetic algorithm. Since the data center in 5G networks takes over the functions of access servers in base stations, it manages the large IoT that is constituted by multiple IoTs corresponding to these base stations. Late, the data center implements partitioning operations to divide the large IoT into multiple sub-IoTs. Then the data center performs grouping operations on each sub-IoT if the sub-IoT still contains many nodes. Subsequently, this work proposes a genetic algorithm based on multi-objective programming for each node group. The algorithm takes the coverage goal and the redundancy goal into account, and produces a set of feasible solutions. Finally, the algorithm adopts a strategy of preferentially selecting non-critical nodes to determine the current configuration of working nodes, and maximum the sequence length of the working nodes' configurations.

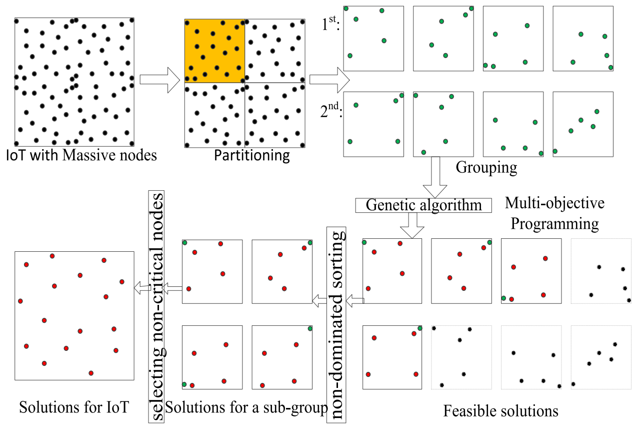

FIGURE 3. The example of the PGA using Hadoop.

Specifically, this work applies Hadoop [12] to calculate configurations of working nodes for each node group in parallel. This solving process is a multi-objective programming problem. First, a configuration of working nodes should maximize the coverage or at least meet the lower bound. We use the coverage rate to describe this goal.

Definition 1: The coverage rate of the j^{th} configuration of working nodes $R_{cov}(C_j)$ is the grid number covered by the working nodes in the configuration C_j, dividing the sum of grids in the monitoring area.

$$R_{\text{cov}}(C_j) = \frac{\sum_{x=1}^{m} \sum_{y=1}^{n} P_{\text{cov}}(x, y, C_j)}{m \times n} \tag{4}$$

Equation 4 calculates the value of the coverage rate. The coverage rate should be no less than the lower bound, $B_{low}(R_{cov})$, to guarantee the QoS, namely, $R_{cov}(C_j) \geq B_{low}(R_{cov})$.

The configuration should minimize the redundancy of working nodes, where the redundancy describes that the current grid is multiply covered by sensor nodes. Let the coverage degree $D_{cov}(x, y, C_j)$ be the number of times that the grid (x, y) is covered by the configuration C_j, as shown in Equation 5.

$$D_{\text{cov}}(x, y, C_j) = \sum_{i=1}^{num} (s_i \in C_j) \,\&\&\, (P_{\text{cov}}(x, y, s_i) = 1) \tag{5}$$

Let $P_{red}(x, y, C_j)$ the condition whether working nodes in the C_j configuration redundantly cover the grid (x, y). Equation 6 calculates the value of $P_{red}(x, y, C_j)$.

$$P_{red}(x, y, C_j) = \begin{cases} 1, & D_{\text{cov}}(x, y, C_j) > 1 \\ 0, & else \end{cases} \tag{6}$$

Definition 2: Redundancy rate $R_{red}(C_j)$ is the proportion of the grids that are redundantly covered by the configuration C_j in the sum of grids in the monitoring area.

$$R_{red}(C_j) = \frac{\sum_{x=1}^{m} \sum_{y=1}^{n} P_{red}(x, y, C_j)}{m \times n} \tag{7}$$

The redundancy rate is calculated in Equation 7. The two above goals are mutually exclusive, and their single objective function corresponding to the optimum solution is unknown.

Definition 3: The critical node is a sensor node that is required by multiple feasible solutions.

When the algorithm determines the current configuration of working nodes from feasible solutions, it should avoid selecting critical nodes. If these critical nodes run out of energy prematurely, the latter configuration will fail to reach the lower bound of the coverage due to lack of critical nodes. Let $N_{config}(C_j)$ be the number of critical nodes in the configuration C_j. The MPGA has to minimize $N_{config}(C_j)$.

Assume the configuration sequence of working nodes $(C_1, C_2, \ldots, C_J)$ continues till the last configuration C_J. Then activating all remaining nodes still cannot meet the coverage requirement. Hence the IoT lifespan is equal to the length of the configuration sequence, and Equation 8 describes the last goal.

$$Objective = \max(J), \quad J \text{ } in \text{ } (C_1, C_2, C_3, \ldots.., C_J) \tag{8}$$

Fig. 4 presents a flowchart of the parallel genetic algorithm to achieve the last goal and extend the IoT lifespan. This algorithm mainly contains three parts: the mapping-reduce process, the solving process, and the merging process. The following sections introduce these processes in detail.

IV. THE MAPPING-REDUCE PROCESS IN PARALLEL GENETIC ALGORITHM

In the mapping-reduce process, the parallel genetic algorithm performs partitioning and grouping operations to degrade the scale of the coverage problem for an IoT with massive nodes. First of all, the algorithm determines the size of sub-IoTs and then partitions the IoT into several sub-IoTs. On the one hand, if the size of each sub-IoT is less than ten times the perception radius, nodes in adjacent sub-IoTs have an apparent influence on the coverage of the current sub-IoT. That violates the assumption that a sensor node affects its local sub-IoT. On the other hand, a large-sized sub-IoT contains more nodes, which will lead to a rapid increase in the execution time. Eventually, the algorithm even cannot complete the calculation after a long period. Let N_{sub} the number of sub-IoTs after partitioning. For the efficiency of the parallel algorithms, this work sets the length and the width of each sub-IoT as ten times of the perception radius, and partitions the IoT into N_{sub} sub-IoTs.

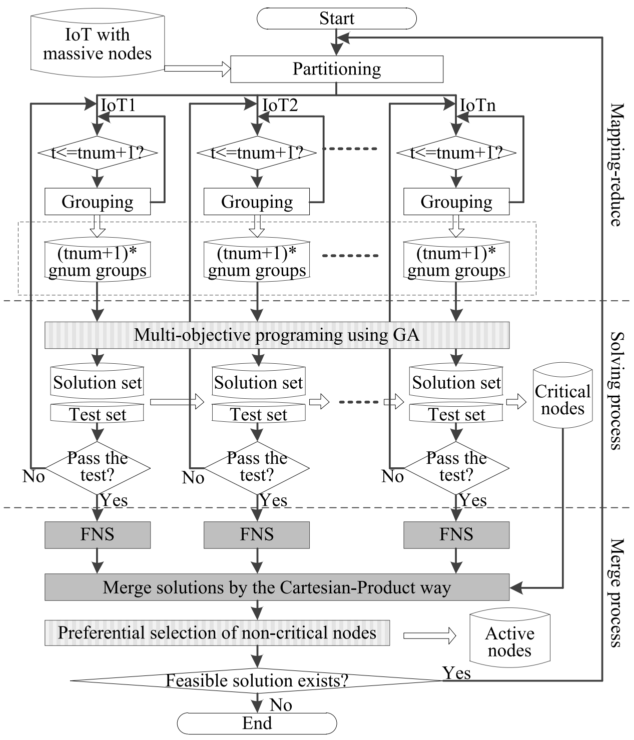

FIGURE 4. The framework of the PGA using Hadoop.

Second, let N_{group} the number of nodes of the sub-IoT divide the solvable number of nodes of the genetic algorithm. The algorithm divides nodes in a sub-IoT into N_{group} groups if the sub-IoT still contains too many nodes. In this process, the algorithm randomly distributes nodes of a sub-IoT, so it may distribute nodes in a particular solution into different groups and miss the solution. Let N_{time} the times of grouping operations. As a result, the algorithm employs $(N_{time} + 1)$ times of grouping operations, where N_{time} times of grouping operations aim to generate more groups to cover feasible solutions, and another time of grouping operation is used to check if other optimal solutions exist after N_{time} times of grouping operations.

In Fig. 4, the parallel algorithm first divides the IoT with massive nodes into several sub-IoTs. Later, the algorithm performs $(N_{time} + 1)$ times of grouping operations for the nodes in each sub-IoT to cover feasible solutions. In this way, the algorithm obtains a set of node groups whose number of nodes the genetic algorithm can handle. With partitioning and grouping, the algorithm maps the coverage problem for the IoT with massive nodes into many small problems, and the following step can solve them in parallel.

V. MULTI-OBJECTIVE PROGRAMMING-BASED GENETIC ALGORITHM

In the solving process, the parallel genetic algorithm applies a multi-objective programming-based genetic algorithm (MPGA) to search for feasible solutions of each group in parallel. The MPGA contains two parts: an improved genetic algorithm using fast non-dominated sorting [13] (called IGA-FNS), and preferentially selection of non-critical nodes. In the first part of MPGA, the algorithm implements the IGA-FNS to optimize coverage and redundancy for a given node group. In this work, we modified the source code of the genetic algorithm in Matlab to form the IGA-FNS algorithm. Specifically, we use the FNS algorithm [13] to replace the single objective function (i.e., the fitness function) for sorting individuals and adopt uniform crossover and individual pruning to optimize the genetic algorithm internally. First of all, the parallel algorithm checks if a node group's coverage meets the lower bound when it activates all nodes. If yes, the parallel algorithm starts the IGA-FNS algorithm; else, the node group cannot provide any feasible solution, and the parallel algorithm abandons the node group.

Fig. 5(a) presents the flowchart of the IGA-FNS, where one-bit models if a sensor node is active, and an N-bit pattern models a configuration of working nodes. Let *gen* the number of generations in the genetic algorithm. IGA-FNS generates the first generation of individuals by crossover and mutation and initializes the value of *gen* as two. Late, IGA-FNS merges the individuals of the current generation and its parent generation and forms a generation. Let *TG* be the generation interval. IGA-FNS performs individual pruning every *TG* generation. In the following, if individuals in the current generation are not sorted, IGA-FNS implements the FNS algorithm to sort these individuals according to their coverage and redundancy. Specifically, FNS firstly ranks the non-dominated individual that has either a higher coverage or a lower redundancy than any other individual. Then IGA-FNS calculates congestion, and finally selects individuals to form a parent generation; else, IGA-FNS performs uniform crossover to generate new individuals. Next, IGA-FNS also applies mutations and selections on these individuals. IGA-FNS continues until *gen* reaches its upper bound *Max*(*gen*). Finally, IGA-FNS provides a set of feasible solutions (i.e., configurations of working nodes) for the given node group.

A. FAST NON-DOMINATED SORTING

The IGA-FNS adopts the FNS algorithm [13] to optimize coverage and redundancy. Suppose two solutions x_1 and x_2 exist. We call x_1 dominates x_2 if x_1 is superior to x_2 for all objectives. If the solution x_1 is not dominated by any other solution, we call it a non-dominated solution. We also call the set of these solutions a non-dominated set.

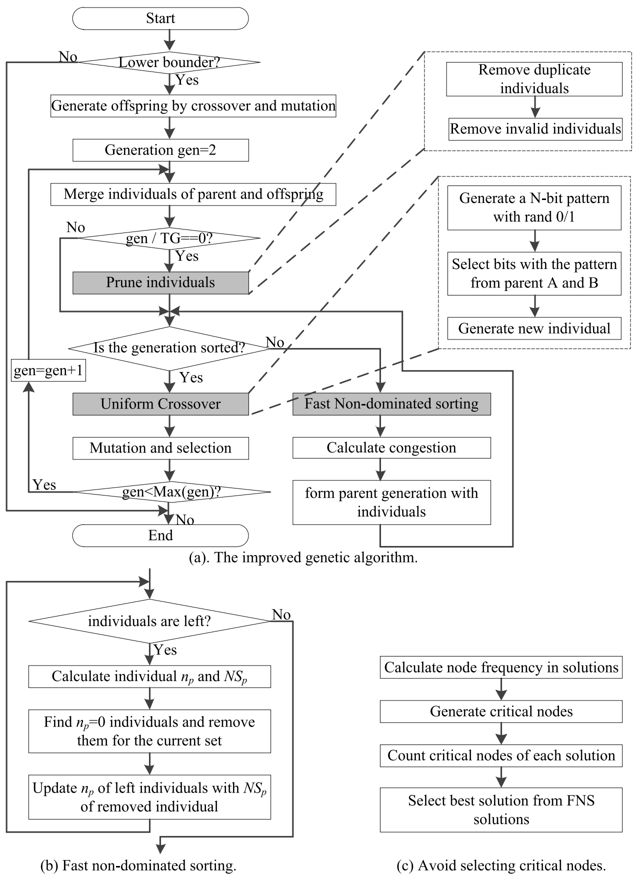
Start
Lower bounder?
No
Yes
Generate offspring by crossover and mutation
Generation gen=2
Merge individuals of parent and offspring
gen / TG==0?
No
Yes
Prune individuals
Is the generation sorted?
No
Yes
gen=gen+1
Uniform Crossover
Mutation and selection
gen<Max(gen)?
Yes
No
End
Fast Non-dominated sorting
Calculate congestion
form parent generation with individuals
Remove duplicate individuals
Remove invalid individuals
Generate a N-bit pattern with rand 0/1
Select bits with the pattern from parent A and B
Generate new individual
(a). The improved genetic algorithm.
individuals are left?
No
Yes
Calculate individual n_p and NS_p
Find n_p=0 individuals and remove them for the current set
Update n_p of left individuals with NS_p of removed individual
(b) Fast non-dominated sorting.
Calculate node frequency in solutions
Generate critical nodes
Count critical nodes of each solution
Select best solution from FNS solutions
(c) Avoid selecting critical nodes.

FIGURE 5. **The flowchart of multi-objective programming-based on genetic algorithm.**

The FNS algorithm performs multi-objective programming by searching the non-dominated set.

Fig. 5(b) presents the flowchart of the FNS algorithm. Let n_p and NS_p be the number of solutions dominating the current solution p and a set of solutions dominated by the current solution p, respectively. First, if individuals still exist, FNS goes to the next step; else, FNS sorts all individuals and ends up. Then, FNS calculates the value of n_p and NS_p for each individual. Specifically, FNS compares the coverage and the redundancy of individuals in the current generation (e.g., x and y). If the coverage and the redundancy of x are superior to those of y (i.e., x dominates y), FNS puts the individual y into the set NS_x, and the value N_y adds one. If x has higher coverage than y, but y has lower redundancy than x, they are non-dominated. Then FNS finds every individual whose N_p is equal to zero. These individuals belong to the non-dominated set, and they are superior to other individuals in terms of coverage and redundancy. Hence, FNS moves these individuals into the first level of the ranking results. Meanwhile, the algorithm analyzes the NS set of each moved individual (e.g., NS_x), finds every individual (e.g., y) that belongs to the set, and reduces the N_y value of the individual by one. After moving the individuals into the first level, FNS continues to search for another non-dominated set belonging to the next level.

For example, Fig. 6 describes that FNS sorts five individuals, where the coverage *CR* and the redundancy *R* are available. First, the fourth and fifth individuals are non-dominated by any other individual whose N_p is 0, so FNS sets them as the first level. After moving these individuals, FNS updates the N_p of the remaining individuals, and then two new non-dominated individuals emerge. In this way, FNS sorts these individuals level by level. Compared with the single-objective function (i.e., fitness function), the FNS algorithm reserves non-dominated solutions that are advantageous in terms of coverage or redundancy level by level. Hence FNS is more suitable to search for the globally optimal solutions for the IoT coverage problem.

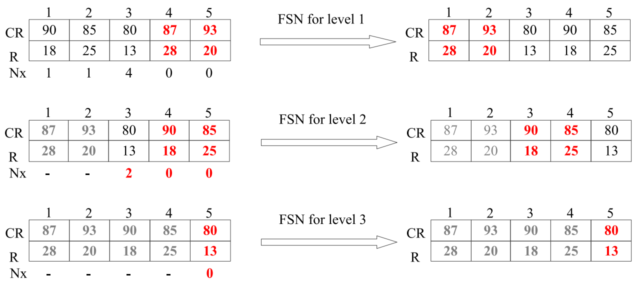

FIGURE 6. The example of fast non-dominated sorting.

B. UNIFORM CROSSOVER AND INDIVIDUAL PRUNING

The IGA-FNS uses uniform crossover and individual pruning to optimize the genetic algorithm internally and force its solving process to quickly converge toward feasible solutions. First, the algorithm adopts uniform crossover to extend the distribution of individuals. Assume two parent individuals A and B exist for crossover. In Fig. 5(a), IGA-FNS randomly generates an N-bit pattern. The number of zero in the pattern is equal to the number of one. Then the algorithm extracts the bits of the parent individuals A and B according to the pattern. As shown in Fig. 7, two 10-bit patterns stand for the parent individual A and B. The algorithm generates another 10-bit pattern for uniform crossover. As the first two bits of that pattern are one, the offspring C extracts the first two bits of the parent A. Then the next three bits are zero, so the offspring C extracts the next three bits of B. In this way, the parent A and parent B are distributed over C. The generated offspring individual is significantly distinguished from the original parent individuals. Uniform crossover extends the distribution of the individuals and ultimately makes the solving process to quickly converge towards feasible solutions.

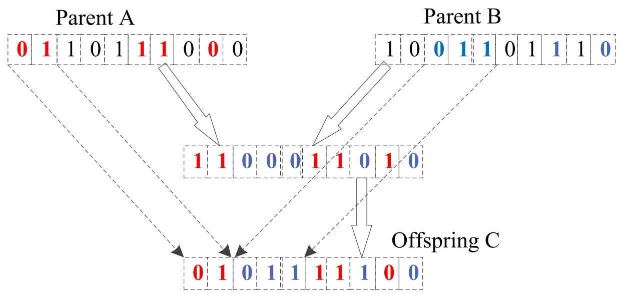

FIGURE 7. The example of uniform crossover.

Second, the IGA-FNS adopts individual pruning to remove redundant individuals in the genetic algorithm. As mentioned above, the genetic algorithm has limited individuals in a generation. However, the solving process inevitably generates redundant individuals. On the one hand, redundant individuals come from repetitive individuals. On the other hand, redundant individuals are also derived from invalid individuals. The ratio of working nodes to all nodes has a reasonable range. If the ratio of an individual exceeds the range, the redundancy is too severe, and the individual is useless to extend the IoT lifespan. If the ratio is below the range, too few working nodes cannot meet the coverage requirement, and the corresponding individual is also useless. In Fig. 5(a), IGA-FNS executes individual pruning every *TG* generation. Individual pruning removes repetitive individuals as well as invalid individuals whose ratios of working nodes are beyond the reasonable range and thereby frees computing resources occupied by the redundant individuals. Ultimately, individual pruning makes the solving process to quickly converge toward feasible solutions.

VI. MERGING SOLUTONS FOR THE CURRENT CONFIGURATION OF WORKING NODES

A. MERGING SOLUTIONS FOR THE ENTIRE IoT

The parallel genetic algorithm uses the FNS algorithm to merge solutions from small node groups and sub-IoTs and finally gets feasible solutions for the entire IoT with massive nodes. FNS reserves all non-dominated solutions to search for globally optimal solutions. Compared with any other solution, a non-dominated solution has either a higher coverage or a lower redundancy. Let $N_{feasible}$ the number of feasible solutions reserved after FNS. First, the algorithm merges feasible solutions from node groups in each sub-IoT and retains the first $N_{feasible}$ solutions for the following steps. Fig. 4 presents the workflow of this process. After the parallel algorithm implements $(N_{time} + 1)$ times of grouping operations for nodes in each sub-IoT and constitutes $(N_{time} + 1) \times N_{group}$ groups, it generates solutions for these groups using Hadoop in parallel. Then the algorithm collects feasible solutions of the $N_{time} \times N_{group}$ groups in the iterative part and then merges with the existing solution set (if it exists). Late the algorithm sorts these solutions using FNS, retains the first $N_{feasible}$ solutions (if its number is less than $N_{feasible}$, all solutions are retained) as a new solution set. Meanwhile, the algorithm also collects solutions of the remaining N_{group} groups as a test set. The algorithm then merges the test set into the solution set and compares the solution sets before and after this merging. Let R_{fs} the ratio of a part of feasible solutions to all feasible solutions. If the first $R_{fs} \times N_{feasible}$ solutions in the solution set remain the same, the algorithm goes to the following steps. Besides, R_{fs} is the checking ratio of the test process, and the setting of its value needs to balance the computing time and the performance (i.e., coverage and redundancy) of feasible solutions. Otherwise, the algorithm does not complete enough grouping operations and thus loses some optimal solutions. Therefore, it returns to the mapping-reduce process, executes ($N_{time}+1$) times of grouping operations again, and iteratively performs the previous steps.

For example, we implement an experiment on an IoT with 4000 nodes whose monitoring area is 100m $\times$ 100m, and the perception radius is 10m. In the experiment, we perform three grouping operations to generate solutions and use another time of grouping operation for testing. Because the algorithm solves at most 20 feasible solutions in a 1000-node IoT, we set $N_{feasible}$ as 20. As the current configuration determined by the parallel genetic algorithm mostly emerges within the first 20% of feasible solutions, we set R_{fs} as 20%. The generated solutions pass the testing and finally form a solution set. Meanwhile, we implement a single-pass grouping on the IoT and generate another solution set as a reference. Fig. 8 presents the coverage and redundancy of the first twenty solutions of these two solution sets. The solutions in the first set contain advantageous coverage or redundancy, and none of these solutions are dominated by the solutions in the second set. Hence the first solution set is generally better than the second set. The experimental results in Fig. 8 confirm that the multi-pass grouping operations with testing can obtain the optimal solutions for the entire IoT.

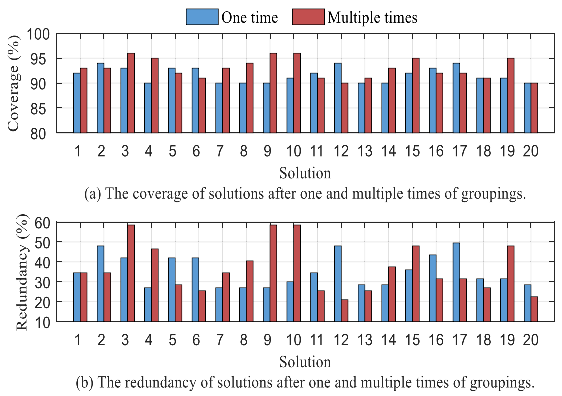

FIGURE 8. The coverage and redundancy of solutions after single-pass and multi-pass grouping operations.

Second, the algorithm merges feasible solutions from two adjacent sub-IoTs layer by layer, combines these two sub-IoTs, and finally generates the solution set for the entire IoT. Specifically, the algorithm merges feasible solutions from the two adjacent sub-IoTs by calculating the Cartesian product. Then the algorithm sorts these solutions using FSN and extracts the first $N_{feasible}$ solutions as the solution set for the IoT combined by these two sub-IoTs. Fig. 9 presents an example of merging solutions from two adjacent sub-IoTs. First, the parallel algorithm obtains four solutions for the first and the second sub-IoTs, respectively. Then, the algorithm obtains sixteen solutions for the combined IoT by calculating the Cartesian product. Finally, the algorithm sorts these solutions using the FSN algorithm and retains the first six solutions as the solution set. The merging process continues until it combines all sub-IoTs into one IoT, and then the algorithm obtains the solution set for the entire IoT.

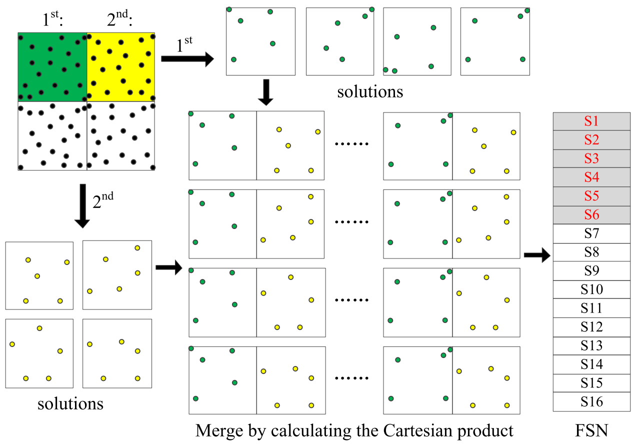

FIGURE 9. An example of merging solutions of two adjacent sub-IoTs.

B. PREFERENTIAL SELECTION OF NON-CRITICAL NODES

The parallel algorithm adopts the second part of MPGA (i.e., preferential selection of non-critical nodes) to determine the current configuration of working nodes. This process considers the impact of the current configuration on the following configuration and demands to minimize the number of critical nodes. Fig. 5(c) presents the flowchart of the preferential selection of non-critical nodes. First of all, the MPGA counts the occurrence number of each node in the feasible solutions that the parallel algorithm generates for each node group. Once the occurrence number of a node exceeds the threshold, it sets the node as a critical node. After the parallel algorithm merges solutions for the entire IoT and sorts the generated solutions according to their coverage and redundancy, MPGA counts the number of critical nodes that each generated solution contains. Finally, MPGA sets the solution with the least critical nodes as the current configuration. It is worth emphasizing that once multiple solutions contain the least critical nodes, MPGA selects the earliest solution according to its sorting order.

Fig. 10 presents an example of the preferential selection of non-critical nodes on a small IoT, where the IoT contains ten nodes in the monitoring area. First, MPGA generates four feasible solutions that meet the coverage and redundancy requirements and uses the FSN algorithm to sort these solutions. Then MPGA counts the occurrence number of each node in these solutions and finds that the second, third, and seventh nodes appear multiple times. These nodes are critical nodes. Fig. 10 shows that the critical nodes are indeed distributed in the center of the monitoring area. Hence MPGA identifies the critical nodes accurately. Next, MPGA finds

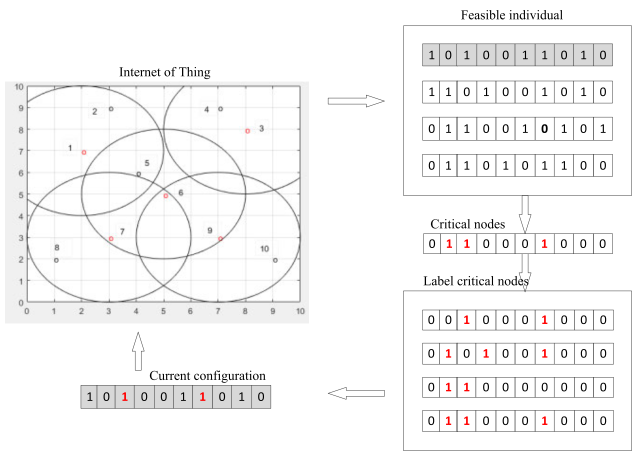

FIGURE 10. **The example of preferential selection of non-critical nodes.**

that the first and the third solutions contain the fewest critical nodes. Since the first solution dominates other ones, MPGA sets the first solution as the current configuration of working nodes. In Fig. 10, the selected configuration well satisfies these three goals of coverage, redundancy, and minimizing critical nodes. Therefore, MPGA can maximize the length of the working nodes' configuration sequence, and extend the IoT lifespan.

VII. EXPERIMENTS

In this section, we implement experiments to evaluate the performance of the MPGA and the parallel genetic algorithm, while Table 2 presents the experiment settings. First, as existing algorithms cannot handle the IoTs with massive nodes in 5G networks, we perform the MPGA on a small IoT and use the original GA and IGA-BAC as references. Besides, we directly apply IGA-FNS without the preferential selection of non-critical nodes as another reference in the experiments of IoT lifespan. The reference is used to demonstrate the impact of the preferential selection of non-critical nodes on the IoT lifespan. The small IoT covers a 100m × 100m monitoring area, and its lower coverage bound is 90%. The sensor nodes are randomly distributed in the IoT. Assume each sensor node contains ten energy units, and can alternately be active or sleep. The number of sensor nodes ranges from 100 to 200. Meanwhile, the perception radius is set as 10m in the beginning, and then gradually increases to 25m to simulate the scenarios that the sensor nodes are overplaced. Finally, the experiment sets the maximum individual number in a generation as 60. It then evaluates the coverage, the number of required nodes, the CPU time, as well as the lifespan and remaining nodes in the final IoT on a PC with an Intel i5 CPU and 4GB memory.

Second, we perform the parallel genetic algorithm on the IoT with massive nodes. The IoT covers a 400m × 400m monitoring area. The parallel genetic algorithm sets the size of each sub-IoT as 100m × 100m and partitions an IoT into 16 sub-IoTs. The number of sensor nodes ranges from 8000 to 64,000, and the number of nodes in a sub-IoT ranges from 500 to 4000. When the parallel algorithm applies grouping operations, the maximum number of nodes in a group is 1000. Other experimental settings are shown in Table 2. Meanwhile, we perform the genetic algorithm to the IoTs in two other ways and use them as references. The first is the direct genetic algorithm (DGA) that uses a genetic algorithm on the IoT directly. The second is MPGA after simple partitioning (MPGA-P) that divides an IoT into 16 sub-IoTs, runs the IGA-FNS on each sub-IoT, and sets the shortest lifespan of the sub-IoTs as the ultimate lifespan. Finally, experiments calculate the coverage and the ultimate lifespan of these methods on a server with two Intel E5 CPU and 64GB memory.

TABLE 2. **Experiment settings.**

IoT type	Small IoT	IoT with massive nodes
Software Platform	Matlab	Hadoop
Hardware Platform	Intel i5 CPU and 4GB memory	Two Intel E5 CPU and 64GB memory
Monitoring area	100m × 100m	400m × 400m
Size of sub-IoT	--	100m × 100m
Coverage bound	90%	90%
Number of nodes	From 100 to 200	From 8000 to 64000
Maximum nodes in a group	--	1000
Perception radius	From 10m to 25m	10m
Energy units in a node	10	1
Individual number in a generation	60	60
Maximum generations	From 50 to 150	100

A. MULTI-OBJECTIVE PROGRAMMING-BASED GENETIC ALGORITHM ON SMALL IoTs

1) OPTIMAZATION OF COVERAGE AND REDUNDANCY

The performance of the IGA-FNS outperforms the original GA and IGA-BAC in terms of coverage and redundancy. First, IGA-FNS achieves better coverage than the original GA and IGA-BAC regardless of the generation number. The original GA often contains limited individuals involved in a generation. If these individuals are similar and concentrated in a small part of the solution space, the algorithm requires more generations to search for feasible solutions. Or, if many repetitive individuals squeeze the computing resource, that will also degrade the performance of the algorithm. The IGA-BAC attempts to improve the IoT coverage by combining two advanced algorithms but does not strike the reason to limit the coverage. IGA-FNS implements uniform crossover and individual pruning to optimize the IoT coverage. The experimental results demonstrate the internal optimization of the genetic algorithm hits the root that limits the improvement of the IoT coverage.

Second, the required nodes of IGA-FNS reduces significantly compared with those of the original GA and

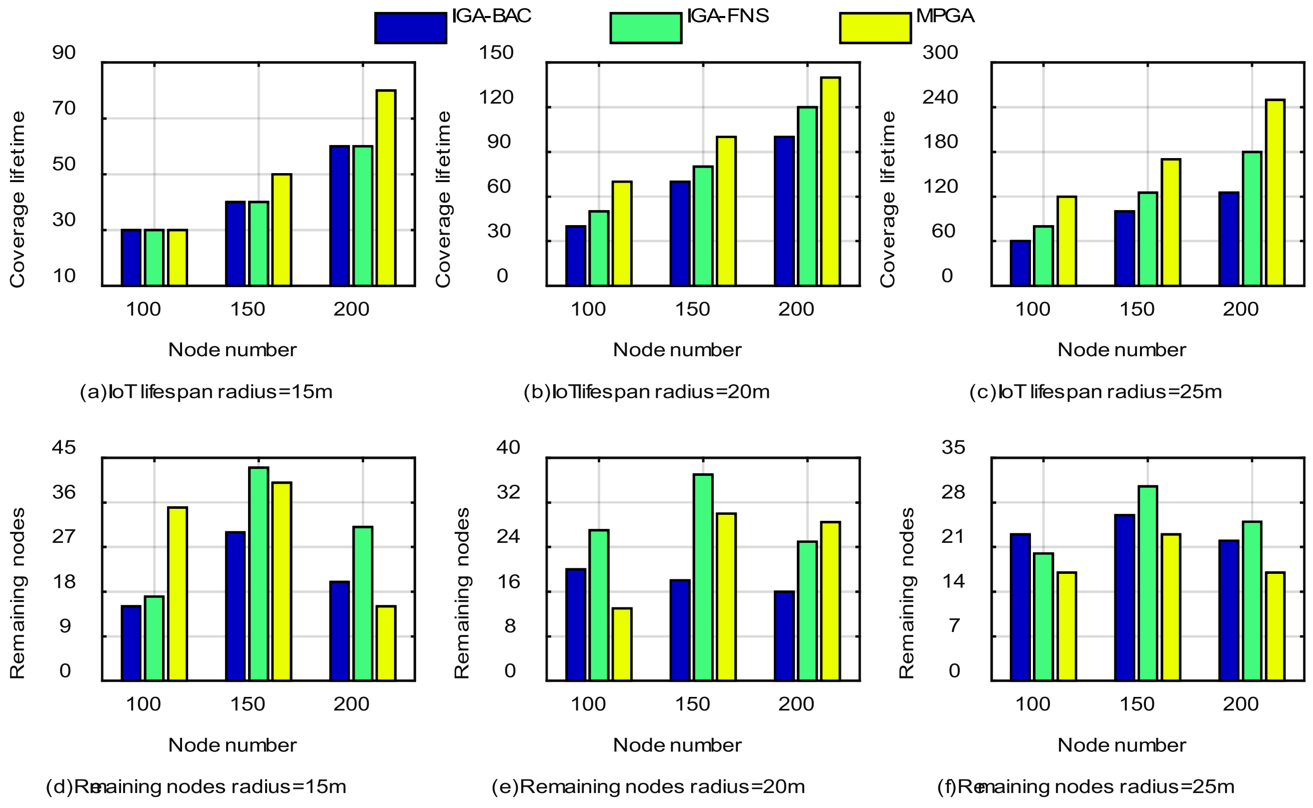

FIGURE 11. **The IoT coverage lifespan and remaining nodes under different perception radiuses.**

IGA-BAC in Table 3. IGA-FNS uses the FNS algorithm to sort individuals according to both of the coverage and the redundancy. When IGA-FNS is going to remove individuals, it removes the individuals that are sorted backward but retains the individuals ranked first. In this way, the algorithm reserves all individuals that contain advantageous coverage or redundancy. As a comparison, both of the original GA and IGA-BAC form a single objective function (i.e., fitness function) using the coverage and the node usage to select individuals. However, as the single objective function cannot accurately reflect the relationship between coverage and redundancy, the method that degrades multi-objective programming into linear programming is not suitable for the IoT coverage problem. Besides, Table 3 shows the required nodes of IGA-FNS after 50 and 100 generations are lower than those of the other methods after 100 and 150 generations. The result also confirms that uniform variation and individual pruning force the solving process to quickly converge towards feasible solutions.

Third, the CPU times of IGA-FNS are far less than those of the original GA and IGA-BAC regardless of the generation number of the genetic algorithm. That is because the FNS algorithm itself has superior performance and low time complexity. Specifically, the rule of uniform crossover is also simple, and IGA-FNS performs individual pruning every *TG* generation. The original GA contains many repetitive and invalid individuals, which result in extra calculation time.

TABLE 3. **The coverage rate, The number of nodes, and CPU time of IGA-BAC and IGA-FNS under different generations.**

gen	GA			IGA-BAC			IGA-FNS		
	R_{cov} (%)	num	time (%)	R_{cov} (%)	num	time (%)	R_{cov} (%)	num	time (s)
50	93.0	65	11	93.0	63	13	94.4	52	6
100	94.3	59	21	95.2	56	26	95.5	50	14
150	95.6	53	31	96.5	51	40	96.7	49	20

IGA-BAC combines two complex algorithms and thereby results in higher CPU time.

2) EXTENDING IoT LIFESPAN

The MPGA significantly extends the IoT lifespan in Fig. 11(a)-11(c). First, IGA-FNS and MPGA become more effective in extending the IoT lifespan when the experiments intensify the condition that sensor nodes are overplaced. In some cases, MPGA even doubles the IoT lifespan compared with the lifespan of IGA-BAC. That is because IGA-FNS and MPGA achieve a collaborative optimization in terms of coverage and redundancy, and save many redundant nodes. Hence, they can use these nodes in the following configurations. Furthermore, the IoT lifespan of MPGA is also higher than that of IGA-FNS. That is because MPGA considers the impact of the current configuration on the following configuration, and thus maximizes the length of the configuration sequence of working nodes.

The preferential selection of non-critical nodes also optimizes the node usage in the IoT. In the beginning, the condition that sensor nodes are over-placed is relatively mild, the IoT lifespans of the three algorithms are not much different, but the MPGA contains the most remaining nodes in Fig. 11(d). When experiments intensify the condition of over-placed nodes, this strategy ensures to activate sensor nodes evenly and alternately instead of frequently activating critical nodes. Hence the MPGA generates a longer sequence of configurations of working nodes. Besides, the remaining nodes of the MPGA are also minimal in most cases of Fig. 11(e) and 11(f). In conclusion, the MPGA outperforms the existing method in terms of coverage, the number of nodes, and computing time, and thus it significantly extends the IoT lifespan.

B. PARALLEL GENETIC ALGORITHM ON IoTs WITH MASSIVE NODES

The experimental results demonstrate that the parallel genetic algorithm can solve the coverage problem in the IoT with massive nodes in 5G networks and maximize its lifespan. First, advanced genetic algorithms cannot directly solve the coverage problems of the IoT with massive nodes. In the experiments, the nodes are at least 8000 nodes, and the number of possible individuals of the genetic algorithm steeply reaches the power of two to eight thousand. That exceeds the solving capacity of the genetic algorithm. Fig. 12 presents that even when the number of nodes is only 8000, DGA cannot solve any feasible solution, and the IoT lifespan of DGA is zero. That is because the number of possible individuals in the genetic algorithm reaches the astronomical scale, and the ratio of individuals that meet the pruning conditions is extremely low. After individual crossover and mutation in the population, the vast majority of new individuals did not meet the pruning conditions. Meanwhile, the number of individuals in a generation is only 60. Hence, after several generations of iterations, the genetic algorithm will terminate because no legal individual exists. Therefore, directly applying the genetic algorithm to solve the coverage problem of the IoT with massive nodes in 5G networks is not feasible.

Second, when the number of nodes is no more than 16000, and the number of nodes in a sub-IoT is no more than 1000, the MPGA-P can solve this IoT coverage problem, but its final lifespans are less than those of the parallel genetic algorithm in Fig. 12(a). Specifically, in the IoT with 8000 or 16000 nodes, the MPGA-P has similar coverage to the PGA, but MPGA-P requires more nodes than PGA. In Fig. 12(b)-(c), MPGA-P consumes more nodes at any timeframe than PGA. Three reasons lead to the result. First, although MPGA-P partitions the IoT into many sub-IoTs, the method completely ignores optimizing feasible solutions in adjacent sub-IoTs. In contrast, the parallel genetic algorithm enumerates all combinations of feasible solutions in two adjacent sub-IoTs by calculating the Cartesian product. Second, the parallel algorithm uses the FNS algorithm to optimize the coverage and redundancy of solutions for two adjacent sub-IoTs, and reserves more nodes for latter configurations of working nodes. Third, PGA implements the strategy of preferentially selecting non-critical nodes, and maximizes the length of the configuration sequence of working nodes.

Third, when the node scale of the IoT exceeds 16,000, only parallel genetic algorithms can calculate the lifespan of the IoTs with massive nodes in 5G networks. That is because each sub-IoT will contain more than a thousand nodes, and that exceeds the solve capacity of the genetic algorithm. The PGA further divides nodes into several groups whose number of nodes should be no more than one thousand, so the parallel algorithm solves the coverage problem of the IoT with 64,000 nodes. Furthermore, the algorithm performs multiple-pass grouping operations and then compares the feasible solutions with those in the test set. If other better solutions exist, the algorithm continues to perform the above steps iteratively. Consequently, the final solution approaches the optimum solution of the current sub-IoT.

Fourth, the parallel genetic algorithm using the divide-and-conquer idea properly degrades the time complexity of the coverage problem in the IoT with massive nodes in 5G networks. In the mapping-reduce process, the time complexity of partitioning and grouping operations is $O(num)$. In the solving phase, the time complexity of the genetic algorithm using FSN for each group is $O((num/(N_{group} \times N_{sub}))^2)$ according to work [13]. In the merging phase, the time complexity is $O(log_2(N_{sub}) \times (N_{feasible}^2)^2 \times num)$. Finally, the total cycles of the above processes, namely the IoT lifespan, is proportional to the number of sensor nodes (num). Since the variable num is much larger than other variables, the time complexity of the entire algorithm is nearly $O(num^3)$. Therefore, the algorithm only takes 36 hours to calculate a configuration sequence of working nodes for the IoT with 64000 nodes. Furthermore, this algorithm has good scalability and is applicable to the IoT with many more nodes. That is because the major process of the algorithm (i.e., solving the coverage problem) can

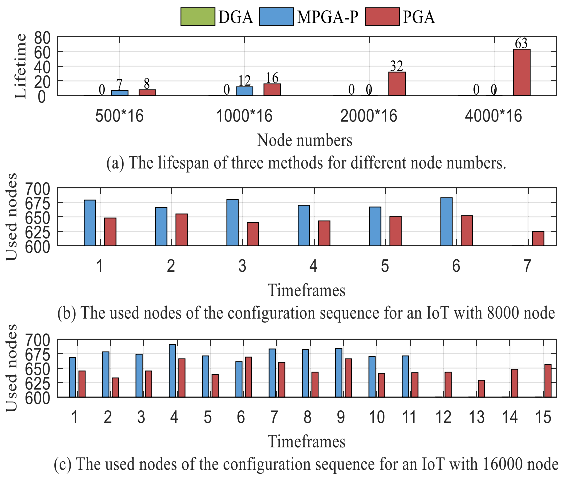

FIGURE 12. The lifespan and used nodes of different methods on IoTs with massive nodes.

be completely parallel with many computers existing in the data center. In conclusion, the parallel genetic algorithm can solve the coverage problem of IoTs with massive nodes in 5G networks and maximize the lifespan of these IoTs.

VIII. CONCLUSION

In this paper, we develop a parallel genetic algorithm for the coverage problem of IoTs with massive nodes in 5G networks and extend the IoT lifespan. Specifically, the parallel algorithm performs partitioning and grouping operations to degrade the coverage problem of a large IoT into many small problems. The algorithm then adopts the multi-objective programming-based genetic algorithm (MPGA) to solve the coverage problem. MPGA uses the fast non-dominated sorting to optimize the IoT coverage and node redundancy; it implements the preferential selection of non-critical nodes to maximize the length of the configuration sequence of working nodes. Finally, the parallel genetic algorithm uses uniform mutation and individual pruning to optimize the genetic algorithm internally and force its solving process to quickly converge toward feasible solutions. The experimental results confirm that the MPGA outperforms the existing work on small IoTs in terms of coverage, the number of nodes, computing time, and IoT lifespan. They also demonstrate that the parallel genetic algorithm successfully solves the coverage problem of the IoT with massive nodes in parallel, and significantly extends the IoT lifespan.

REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Comput. Netw.*, vol. 38, no. 4, pp. 393–422, 2002.
[2] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Comput. Netw.*, vol. 52, no. 12, pp. 2292–2330, 2008.
[3] A. Gupta and R. K. Jha, "A survey of 5G network: Architecture and emerging technologies," *IEEE Access*, vol. 3, pp. 1206–1232, 2015.
[4] L. Liu, "An intelligent optimization approach to non-stationary interference suppression for wireless networks," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 2, pp. 452–459, Mar. 2019.
[5] P. Zhang, M. Zhou, and X. Wang, "An intelligent optimization method for optimal virtual machine allocation in cloud data centers," *IEEE Trans. Autom. Sci. Eng.*, early access, Mar. 9, 2020, doi: 10.1109/TASE.2020.2975225.
[6] R. Elhabyan, W. Shi, and M. St-Hilaire, "Coverage protocols for wireless sensor networks: Review and future directions," *J. Commun. Netw.*, vol. 21, no. 1, pp. 45–60, Feb. 2019.
[7] J. N. Al-Karaki and A. Gawanmeh, "The optimal deployment, coverage, and connectivity problems in wireless sensor networks: Revisited," *IEEE Access*, vol. 5, pp. 18051–18065, 2017.
[8] D. Qin, J. Ma, Y. Zhang, P. Feng, P. Ji, and T. M. Berhane, "Study on connected target coverage algorithm for wireless sensor network," *IEEE Access*, vol. 6, pp. 69415–69425, 2018.
[9] L. Chettri and R. Bera, "A comprehensive survey on Internet of Things (IoT) toward 5G wireless systems," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 16–32, Jan. 2020.
[10] GSMA Report. [Online]. Available: https://www.gsma.com/newsroom/press-release/new-gsma-study-5g-to-account-for-15-of-global-mobile-industry-by-2025/
[11] Y. Zou and K. Chakrabarty, "A distributed coverage- and connectivity-centric technique for selecting active nodes in wireless sensor networks," *IEEE Trans. Comput.*, vol. 54, no. 8, pp. 978–991, Aug. 2005.
[12] Hadoop. [Online]. Available: https://hadoop.apache.org/releases.html
[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[14] H. M. Ammari, "On the problem of k-coverage in mission-oriented mobile wireless sensor networks," *Comput. Netw.*, vol. 56, no. 7, pp. 1935–1950, May 2012.
[15] E. Lattanzi and A. Bogliolo, "Virtual sense: A Java-based open platform for ultra-low-power wireless sensor nodes," *Int. J. Distrib. Sensor Netw.*, vol. 8, no. 11, 2012, Art. no. 154737.
[16] W. Chen, S. Chen, and D. Li, "Minimum-delay POIs coverage in mobile wireless sensor networks," *EURASIP J. Wireless Commun. Netw.*, vol. 2013, no. 1, p. 262, Dec. 2013.
[17] J. Tian, M. Gao, and G. Ge, "Wireless sensor network node optimal coverage based on improved genetic algorithm and binary ant colony algorithm," *EURASIP J. Wireless Commun. Netw.*, vol. 2016, no. 1, p. 104, Dec. 2016.
[18] M. Farsi, M. A. Elhosseini, M. Badawy, H. A. Ali, and H. Z. Eldin, "Deployment techniques in wireless sensor networks, coverage and connectivity: A survey," *IEEE Access*, vol. 7, pp. 28940–28954, Feb. 2019.
[19] Y. E. Khamlichi, A. Tahiri, A. Abtoy, I. Medina-Bulo, and F. Palomo-Lozano, "A hybrid algorithm for optimal wireless sensor network deployment with the minimum number of sensor nodes," *Algorithms*, vol. 10, no. 3, p. 80, Jul. 2017.
[20] K. Y. Bendigeri and J. D. Mallapur, "Energy aware node placement algorithm for wireless sensor network," *Adv. Electron. Electr. Eng.*, vol. 4, no. 6, pp. 541–548, 2014.
[21] C. Öztürk, D. Karaboğa, and B. Görkemli, "Artificial bee colony algorithm for dynamic deployment of wireless sensor networks," *Turkish J. Elect. Eng. Comput. Sci.*, vol. 20, no. 2, pp. 255–262, 2012.
[22] B. Cao, J. Zhao, Z. Lv, X. Liu, X. Kang, and S. Yang, "Deployment optimization for 3D industrial wireless sensor networks based on particle swarm optimizers with distributed parallelism," *J. Netw. Comput. Appl.*, vol. 103, pp. 225–238, Feb. 2018.
[23] O. M. Alia and A. Al-Ajouri, "Maximizing wireless sensor network coverage with minimum cost using harmony search algorithm," *IEEE Sensors J.*, vol. 17, no. 3, pp. 882–896, Feb. 2017.
[24] R. Özdağ and M. Canayaz, "A new dynamic deployment approach based on whale optimization algorithm in the optimization of coverage rates of wireless sensor networks," *Eur. J. Technic*, vol. 7, no. 2, pp. 119–130, Dec. 2017.
[25] E. Tuba, M. Tuba, and M. Beko, "Mobile wireless sensor networks coverage maximization by firefly algorithm," in *Proc. 27th Int. Conf. Radioelektronika (RADIOELEKTRONIKA)*, Apr. 2017, pp. 1–5.
[26] M. Abo-Zahhad, S. M. Ahmed, N. Sabor, and S. Sasaki, "Coverage maximization in mobile wireless sensor networks utilizing immune node deployment algorithm," in *Proc. IEEE 27th Can. Conf. Electr. Comput. Eng. (CCECE)*, May 2014, pp. 1–6.
[27] S. K. Gupta, P. Kuila, and P. K. Jana, "Genetic algorithm for K-connected relay node placement in wireless sensor networks," in *Proc. 2nd Int. Conf. Comput. Commun. Technol.* New Delhi, India: Springer, 2016, pp. 721–729.
[28] J. George and R. M. Sharma, "Relay node placement in wireless sensor networks using modified genetic algorithm," in *Proc. 2nd Int. Conf. Appl. Theor. Comput. Commun. Technol. (iCATccT)*, Jul. 2016, pp. 551–556.
[29] S. K. Gupta, P. Kuila, and P. K. Jana, "Genetic algorithm approach for k -coverage and m -connected node placement in target based wireless sensor networks," *Comput. Electr. Eng.*, vol. 56, pp. 544–556, Nov. 2016.
[30] S. Slijepcevic and M. Potkonjak, "Power efficient organization of wireless sensor networks," in *Proc. ICC. IEEE Int. Conf. Commun. Conf.*, Helsinki, Finland, Jun. 2001, pp. 472–476.
[31] D. Tian and N. D. Georganas, "Location and calculation-free node-scheduling schemes in large wireless sensor networks," *Ad Hoc Netw.*, vol. 2, no. 1, pp. 65–85, Jan. 2004.
[32] F. Ye, G. Zhong, J. Cheng, S. Lu, and L. Zhang, "PEAS: A robust energy conserving protocol for long-lived sensor networks," in *Proc. 23rd Int. Conf. Distrib. Comput. Syst.*, Pairs, France, Nov. 2002, pp. 200–201.
[33] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks," *Wireless Netw.*, vol. 8, no. 5, pp. 481–494, 2002.
[34] K. Wu, Y. Gao, F. Li, and Y. Xiao, "Lightweight deployment-aware scheduling for wireless sensor networks," *Mobile Netw. Appl.*, vol. 10, no. 6, pp. 837–852, 2005.
[35] M. Huang, K. Zhang, Z. Zeng, T. Wang, and Y. Liu, "An AUV-assisted data gathering scheme based on clustering and matrix completion for smart ocean," *IEEE Internet Things J.*, early access, Apr. 15, 2020, doi: 10.1109/JIOT.2020.2988035.

[36] H. Teng, K. Ota, A. Liu, T. Wang, and S. Zhang, ''Vehicles joint UAVs to acquire and analyze data for topology discovery in large-scale IoT systems,'' *Peer-Peer Netw. Appl.*, pp. 1–24, Feb. 2020.

[37] J. Tan, W. Liu, T. Wang, M. Zhao, A. Liu, and S. Zhang, ''A high-accurate content popularity prediction computational modeling for mobile edge computing using matrix completion technology,'' *Trans. Emerg. Telecommun. Technol.*, p. e3871, Jan. 2020.

YING ZHANG (Member, IEEE) received the B.S. degree in computer science from Harbin Engineering University, Harbin, China, in 2006, and the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2011. He completed one-year postdoc in the Embedded Systems Laboratory, Linkoping University, Sweden. He also worked as a Visiting Scholar with the ECE Department, Duke University, USA, in 2016. He is currently an Associate Professor with the School of Software Engineering, Tongji University. He has served on several program committees of international conferences and symposiums, including IEEE Asia and South Pacific Design Automation Conference and IEEE Asian Test Symposium. His research interests include signal integrity, reliable design of network-on-chip, and wireless sensor networks. He is a member of CCF.

WEIHONG YU is currently pursuing the bachelor's degree in software engineering with the School of Software Engineering, Tongji University. He is also a Research Assistant with the Laboratory of Dependable Computing and Communications, Tongji University. His current research interests include parallel computing, machine learning, and data mining.

XIAODONG CHEN received the master's degree in software engineering from the School of Software Engineering, Tongji University. He worked as a Research Assistant with the Laboratory of Dependable Computing and Communications, Tongji University. He is currently working with Meituan Company. His research interests include wireless sensor networks, the Internet of Things, and computer networks.

JIANHUI JIANG received the B.E., M.E., and Ph.D. degrees in 1985, 1988, and 1999, respectively. He is currently a Full Professor of computer science and technology and the Vice Dean of the School of Software Engineering, Tongji University. He is also the Vice Director of the Technical Committee on Fault-tolerant Computing, Chinese Computer Federation (CCF). He has served on several program committees of national or international symposiums or workshops, including IEEE Pacific Rim International Symposium on Dependable Computing, IEEE Asian Test Symposium, IEEE Workshop on RTL and High-Level Testing. He has coauthored two books and published more than 200 technical articles. His current research interests include dependable systems and networks, software reliability engineering, VLSI/SoC testing, and fault-tolerance. He is a Senior Member of CCF.