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

Wireless Sensor Networks (WSNs) have gained significant attention across various fields due to their versatile applications. They have proven to be immensely valuable in domains such as health and environmental monitoring, industrial automation, and military operations. A WSN is a network composed of multiple wireless devices called sensors. These sensors are capable of measuring various physical or environmental conditions, including temperature, humidity, light, and motion. Sensors are deployed in predetermined patterns or placed randomly within the target region. By placing the sensors in different locations and gathering data from their surroundings, we can achieve a comprehensive view of the environment. This allows for the collection of large amounts of data, leading to valuable insights.

A significant challenge in WSNs revolves around achieving adequate coverage and connectivity while minimizing energy consumption. The coverage problem refers to ensuring efficient sensor coverage over the target region; however, this problem only requires the target region to be covered by 1 sensor. In the k-coverage problem, the objective is to guarantee that each point within the target region is covered by at least k sensors (where $k \geq 1$). This is especially important in hostile environments where some sensors may lose their target due to environmental factors (such as waves in the ocean). Therefore, it is critical to have multiple sensors cover that target in order to have proper fault tolerance. Additionally, ensuring that all sensors in a network are connected to the base station is vital in order for a sensor to relay the information it gathers back to the base station.

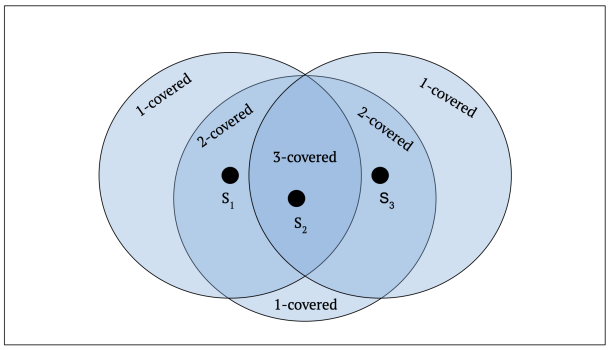


Fig. 1: Example of 1-covered, 2-covered, and 3-covered areas.

Considering the limited battery capacity of each sensor and the impracticality of recharging large amounts of sensors, it is imperative to optimize the network lifetime. Thus, it is critical to have a scheme that controls which nodes should be in the active state (doing sensing work) and which nodes should be in the inactive state. If all the sensors in a network are in the active state simultaneously, excess energy would be wasted, and the collected data would also be redundant. Therefore, by having some of the sensors in the inactive state, we decrease the energy consumption within the network. The issue, however, is identifying which sensors should be in the inactive state because connectivity and coverage still have to be maintained.

Another challenge presents itself in the form of randomly deployed sensor networks. Oftentimes deploying a WSN in a uniform formation can prove to be a challenge, and environmental factors resulting in unintentional movement of sensors can make this all the more impractical. As such, many applications of WSNs require randomized or nearly randomized sensor deployments (such as airdropping sensors from an airplane into a rain forest) and it is important to determine optimal sensor locations within these randomly deployed networks to

achieve k-coverage and connectivity. One approach to combat this issue is to implement mobility within sensors and move sensors to find the optimal locations; however, mobile sensors use significantly more energy than static sensors (since energy required for movement is greater than energy required for simple transmission) thus reducing network lifetime. Therefore, in order to develop an energy-efficient network, we must work with static sensors which may or may not be randomly deployed depending on the application.

The problem of finding the minimum number of sensors that k-cover a region among a set of randomly deployed sensors is an NP-hard problem [1]. A problem in the NP (non-deterministic polynomial time) class is a problem that cannot be solved exactly in polynomial time; however, its correctness can be verified in polynomial time. A problem in the NP-hard class is not guaranteed to have verifiable solutions in polynomial time. This means that coming up with an exact solution to this problem and verifying it in polynomial time is considered impossible. Therefore, we aim to generate an approximation that is as close to the optimal solution as possible.

We can achieve this by utilizing a genetic algorithm (GA). A GA is an approach inspired by the process of natural selection that drives evolution. We start by creating a population of randomly generated solutions and then evaluating how close those solutions are to the optimal or desired solution in a process called selection. Then, we pick the best solutions based on their scores and create a new generation with slight modifications which mimics the natural process of random genetic mutation. We iterate through this process several times, getting closer to the solution each time, until a satisfactory solution has been obtained. GAs can also include methods such as crossover which involves taking the best traits from 2 "parent" solutions and combining them to make a more fit "child" solution; however, in our approach we do not consider the crossover method, because we find that it is too computationally expensive and does not necessarily provide a more fit solution.

The algorithm is designed with the assumption that the sensors in the network are static (they cannot move); however, in a practical deployment, these sensors would be mobile. We implement it this way because sensor movement is costly in terms of energy which means moving sensors around the target region to find the optimal sensor placement would defeat the purpose of an energy-efficient algorithm. Instead, we find the optimal locations for the static sensors to k-cover the region from their initial deployment positions. Then, we can activate these sensors while deactivating all others. As the active sensors start to run out of energy, other inactive sensors near them can become activated and move towards them to take over, thereby retaining k-coverage and connectivity to prolong the network lifetime.

1.1 Problem Statement

Our research addresses the k-coverage problem within randomly deployed 2D wireless sensor networks, aiming to ensure that all points in a given target region are covered by k or more sensors (where $k \geq 1$) while selecting the minimum

amount of sensors required to achieve this. Additionally, we require that the resulting network of active sensors is connected via one or multi-hop neighbors. Therefore, the problem statement can be written as follows: given a set of randomly deployed sensors in a 2D field, find the minimum set of connected sensors that ensures that the 2D field is k-covered.

1.2 Motivations

The problem of achieving k-coverage in a 2D space remains an unresolved issue in WSNs. More specifically, the problem of finding the minimum set of active sensors to k-cover a region is an NP-hard problem. An exhaustive approach to find the optimal solution to this problem would be to find all combinations of sensors within a deployed WSN and return the one with the least number of connected sensors that k-cover the target region. The way to achieve this would be to create a decision tree, where each node represents the decision to include or not to include a sensor that has not already been selected. The time complexity of generating a decision tree like this is $O(2^N)$ where N is the number of sensors in the network. This approach is extremely slow and shows that finding an exact solution is completely impractical. Therefore, it is imperative that we find an approach that computes a close approximation to the optimal solution within a reasonable time complexity.

1.3 Major Contributions

The main contribution of this paper is a genetic algorithm that maximizes the k-coverage and connectivity of a given target region while simultaneously minimizing the number of active sensors in order to extend the network lifetime. The algorithm achieves this with a time complexity of $O(N^2\sqrt{N})$. Furthermore, we introduce a way to terminate the generational loop without knowing the optimal solution in advance.

1.4 Paper Organization

The rest of the paper is organized as follows: in Section 2, we review related work with respect to genetic algorithms in WSNs. Next, in Section 3, we define key terms and present some assumptions. Then, in section 4, we go into detail about our genetic algorithm approach. After presenting the GA, we show our simulation results in section 5. Lastly, we recap our approach and discuss future work in Section 6.

2 Related Work

Due to the widespread use of WSNs in many applications, many researches have contributed immensely to the development of the current state of WSNs.

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. Within WSNs, several other problems also belong to the NP-hard category. These include optimizing node placement, calculating the minimum energy necessary for WSN operation, determining the ideal coverage degree for k-coverage, and various others.

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. Their approach works in 2 stages: 1) they employ a GA-based cover forming method that generates all potential sensor covers, allowing for comprehensive exploration of coverage possibilities, 2) they implement a management method that efficiently switches between active sensors, effectively extending the network's lifetime. This method relocates sensors to areas where coverage can be further optimized, ensuring an adaptive and efficient utilization of resources. Yoon and Kim [3] address the maximum coverage deployment problem in WSNs by proposing a GA-based approach combined with the Monte Carlo Method. Their goal is to deploy nodes strategically in a designated area to achieve the maximum coverage. However, it is important to note that their solution lacks consideration for connectivity in its evaluation process. In [4], researchers introduce a mathematical formula to strategically deploy sensors and devise a genetic algorithm to tackle the coverage hole problem efficiently.

Furthermore, researchers have extensively investigated methods to optimize energy efficiency within the network. One way to tackle this challenge is by organizing dense sensor networks into cooperative groups called "covers". By increasing the number of covers and optimizing their lifetime, the overall network lifetime is extended. El Sherif, Fahmy, and Kamal [5] investigate the WSN lifetime problem using a two-objective optimization approaches aimed to prolong the network's overall lifetime. The first objective is to maximize the number of covers, while the second objective focuses on minimizing wasted energy in critical sensors. This is achieved through a difference factor that compares critical sensor lifetime to cover lifetime. The author's approach is called non-dominated sorting genetic algorithm-II (NSGA-II) and is used to address this optimization problem. The distinction between a non-dominated sorting algorithm (NSGA) and a genetic algorithm lies in their optimization objectives. NSGA is purposefully crafted for multi-objective optimization, dealing with scenarios where multiple conflicting objectives need simultaneous consideration. Additionally, Zahmatkesh and Yaghmaee [6] introduce a GA that optimizes energy consumption in WSNs by utilizing cluster-heads, a form of sensor grouping. The objective of their algorithm is to determine an optimal number of cluster heads and optimize the number of cluster members associated with each cluster head. The algorithm considers the distances between sensors to minimize data transmission costs within the network, thereby extending the overall network lifetime. In [7], the authors concentrate on minimizing power consumption by investigating the optimal multi-hop path between a source (cluster head) and its destination (base

station). They employ a GA to assess various factors, including the distance between the cluster head and the base station, the number of cluster heads along the path to the base stations, and the number of nodes within each cluster head.

Additionally, Hurizan and Kuila [8] 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 [8] 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. Moreover, the research in [9] on GAs served as our inspiration to further explore GAs and develop a faster and more efficient solution. The authors propose a similar GA to utilize a k-coverage model to extend WSN lifetime. The model considers factors such as the distance between the cover head and the base station, the expected energy consumption, and the amount of battery power remaining, to successfully choose what set of nodes to be active.

In [10], the paper examines the role of mutation and crossover ratios in the genetic algorithm. This study analyzes the individual impact of each ratio on the GA and explores how they can function independently of each other. While both mutation and crossover are not mandatory in the algorithm, at least one of them must be present for the approach to be considered a genetic algorithm-based approach. While all the proposed related works center around the GA, each of them tackles distinct challenges with unique objectives. A fundamental difference among these works lies in their fitness function, which houses the parameters used to evaluate solutions.

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. In Figure 2, each inner node represents a sensor and the circular disk around it represents its communication range S_c. Here, we can see that S_3 and S_4 can communicate with the base station (represented by the node with an "S") despite it not being in their communication ranges. Both of these nodes can communicate with S_2 which can communicate with S_1 which can communicate with the 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.

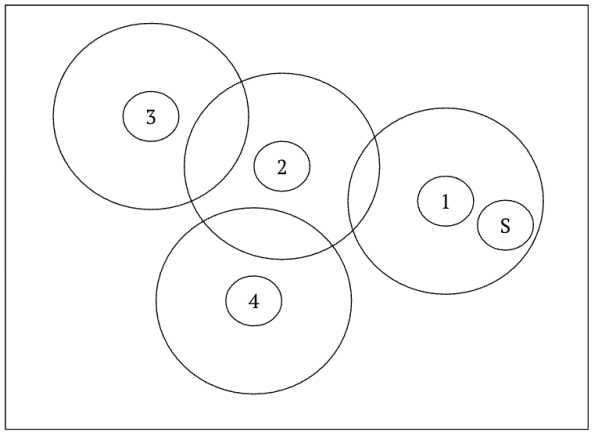


Fig. 2: S_3 and S_4 can communicate with the base station through multi-hop communication neighbors.

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 x 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 x 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 $d(S, p) \leq r$.). The following is a binary model representation of point coverage:

$$cov(S) = \begin{cases} 1, \ d(S,p), \leq r \\ 0, \ d(S,p), > r \end{cases}$$

4.2 Genetic Algorithm

Our objective is to achieve optimal k-coverage, ensuring connectivity and minimizing the number of active sensors to prolong the lifetime of a given WSN. 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 sink through a communication path. Since

the algorithm is centralized, the base station takes charge of monitoring sensor locations and keeping track of potential solutions.

Assuming all sensors are stationary at this point, the genetic algorithm can begin. The algorithm randomly generates an initial population of potential solutions (see Figure 3). 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, the rate of k-coverage of the target region, the rate of connectivity among the set of active sensors S_{active}, and the rate of inactivity. This process is repeated $pop(N)$ times to get $pop(N)$ possible solutions in a single generation.

The rate of coverage is computed as a function of the set of covered points in the target region p_{cov} and the total number of points in the target region P:

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

The rate of k-coverage is computed as a function of the set of k-covered points in the target region p_{kcov} and the total number of points in the target region P:

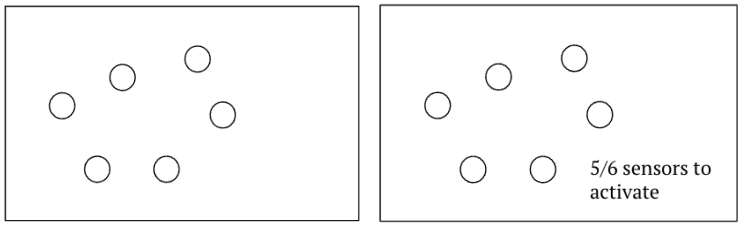
$$RokC(p_{kcov},\ P) = \frac{|p_{kcov}|}{P}$$

The rate of connectivity is computed as a function of the set of connected sensors S_{conn} and the set of active sensors S_{active}:

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

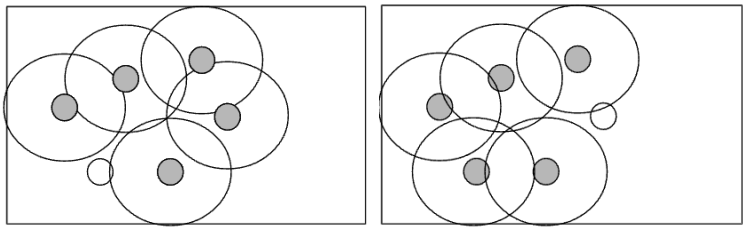
The rate of inactivity is computed as a function of the set of active sensors S_{active} and the total number of sensors N:

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



(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.

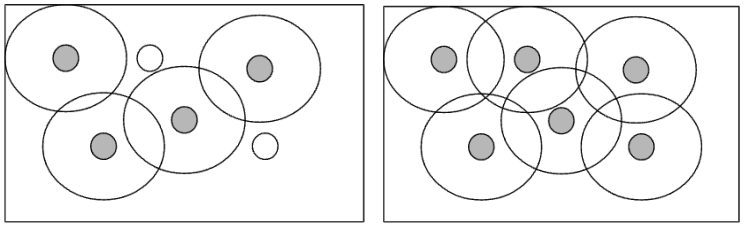
Fig. 3: Represents how a solutions is generated.

After determining the metrics of each solution, the algorithm calculates the score of each solution using the fitness function (discussed in subsection D) 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 (which is initially NULL). 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 percent mutation. The function for computing the mutation factor can be written as:

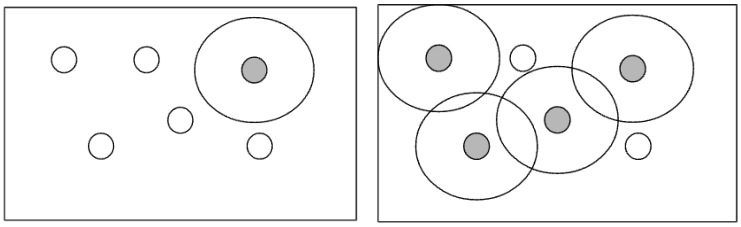
$$mut(N) = \begin{cases} randInt(-3, 3), & N \leq 100 \\ randInt(\frac{-N}{30}, \frac{N}{30}), & N > 100 \end{cases}$$

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 Figure 4). This is implemented so that a mutation is not forced upon a solution that is already optimal. 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.



(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. 4: Represents how a solutions is generated.

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. There are 3 terminating conditions that we can implement into the algorithm based on the application of the algorithm. 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 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. Furthermore, if an optimal solution has been attained, and the algorithm continues to mutate the population, a phenomenon called "negative selection" can occur, where each generation produces a worse outcome than the previous one due to negative mutations.

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 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 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 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.

Algorithm 2: Fitness Function

Input : coverageRate, kCoverageRate, connectivity, inactivity
Ouput: solutionScore

```
// boost score if solution reaches optimal metrics
if (coverageRate == optimalCoverageRate) then
    coverageRate = coverageRate * 10
if (kCoverageRate == optimalKCoverageRate) then
    kCoverageRate = kCoverageRate * 10
if (connectivity == optimalConnectivity) then
    connectivity = connecitivity * 10
solutionScore = (0.045 * connectivity + 0.030 * coverageRate + 0.024 *
  kCoverageRate + 0.01 * inactivity)
return solutionScore
```

4.5 Updating Coverage

As the algorithm activates or deactivates sensors, we need to update the coverage of the current solution in order to determine its score. To keep track of the coverage of a point, we can increment a point's coverage counter for each sensor that covers it. If a point is at least 1-covered, we add it to the set of all covered points p_{cov}. If the point is k-covered, we add it to the set of all k-covered points p_{kcov}. This allows us to determine the total number of covered and k-covered points in order to score a solution.

The brute-force approach to updating coverage is to iterate through all the points in the target region and for every point, if an active sensor covers the point, we increment the point's coverage counter. This requires us to iterate through every active sensor in the network for every point in the target region making the time complexity of this approach $O(a*P)$ where a is the number of currently active sensors. We can improve upon this time complexity by iterating through all the sensors and only updating the coverage of the points that are covered by the current sensor. This can be accomplished by using a depth-first search algorithm starting at the point at which the sensor is located. The algorithm recursively explores all points that are adjacent to the current point, stopping only when the current point is no longer covered by the sensor (base case). The time complexity of this approach is $O(a*r^2)$. The r^2 term is included due to the fact that the number of points we have to update the coverage of scales linearly with the area of the sensing disk which scales quadratically with the radius of the sensing disk (since $area = \pi * r^2$). And since we have to do this for every active sensor, we multiply r^2 by a. The algorithm works the same way for deactivating sensors, but instead of adding a point to p_{cov} or p_{kcov}, we remove it from the respective set if it is no longer 1-covered or k-covered.

4.6 Checking Connectivity

An important functionality of the algorithm is to be able to efficiently check how many sensors in a set of active sensors S_{active} are connected to the base station through communication neighbors. Two sensors are considered connected if they are communication neighbors. If S_1 is communication neighbors with S_2 and S_3, and S_2 and S_3 are not communication neighbors, then S_2 and S_3 are considered connected through one-hop neighbors. The one-hop neighbor in this case is S_1 since any transmission between S_2 and S_3 must pass through or "hop over" S_1 before reaching its destination. As such, all sensors reachable to a parent sensor through one or multi-hop communication neighbors are considered connected to the base station. This allows us to determine the rate of connectivity to evaluate the score of a particular solution. Checking how many sensors are connected to the base station can be done by utilizing a depth-first search algorithm that recursively explores all communication neighbors of the current sensor, starting with any parent sensor. This algorithm will also establish communication paths to the base station from any sensor that is connected. The time complexity for this depth-first search algorithm is $O(a)$ since it visits every active sensor once.

While the depth-first search algorithm establishes communication paths to the base station from every connected sensor, these paths are not guaranteed to be the shortest paths. We can use Dijkstra's shortest path algorithm [11] to compute the shortest path from each connected sensor to a parent sensor in $O(a^2)$. While computing the shortest communication path to simply check how many sensors in S_{active} are connected is not efficient, it is important to establish the shortest communication paths once the algorithm has arrived at an optimal solution as not doing so would result in wasted energy in the form of transmission. Therefore, we utilize a simple depth-first search to check for connectivity in $O(a)$, but compute the shortest path in $O(a^2)$ once an optimal sensor network has been established.

4.7 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. Thus far, the time complexity of computing solutions in every generation is $O(pop(N) * C)$ which can be simplified to $O(\sqrt{N})$.

The computation of every solution requires us to activate or deactivate sensors, the time complexity of which is $O(a * r^2)$. Note that after the initial computation of solutions, we do not need to recompute the coverage of every single active sensor, instead we only need to update the coverage for the sensors that we activate or deactivate in the mutation step. In other words, after the initial computation of random solutions (which is $O(a * r^2)$), the time complexity of creating a mutated solution is $O(mut(N) * r^2)$.

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 $O(a^2)$. After the initial computation of random solutions, this time complexity reduces to $O(mut(N) * a)$.

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. Previously, we established the time complexity of this traversal to be $O(a)$.

Despite the optimizations in computing communication neighbors and updating coverage, in order to compute the time complexity, we must consider the worst case of the initial computation of random solutions. Therefore, 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). While the goal of the genetic algorithm

is to reduce the number of active sensors, in the worst case, the number of total sensors N is equal to the number of active sensors a. This can occur when every single sensor in the network needs to be activated in order to achieve k-coverage. Taking this into consideration, 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 (i.e. $N^2 > (N * r^2)$ if $N > r^2$). This is the most likely case in a randomly deployed sensor network dense enough to k-cover a target region and thus will be the case in our simulations. 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 5, 6, and 7 show the minimum number of active sensors generated by the genetic algorithm with respect to different degrees of required coverage. Figure 8 shows 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 Figure 5: 50 m x 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 Figure 6: 100 m x 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 Figure 7: 150 m x 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 Figures 5, 6, and 7 is the average of 10 simulations, each ran with a different randomly deployed sensor network. In Figure 5, 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 Figures 6 and 7, 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 Figure 8: 100 m x 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 Figure 8 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.

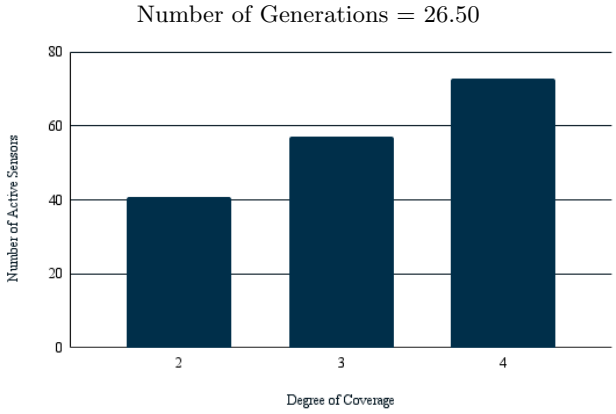


Fig. 5

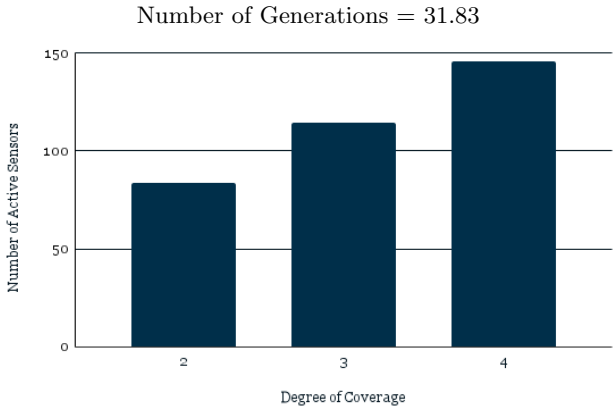


Fig. 6

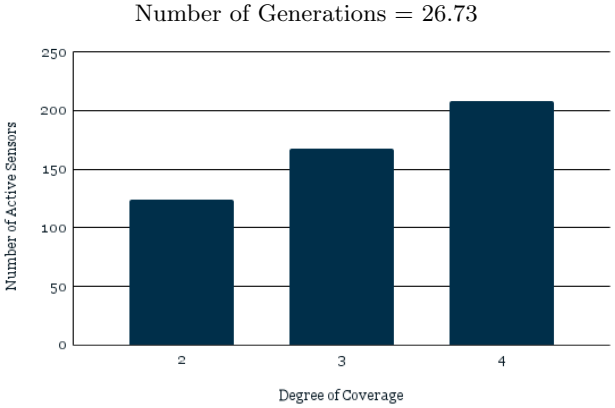


Fig. 7

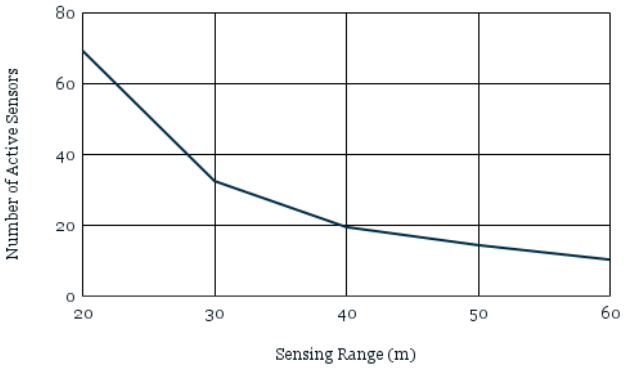


Fig. 8

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. Shuhui Yang, Fei Dai, M. Cardei and Jie Wu, "On multiple point coverage in wireless sensor networks," IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005., Washington, DC, 2005, pp. 8 pp.-764, doi: 10.1109/MAHSS.2005.1542868.

2. Mohamed Elhoseny, Alaa Tharwat, Xiaohui Yuan, Aboul Ella Hassanien, Optimizing K-coverage of mobile WSNs, Expert Systems with Applications, Volume 92, 2018, Pages 142-153, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2017.09.008.

3. Yoon Y, Kim Y-H. An efficient genetic algorithm for maximum coverage deployment in wireless sensor networks. IEEE Trans Cybern 2013;43(5):1473–83.

4. Mnasri, Sami and Thaljaoui, Adel and Nasri, Nejah and Val, Thierry A genetic algorithm-based approach to optimize the coverage and the localization in the wireless audiosensors networks. (2015) In: IEEE International Symposium on Networks, Computers and Communications (ISNCC 2015), 13 May 2015 - 15 May 2015 (Hammamet, Tunisia).

5. El-Sherif, M., Fahmy, Y. and Kamal, H. (2018), Lifetime maximisation of disjoint wireless sensor networks using multiobjective genetic algorithm. IET Wirel. Sens. Syst., 8: 200-207. https://doi.org/10.1049/iet-wss.2017.0069

6. Zahmatkesh, A., Yaghmaee, M.H.: A genetic algorithm-based approach for energy-efficient clustering of wireless sensor networks. Int. J. Inf. Electron. Eng. 2(2), 165 (2012)

7. Al-Shalabi, M., Anbar, M., Wan, T.-C., Alqattan, Z.: Energy efficient multi-hop path in wireless sensor networks using an enhanced genetic algorithm. Inf. Sci. (Ny) 500, 259–273 (2019)

8. Harizan, S., Kuila, P. Coverage and connectivity aware energy efficient scheduling in target based wireless sensor networks: an improved genetic algorithm based approach. Wireless Netw 25, 1995–2011 (2019). https://doi.org/10.1007/s11276-018-1792-2

9. M. Elhoseny, A. Tharwat, A. Farouk and A. E. Hassanien, "K-Coverage Model Based on Genetic Algorithm to Extend WSN Lifetime," in IEEE Sensors Letters, vol. 1, no. 4, pp. 1-4, Aug. 2017, Art no. 7500404, doi: 10.1109/LSENS.2017.2724846.

10. Hassanat A, Almohammadi K, Alkafaween E, Abunawas E, Hammouri A, Prasath VBS. Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach. Information. 2019; 10(12):390. https://doi.org/10.3390/info10120390

11. Dijkstra, E.W. A note on two problems in connexion with graphs. Numer. Math. 1, 269–271 (1959). https://doi.org/10.1007/BF01386390