GAMA: Genetic Algorithm for *k*-Coverage and Connectivity with Minimum Sensor Activation in Wireless Sensor Networks

Syed F. Zaidi[1], Kevin W. Gutama[2], and Habib M. Ammari[3](✉)

[1] Kean University, Union, NJ 07083, USA
zaidisye@kean.edu

[2] New Jersey Institute of Technology, Newark, NJ 07102, USA
kg567@njit.edu

[3] Texas A&M University - Kingsville, Kingsville, TX 78363, USA
habib.ammari@tamuk.edu

Abstract. In wireless sensor networks, ensuring k-coverage and connectivity is crucial in order to efficiently gather data and relay it back to the base station. We propose an algorithm to achieve k-coverage and connectivity in randomly deployed wireless sensor networks while minimizing the number of active sensors. It has been shown that selecting a minimum set of sensors to activate from an already deployed set of sensors is NP-hard. We address this by using a genetic algorithm that efficiently approximates a solution close to the optimal solution. The algorithm works by selecting random solutions and mutating them, retaining only the best solutions for the next generation until it converges to a near-optimal solution. We examine the time complexity of our approach and discuss possible optimizations. Our simulation results show that our approach works consistently across different types of wireless sensor networks and for different degrees of required coverage.

Keywords: Wireless sensor networks · k-coverage · Connectivity · Sensor selection · Genetic algorithm

1 Introduction

A wireless sensor network (WSN) is a network consisting of wireless sensors capable of measuring various environmental conditions. These sensors are deployed in predetermined patterns or placed randomly within a target region, enabling a comprehensive view of the environment and valuable data collection. WSNs face the challenge of achieving adequate coverage, connectivity, and energy efficiency. The coverage problem entails that the target region is covered by at least one sensor, while the k-coverage problem focuses on covering each point with at least k sensors, crucial for fault tolerance. Connectivity is vital for relaying information to the base station.

To optimize the network's lifetime, it is essential to manage sensor states actively or inactively. All sensors operating simultaneously lead to energy waste, and redundant data collection. Balancing active and inactive sensors is challenging while maintaining connectivity and coverage. Randomly deployed sensor networks pose additional challenges. Uniform deployment is often impractical, and environmental factors can lead to sensor movement. Applications may require random or near-random deployments, necessitating optimal sensor placement.

The problem of finding the minimum number of sensors for k-coverage in randomly deployed networks is NP-hard, making an exact solution computationally infeasible. Genetic algorithms (GA), inspired by natural selection, offer an approximation approach. GAs generate a population of solutions, evaluate their proximity to the optimal solution, and iteratively improve them through mutation. Our GA is designed for static sensors, as mobility consumes more energy. We identify optimal sensor locations for k-coverage and connectivity while retaining network lifetime. Inactive sensors can activate as active sensors deplete energy.

2 Related Work

Yang et al. [1] establishes that selecting the minimum set of active sensors in a randomly deployed network to achieve k-coverage is NP-hard. Previous research efforts have tackled the k-coverage problem by introducing mobile sensors capable of moving to areas within the network where coverage is lacking. In [2], the authors propose a GA approach that utilizes mobile sensors to optimize coverage. This differs from our GA as we only consider static, pre-deployed sensors.

Hurizan and Kuila [3] investigate the activation of a specific set of nodes instead of deploying all nodes within the network. They employ a GA approach to assess the minimum selection of nodes required for full coverage, connectivity, and energy optimization. The main distinction between our approach and [3] lies in the integration of mobile sensors, which prevents the genetic algorithm from frequent reactivation. This prolongs the network's lifespan and effectively addresses potential environmental challenges that may arise within the network.

3 Preliminaries

This section provides an introduction to some of the terminology and notation used in our explanation of the genetic algorithm. Additionally, we outline some assumptions relating to the coverage model.

3.1 Key Terminology and Notation

The following are key definitions:

Definition 1. Sensors and points - A sensor is denoted by S and a point is denoted by p. The total number of sensors in a network is denoted by N and the total number of points in a network is denoted by P.

Definition 2. Sensing and communication range - The sensing radius of a sensor S is denoted by r and the communication radius is denoted by c. The sensing range and communication range of a sensor is the area formed by the circular disk centered at the sensor with radius r and c, respectively. The sensing range of a sensor S is denoted by S_r and the communication range is denoted by S_c.

Definition 3. Distance - The distance between a sensor S and a point p is denoted by $d(S, p)$. Similarly, the distance between two sensors S_1 and S_2 is denoted by $d(S_1, S_2)$.

Definition 4. Communication neighbors - Sensors S_1 and S_2 are considered communication neighbors if the distance between them $d(S_1, S_2)$ is less than or equal to their communication range c (i.e. S_1 and S_2 are communication neighbors if $d(S_1, S_2) \leq c$).

Definition 5. Active sensors - The active sensor set consists of all sensors that are currently in the active state and is denoted by S_{active}.

Definition 6. Parent sensors - A sensor is considered a parent sensor if it is active and the base station is within its communication range. The set of all parent sensors is denoted by S_{parent}. The parent sensor set is a subset of the active sensor set ($S_{parent} \subseteq S_{active}$).

Definition 7. Connected sensors - The set of all active sensors that are connected to a parent sensor via communication neighbors is denoted by S_{conn}. The connected sensor set is a subset of the active sensor set ($S_{conn} \subseteq S_{active}$).

Definition 8. Covered points and k-covered points - The set of all points in the target region that are at least 1-covered is denoted by p_{cov}. The set of all points in the target region that are at least k-covered is denoted by p_{kcov}. The set of k-covered points is a subset of the set of covered points ($p_{kcov} \subseteq p_{cov}$).

3.2 Assumptions

The following are assumptions that our approach is based upon:

Assumption 1. All sensors in the network are homogeneous i.e. all sensors have the same sensing and communication range.

Assumption 2. Sensors communicate to the base station via neighboring sensors within their communication range. If a sensor is a parent sensor it relays information from neighboring nodes to base station.

Assumption 3. The target region is populated with mobile sensors, which are randomly deployed. Similarly, the base station is also positioned randomly within a predefined area located at the center of the target region.

Assumption 4. The deployed sensor network is dense enough to k-cover the region despite the random deployment. If the sensor network is not dense enough to k-cover the region, the algorithm will return the minimum set of active sensors that k-covers the region as much as possible.

Assumption 5. The algorithm operates under the assumption that the sensors remain stationary; however, the inclusion of mobile sensors serves the purpose of seamlessly replacing a failed or dying sensor within the network by utilizing a neighboring sensor. This enables continuous k-coverage of the area even in the event of sensor failure.

4 Genetic Algorithm

In this section, we present our genetic algorithm approach and go into detail about the design.

4.1 Coverage Model

The target region consists of an $m \times n$ grid with $m * n$ grid points. We utilize a point coverage model where a target region is considered k-covered if all grid points within that target region are k-covered. For example, a 50 m × 50 m target region would have 2500 grid points. We consider this target region k-covered if all 2500 grid points are k-covered. A point p in the target region is considered covered by a sensor S if the distance *d(S, p)* from S to p is less than or equal to the sensing radius r (i.e. p is covered if $d(S, p) \leq r$.).

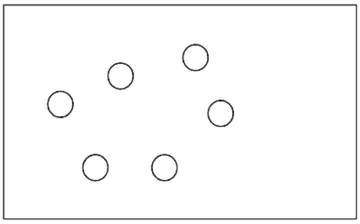
4.2 Genetic Algorithm

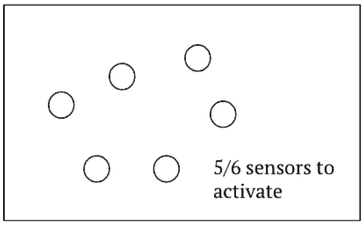
Before starting the algorithm, we deploy a given number of sensors in the target region and we deploy the base station randomly in a bounded region towards the center of the target region. A sensor determines its location using GPS technology which is communicated to the base station through a communication path. Since the algorithm is centralized, the base station takes charge of monitoring sensor locations and keeping track of potential solutions.

The algorithm randomly generates an initial population of potential solutions (see Fig. 1). Generating additional potential solutions for a larger sensor network is logical, however, scaling the population size linearly with the number of sensors in the network would result in excessive computational costs. On the other hand, scaling logarithmically with the number of sensors would result in a population size that is too small to properly explore solutions in a large network. Therefore, we compute the population size as a radical function of the total number of sensors N. We start with a base population size of 10 for small networks where $\sqrt{N}$ would not result in a sufficiently large population size. Then, we add to the population size $\frac{\sqrt{N}}{2}$ (we divide by 2 to reduce the population size further). Since the population size must be a whole number, we can apply a floor operation to the $\frac{\sqrt{N}}{2}$ term in case the result is a fraction. This calculation can be represented as the following function of N:

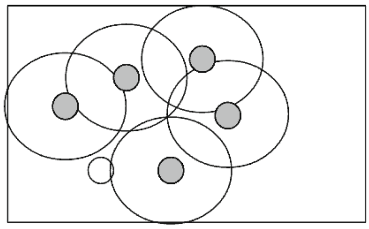
$$pop(N) = 10 + \lfloor \frac{\sqrt{N}}{2} \rfloor$$

To generate potential solutions, the algorithm picks a random number of sensors from 1 to N, denoted by $rand(N)$, and then randomly picks *rand(N)* sensors to activate from the network. After activating these sensors, it determines the following metrics to evaluate the fitness of the solution: the rate of coverage of the target region $RoC(p_{cov},\ P)$, the rate of k-coverage of the target region $RokC(p_{kcov},\ P)$, the rate of connectivity among the set of active sensors $Conn(S_{conn},\ S_{active})$, and the rate of inactivity $RoI(S_{active},\ N)$. This process is repeated *pop*(N) times to get *pop*(N) possible solutions in a single generation.


(a) Initial random sensor deployment. All sensors are inactive.



(b) The number of sensors to activate is randomly picked from 1 to N ($rand(N)$).


(c) $rand(N)$ sensors are randomly activated.

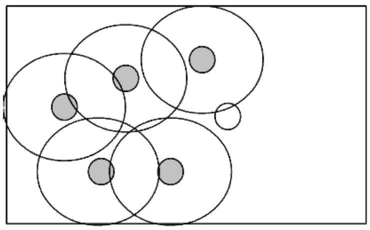

(d) Another possible formation of $rand(N)$ active sensors.

Fig. 1. Represents how a solutions is generated.

Fitness Metrics

$$RoC(p_{cov},\ P) = \frac{|p_{cov}|}{P}$$

$$RokC(p_{kcov},\ P) = \frac{|p_{kcov}|}{P}$$

$$Conn(S_{conn},\ S_{active}) = \frac{|S_{conn}|}{|S_{active}|}$$

$$RoI(S_{active},\ N) = \frac{N - |S_{active}|}{N}$$

After determining the metrics of each solution, the algorithm calculates the score of each solution using the fitness function and keeps track of the top 20% of the solutions as this provides the best results (determined through trial and error). We also keep track of the best solution for each generation and compare it to the universal best solution. If the best solution of the current generation has a higher score than the universal best solution, then the universal best solution is assigned the best solution of the generation (i.e. *bestSolUniv = max(bestSolGen, bestSolUniv)*). Storing the best universal solution is not only useful for determining the final result, but it also helps terminate the generation loop.

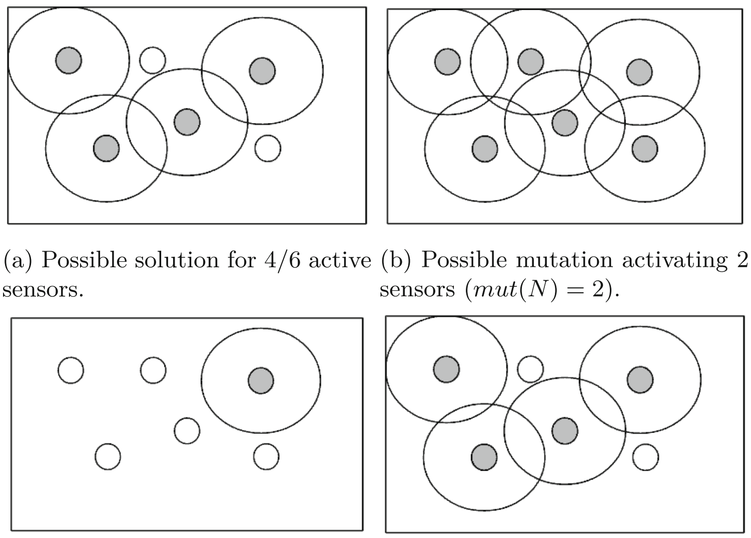
The next step is to create a new generation based on the best solutions of the current generation. We start by picking a random solution from the best solutions and apply a mutation factor to it in order to mimic random genetic mutation. This is an important step in order to explore different solution permutations. If the number of sensors N is less than or equal to 100, then the mutation factor is randomly picked from the range of integers from -3 to 3, inclusive. Otherwise, the mutation factor is randomly picked from the range of integers from $\frac{-N}{30}$ to $\frac{N}{30}$, inclusive. These ranges represent a 0–3% mutation. The function for computing the mutation factor can be written as:

$$mut(N) = \begin{cases} randInt(-3, 3), & N \leq 100 \\ randInt(\frac{-N}{30}, \frac{N}{30}), & N > 100 \end{cases}$$

If the mutation factor is a positive integer, then we must activate $mut(N)$ more sensors in the current network. If the mutation factor is a negative integer, then we must deactivate $mut(N)$ sensors in the current network. Note that the mutation factor can also be 0, in which case there will be no changes to the current solution (see Fig. 2). Once a solution is mutated, we add it to our new generation. This step of randomly selecting a solution from the best solutions and mutating it is done $pop(N)$ times to produce a new generation. After creating a new generation, we can repeat the algorithm, starting from the fitness evaluation step, to create an even more fit generation. See *Algorithm 1* for the psuedo-code of the genetic algorithm.

4.3 Terminating Conditions

As we start to create better solutions in each generation and approach the optimal solution, we need a terminating condition to stop the generational loop. Since there is no way to verify the correctness of a solution in polynomial time, we must utilize a heuristic that assumes we have determined the optimal solution based on a terminating condition. One way is to set a constant limit to the number of generational loops. This limit can be a large upper bound to the number of generations required to compute an optimal solution (such as 1000 generations) to ensure that we arrive at the most optimal solution that the algorithm can generate before exiting the loop. This approach, however, introduces redundancy as some networks will arrive at the optimal solution significantly quicker than other networks despite having the same number of sensors. So, a



(a) Possible solution for 4/6 active sensors.

(b) Possible mutation activating 2 sensors ($mut(N) = 2$).

(c) Possible mutation deactivating 3 sensors ($mut(N) = -3$)

(d) No mutation occurs $mut(N) = 0$

Fig. 2. Represents how a solutions is generated.

given network may arrive at the optimal solution long before the generation limit is reached and continue to unnecessarily calculate new solutions, wasting computational resources.

Another way to terminate the generational loop is to compute a score threshold and stop the loop when that threshold has been exceeded. The score threshold can be computed by inputting the metrics of a desired solution into the fitness function. We can then use this threshold as our terminating condition for the generational loop, ensuring that the resulting solution meets the metrics requirement of our desired solution. In other words, the resulting solution is as good as or better than our desired solution. An issue that can arise with setting a score threshold is that an infinite loop can occur if the algorithm is unable to generate a solution that exceeds the threshold. To combat this, we can set a constant limit for the number of generations in the event that a solution with a score exceeding the threshold is unattainable. If this is the case, then the algorithm will continue computing more generations despite having already achieved the most optimal solution it can produce, resulting in the same issue of excess computation.

Perhaps the best terminating condition - if we are concerned with efficiently arriving at a solution reasonably close to the most optimal solution - is to keep track of the highest score across all generations, and if that score has not been exceeded after a set number of generations, we can assume that the algorithm has arrived at the optimal solution and terminate the generational loop. The number of generations after which we want to break the loop if we have not achieved a higher score can be referred to at the "repeat threshold" (since the highest score is repeating). Having a high repeat threshold ensures that the solution is the most optimal, but also requires the computation of more generations. Finding the right number depends on the specific use case; however, in our simulations we found having the repeat threshold set to 5 provided the best results when considering accuracy and saving time.

Deciding which terminating condition to use depends on the application of the algorithm. If the goal is to efficiently find a solution close to the optimal solution that the algorithm can generate, then we can use the repeat threshold. If we simply aim to achieve a desired solution and stop computation once that solution has been attained, we can set a score threshold. If the goal is to simply achieve the most optimal possible solution that the algorithm can come up with regardless of time and computational constraints, then we can set a large upper bound on the generational loop.

Algorithm 1: Genetic Algorithm

```
initialize number of sensors as $N$
initialize population size as $popSize$
initialize empty list $solutions$
for $(i = 0 \text{ to } popSize)$ do
    create a solution object $sol$
    $numActive$ = random integer from 1 to $N$
    activate $numActive$ sensors in $sol$
    append $sol$ to $solutions$
initialize empty list $scoredSolutions$
while $repeatCounter < repeatThreshold$ do
    for $(j = 0 \text{ to } popSize)$ do
        score = fitness($solutions[j]$)
        add current solution and score to $scoredSolutions$
    sort $scoredSolutions$ in ascending order
    $bestSolutions$ = top 20% of $scoredSolutions$
    update $repeatCounter$ and highest score
    // create new generation
    initialize empty list $newGen$
    for $(j = 0 \text{ to } popSize)$ do
        $sol$ = random solution from $bestSolution$
        if $mut(N) > 0$ then
            activate $mut(N)$ more sensors in $sol$
        else
            deactivate $mut(N)$ sensors in $sol$
        append $sol$ to $newGen$
    solutions = newGen
```

4.4 Fitness Function

The fitness function calculates a score for each generated solution based on the following parameters: coverage, k-coverage, connectivity, and inactivity. The goal is to maximize each of these metrics, but to do it with different priority. For example, we must prioritize connectivity over inactivity since a connected sensor network is preferable to a disconnected sensor network with less sensors. Therefore, we must multiply each metric by a weight that represents its priority. Connectivity takes the highest precedence as a sensor network must be connected to the base station in order to relay any information it gathers. As such, we assign the highest weight to connectivity. Next, we prioritize coverage, then k-coverage, and lastly inactivity. This order of precedence ensures that we first achieve a connected network, then achieve 1-coverage, after which we focus on achieving k-coverage, and lastly, once we have a connected and k-covered sensor network, we can focusing on reducing the number of active sensors.

An issue that can arise when calculating the fitness of a solution is that a solution can come close to achieving an optimal metric, but not be exactly optimal. For example, if the optimal achievable k-coverage in a randomly deployed sensor network is 100% but requires the activation of far more sensors than achieving 99% k-coverage, then the fitness function will give a higher score to a solution that achieves 99% k-coverage with fewer sensors than to a solution that achieves 100% k-coverage with more sensors. To prioritize achieving optimal k-coverage and connectivity before minimizing the number of active sensors, we enhance the scoring of solutions that reach these optimal metrics. Specifically, we multiply the weight of a metric by 10 when it is considered optimal, thereby assigning a significantly higher score to solutions that meet these criteria compared to those that do not achieve any optimal metrics. By adopting this approach, the genetic algorithm will experience notably faster convergence to the optimal solution, given that solutions with optimal metrics will consistently attain the highest scores. If a solution achieves optimal connectivity, coverage, and k-coverage, then its score will be 0.99. From there, the fewer the number of active sensors in a solution, the closer its score will be to 1. The score of a solution should never actually be 1 as this would require the solution to have optimal metrics with 0 active sensors, which is not possible. See *Algorithm 2* for the pseudo-code of the fitness function.

4.5 Time Complexity

The algorithm computes $pop(N)$ solutions in every iteration of the generational loop. The generational loop will typically terminate when an optimal solution has been achieved; however, we set some bounding constant C as the limit of the generational loop in the event that this does not occur. This operation reduces to $O(\sqrt{N})$.

The computation of every solution requires us to activate or deactivate sensors and update coverage. To update coverage, the algorithm maintains two sets, p_cov and p_kcov. It iterates through all active sensors, employing a depth-first

Algorithm 2: Fitness Function

```
Input : coverageRate, kCoverageRate, connectivity, inactivity
Ouput: solutionScore
1 // boost score if solution reaches optimal metrics
2 if (coverageRate == optimalCoverageRate) then
3     coverageRate = coverageRate * 10
4 if (kCoverageRate == optimalKCoverageRate) then
5     kCoverageRate = kCoverageRate * 10
6 if (connectivity == optimalConnectivity) then
7     connectivity = connecitivity * 10
8 solutionScore = (0.045 * connectivity + 0.030 * coverageRate + 0.024 *
  kCoverageRate + 0.01 * inactivity)
9 return solutionScore
```

search starting at the sensor's location to update the coverage of only the points covered by the sensor. This approach has a time complexity of $O(a * r^2)$, where a is the number of active sensors, and r is the radius of the sensing disk, accounting for the quadratic scaling of the area. This process is also used when deactivating sensors to remove points from the respective sets if they are no longer 1-covered or k-covered.

Activating a sensor also requires that we initialize its communication neighbors to determine connectivity. In order to do this, we must iterate through every currently active sensor and check if it is communication neighbors with the newly activated sensor. Similarly, when we deactivate a sensor, we need to remove it from the communication neighbor set of every other active sensor, which also requires us to iterate through S_{active}. Doing this for every active sensor results in a time complexity of $O(a^2)$.

Once we have computed a potential solution, we need to determine its metrics in order to give it a score. Calculating coverage, k-coverage, and inactivity are constant time operations, but computing connectivity requires a depth-first search traversal of all currently active sensors. Since we need to traverse all active sensors at least once, the time complexity of this traversal is $O(a)$.

The time complexity of this algorithm stands at $O(\sqrt{N} * (a * r^2 + a^2))$ (the $O(a)$ step of computing connectivity reduces here). In the worst case, the number of total sensors N is equal to the number of active sensors a and taking this into account, we must write the time complexity as $O(\sqrt{(}N) * (N * r^2 + N^2))$. N^2 is greater than $N * r^2$ when the number of sensors N is greater than the sensing radius r squared which is the most likely case in a randomly deployed sensor network dense enough to k-cover a target region. Therefore, we can say that the overall time complexity of the genetic algorithm is $O(N^2\sqrt{N})$.

5 Simulation

In this section, we present our experimental results under different simulation conditions. Figures 3a, 3b and 3c show the minimum number of active sensors generated by the genetic algorithm with respect to different degrees of required coverage. Figure 3d hows the number of active sensors generated by the algorithm with respect to different sensing ranges.

5.1 Coverage Degree Vs. Number of Active Sensors

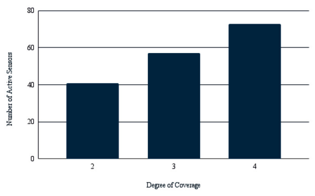
Simulation parameters for Fig. 3a: 50 m × 50 m target region (2500 points to cover), 100 deployed sensors, 10 m sensing range, and 20 m communication range ($N = 100$, $r = 10$, $c = 20$). Simulation parameters for Fig. 3b: 100 m × 100 m target region (10,000 points to cover), 300 deployed sensors, 15 m sensing range, and 30 m communication range ($N = 300$, $r = 15$, $c = 30$). Simulation parameters for Fig. 3c: 150 m × 150 m target region (22,500 points to cover), 500 deployed sensors, 20 m sensing range, and 40 m communication range ($N = 500$, $r = 20$, $c = 40$).

Note that the minimum number of active sensors for each value of k in Figs. 3a, 3b and 3c is the average of 10 simulations, each ran with a different randomly deployed sensor network. In Fig. 3a the algorithm computes an average of 40.7 active sensors for $k = 2$, 57 for for $k = 3$, and 72.9 for $k = 4$. We can see that the number of active sensors increases linearly with the required degree of coverage. The same trend can be observed in Figs. 3b and 3c, demonstrating that the algorithm performs similarly for sensor networks of different sizes and varying sensor density.

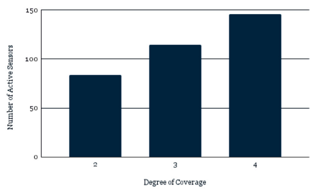
Furthermore, the average number of generations to compute a solution (which is averaged among 30 different simulations - 10 for each value of k) is roughly the same for each figure, and does not scale with any of the simulation parameters. This demonstrates that the algorithm computes solutions in roughly the same amount of generations regardless of the specific attributes of a wireless sensor network, and also explains why the number of generations is constant when computing the time complexity of the algorithm.

5.2 Sensing Range Vs. Number of Active Sensors

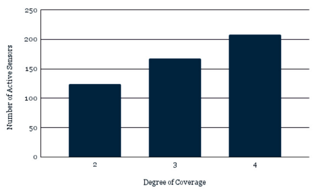
Following are the simulation parameters for Fig. 3d 100 m × 100 m target region (10,000 points to cover), 300 deployed sensors, the required degree of coverage is 3, and the communication range is twice the sensing range ($N = 300$, $k = 3$, $c = 2 * r$). Note that, as with the previous experimental results, the number of active sensors for each value of r is the average of the results of 10 simulations. We can see in Fig. 3d that as we increase the sensing range r, the number of active sensors that the algorithm computes decreases which is consistent with the expected result.



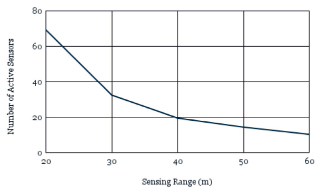
(a) Number of Generations = 26.50



(b) Number of Generations = 31.83



(c) Number of Generations = 26.73



(d)

Fig. 3. Simulation Results

6 Conclusion and Future Work

In this paper, we proposed a genetic algorithm approach to the k-coverage problem. Specifically, we focused on finding the minimum set of active sensors required to k-cover a region among a randomly deployed wireless sensor network while ensuring connectivity. We detailed how our genetic algorithm selects only the best solutions in each generation and mutates them, converging closer to the optimal solution in each iteration. Through our simulation results, we showed that the algorithm performs consistently across different types of wireless sensor networks and across varying degrees of required coverage.

Our future works consists of improving the time complexity of our algorithm by finding a faster way to verify area or point coverage. Furthermore, we plan to extend our approach by developing a scheme that allows inactive sensors to activate and move to the location of dying active sensors in order to prolong the lifetime of the network.

Acknowledgements. We would like to thank Dr. Habib M. Ammari, NSF REU Site PI, for his diligent support and review of our paper, which helped improve its overall quality. This work is funded by the US National Science Foundation under NSF grant 2338521.

References

1. Yang, S., Dai, F., Cardei, M., Wu, J.: On multiple point coverage in wireless sensor networks. In: IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, Washington, DC, 2005, pp. 8–764 (2005). https://doi.org/10.1109/MAHSS.2005.1542868

2. Elhoseny, M., Tharwat, A., Yuan, X., Hassanien, A.E.: Optimizing K-coverage of mobile WSNs. Exp. Syst. Appl. **92**, 142–153 (2018). https://doi.org/10.1016/j.eswa.2017.09.008. ISSN 0957–4174
3. Harizan, S., Kuila, P.: Coverage and connectivity aware energy efficient scheduling in target based wireless sensor networks: an improved genetic algorithm based approach. Wirel. Netw. **25**, 1995–2011 (2019). https://doi.org/10.1007/s11276-018-1792-2