Thesis submitted to the

UNIVERSITY OF MOHAMED BOUDIAF - M'SILA, ALGERIA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

in Partial Fulfillment of the Requirements for the Degree of:

Master's in Computer Science

Specialization: Business Intelligence and Optimization

By

Bougoutaia, Chaima Reyane

Khelifi, Takoua

Entitled

Coverage Optimization in Wireless Sensor Networks Using Memetic Algorithm

Under the supervision of

Raouf Ouanis Lakehal Ayat

Jury Members

University of M'sila	President
University of M'sila	Reporter
University of M'sila	Examiner

June, 2025

الملخص

تتناول هذه المذكرة مشكلة تحسين التغطية في شبكات المستشعرات اللاسلكية، وهي من بين أبرز التحديات التي تؤثر على كفاءة الشبكات وفعاليتها في مراقبة البيئة. تم اقتراح خوارزمية ميمية هجينة تجمع بين البحث الشامل والبحث المحلي لتحسين توزيع المستشعرات وتقليل التداخل مع الحفاظ على التغطية القصوى بأقل عدد ممكن من العقد. تم نمذجة المشكلة رياضياً وتنفيذ محاكاة لاختبار أداء الخوارزمية في سيناريوهات مختلفة. أظهرت النتائج أن النهج المقترح يحقق توازناً جيداً بين جودة التغطية واستهلاك الموارد، مقارنة بالخوارزميات التقليدية.

الكلمات المفتاحية: الخوارزميات الميمتيكية، شبكات المستشعرات اللاسلكية، تحسين التغطية.

Abstract

This dissertation addresses the coverage optimization problem in Wireless Sensor Networks (WSNs), a critical factor affecting network efficiency and sustainability. A hybrid memetic algorithm is proposed, combining global and local search strategies to optimize sensor deployment and reduce overlap while maintaining maximum coverage with minimal sensor usage. A precise mathematical model is formulated, and simulation experiments are conducted to evaluate the performance of the proposed approach. Results show significant improvements in coverage quality and energy efficiency compared to conventional algorithms.

Keywords: Memetic Algorithms, Wireless Sensor Networks, Coverage Optimization.

Dedication

“For those who stood quietly behind us,

believing, encouraging, and giving—

This work is a reflection of your presence in our journey.”

Khelifi Takoua, Bougoutaia Chaima Rayene

Acknowledgements

We are deeply indebted to our thesis supervisor **Raouf Ouanis Lakehal Ayat** whose steadfast support and inspiration made this project a success. In a very special way, we thank them for their continuous guidance and encouragement throughout this challenging study.

Special thanks go to our friends and families who endured the hectic moments and supported us during the course of the research.

We also thank our institution for giving us the opportunity to collaborate as a team, which greatly improved our teamwork and communication skills. Our sincere appreciation goes to all group members for their solidarity and commitment.

Contents

List of Tables

List of Figures

Introduction

In recent decades, the world has witnessed rapid and unprecedented technological advancement, leading to the emergence of intelligent systems capable of processing large volumes of data and making precise and informed decisions in various fields such as healthcare, industry, agriculture, and environmental monitoring. This significant technological progress has coincided with an increasing demand for effective and efficient tools to observe and analyze field data in real-time, which paved the way for the development and widespread use of Wireless Sensor Networks (WSNs).

WSNs are among the modern technological solutions designed to monitor different environments by deploying a set of sensors across a given area. These sensors collect information and transmit it to a central processing station. Thanks to their flexibility, scalability, and low cost, WSNs have been widely adopted in critical and diverse applications such as temperature monitoring, fire detection, object tracking, and many others.

Despite the notable benefits offered by WSNs, their effectiveness remains closely tied to the efficiency of coverage, i.e., the ability of the deployed nodes to monitor the target area without leaving blind spots, excessively consuming energy, or using an unnecessary number of sensors. Random or poorly planned node deployment can lead to what is known as sensing holes or excessive overlap, which negatively impacts network lifetime and accuracy.

To improve this crucial aspect, many studies have employed intelligent optimization techniques aimed at achieving optimal node placement to ensure full coverage with the minimum number of sensors while minimizing energy consumption. Several metaheuristic algorithms have been explored in this context, including Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and others.

However, despite the promising results achieved, some of these algorithms suffer from limitations such as slow convergence or inefficient exploitation of good solutions, especially in large and complex search spaces. In response to these recurring challenges, Memetic Algorithms (MAs) have emerged as a promising alternative due to their hybrid nature, combining global and local search strategies. This integration significantly enhances the algorithm's ability to find high-quality solutions to complex optimization problems, such as the coverage problem in WSNs.

Based on these considerations, this dissertation proposes a hybrid multi-objective memetic algorithm designed to improve the coverage of Wireless Sensor Networks by optimizing node distribution, reducing coverage redundancy, and improving energy efficiency while minimizing the number of deployed sensors.

This work is structured as follows:

- **Chapter 1:** provides a comprehensive overview of Wireless Sensor Networks, presenting their key components, core characteristics, deployment environments, and the critical challenges they face. A special focus is placed on the coverage problem, analyzing its impact on overall network efficiency, its relationship with energy consumption, and common issues such as sensing holes and redundancy.

- **Chapter 2:** introduces essential optimization concepts and explores different categories of optimization problems, including their complexity and mathematical structure. It highlights the role of metaheuristic algorithms in solving such problems, with a particular focus on memetic algorithms. The chapter also justifies the choice of MAs for addressing WSN coverage issues, explaining their advantages over other methods.

- **Chapter 3:** details the design of the proposed memetic algorithm. It covers the modeling of the coverage optimization problem as a multi-objective optimization task, defines the input parameters and constraints, and elaborates on the structure of the algorithm, including global and local search mechanisms. The chapter also presents the strategies used for balancing exploration and exploitation, and how the algorithm is adapted to the specific characteristics of WSNs.

- **Chapter 4:** presents the simulation setup and experimental results. It describes the software and hardware environments, simulation parameters, and performance metrics. The results of the proposed algorithm are analyzed and compared against other existing approaches to evaluate improvements in coverage, sensor usage, and energy efficiency. Observations and insights from the experiments are also discussed, highlighting the strengths and limitations of the approach.

Chapter 1: Wireless Sensor Networks and the Coverage Problem

1.1 Introduction

With the advancement of communication technologies, wireless networks have become a fundamental component of modern infrastructures, offering flexibility and ease of connectivity without the need for complex physical setups. These networks encompass various types used across multiple fields, most notably Wireless Sensor Networks (WSNs), which have proven effective in monitoring physical environments and vital data. This chapter discusses the general principles of these networks, their architecture, components, and key challenges, with a particular focus on coverage as a central element in improving performance.

1.2 Wireless Networks

1.2.1 Definition of Wireless Networks

A wireless network is defined as a computer network that relies on wireless communication using Radio Frequency (RF) waves to transfer data between nodes without physical connections. These networks are used to provide connectivity between devices in various environments and are characterized by their flexibility and ease of deployment compared to wired networks[1].

1.2.2 Classification of Wireless Networks

As communication technologies continue to advance, wireless networks have become increasingly diverse in their architecture, applications, and coverage areas. This growing complexity calls for a systematic classification based on well-defined criteria. The primary types of wireless networks can be categorized as follows:[1]

- Wireless Personal Area Networks (WPAN), such as Bluetooth and ZigBee.
- Wireless Local Area Networks (WLAN), such as Wi-Fi.
- Wireless Wide Area Networks (WWAN), such as 4G and 5G.
- Wireless Sensor Networks (WSNs).

1.3 Wireless Sensor Networks

1.3.1 Definition of a Wireless Sensor

A device that performs sensing functions is called a 'sensor'. The human body, for example, can perceive visual information, sounds, and smells from the environment using the eyes, ears, and nose without physically touching the observed object; this is an example of a remote sensor. A sensor acts as a transducer, converting energy into electrical energy or another form of energy[2].

There are various types of sensors, each specializing in a specific field, such as temperature sensors, humidity sensors, gas sensors, and smoke sensors[3]

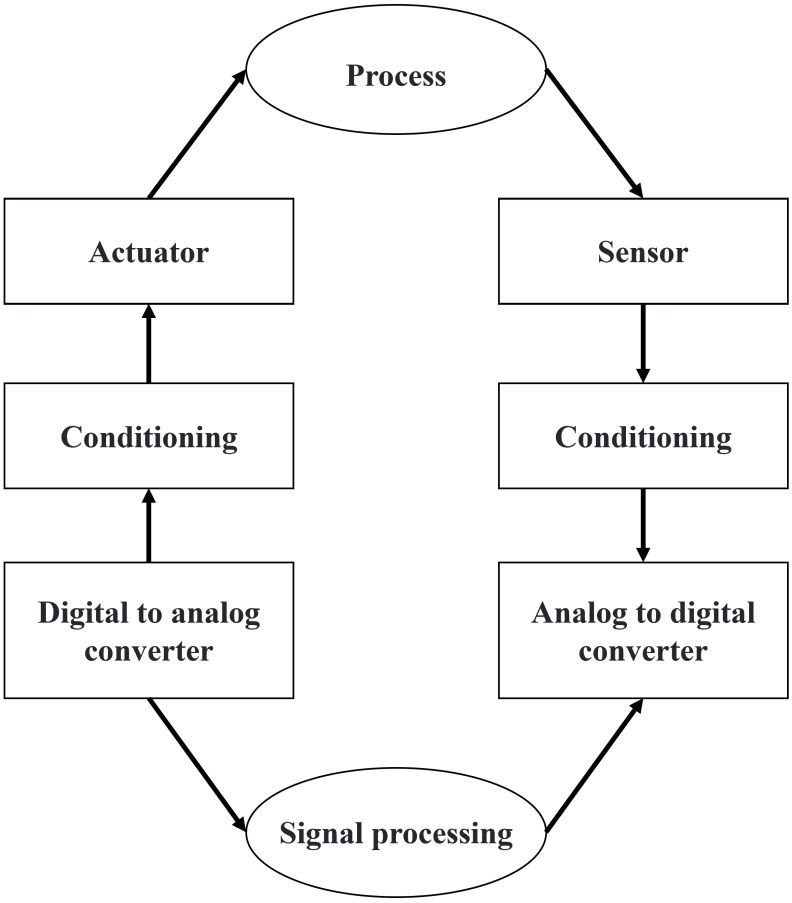

Figure 1.1: Diagram of sensor data collection and activation process[2].

1.3.2 Definition of Wireless Sensor Networks (WSNs)

A Wireless Sensor Network (WSN) is an infrastructure-less wireless network composed of a large number of wirelessly connected sensor devices deployed in a dedicated manner to monitor a system or physical/environmental conditions. The sensor nodes in a WSN are equipped

with embedded processors that manage and monitor the environment within a defined area. These nodes communicate with a base station that acts as a processing unit in the WSN architecture. The base station in a WSN system is connected to the Internet for data sharing. Wireless Sensor Networks can be used for data processing, analysis, storage, and retrieval[4].

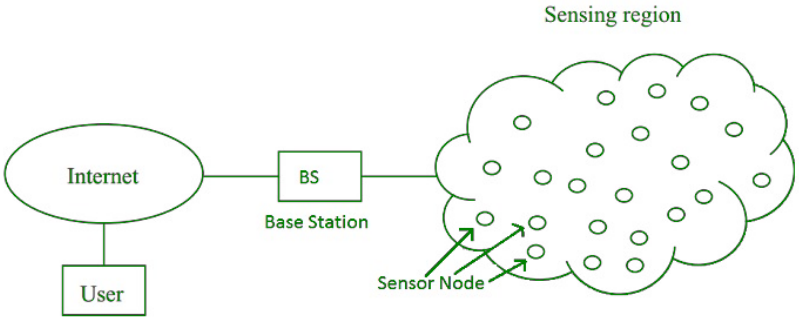

Figure 1.2: Basic structure of a Wireless Sensor Network[4].

The choice of network type depends on the environment, allowing deployment underwater, underground, on land, and in other environments. Different types of WSNs include:[5]

- Terrestrial Wireless Sensor Networks.
- Underground Wireless Sensor Networks.
- Underwater Wireless Sensor Networks.
- Multimedia Wireless Sensor Networks.
- Mobile Wireless Sensor Networks.

1.4 Components of Wireless Sensor Networks

The principle components of WSNs are:[6]

- **Sensor Nodes:** These are devices that collect data using sensors that measure temperature, humidity, pressure, gas, or motion. Each node also contains a data processor, a wireless communication unit, and a transmitter.
- **Gateways and Routers:** These are used to connect the wireless sensor network to other networks such as the internet or local area networks. They aggregate data from the sensor nodes and forward it to its destination. They also help manage and route data flow within the network for efficient communication.
- **Protocols:** These include wireless protocols such as Zigbee, Wi-Fi, Bluetooth, and LPWAN, which facilitate communication between sensor nodes.

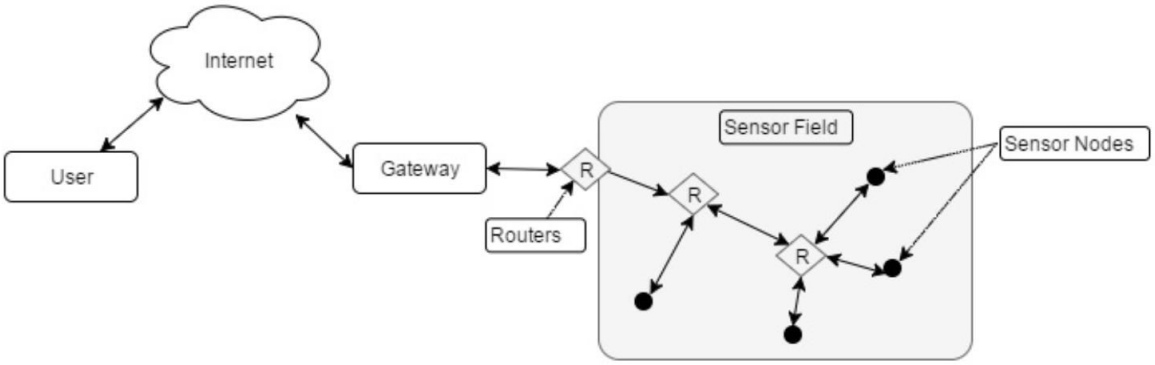

Figure 1.3: Wireless Sensor Network components[6].

1.5 Performance Metrics in Wireless Sensor Networks

To ensure the efficiency of WSNs, a set of technical metrics is adopted to evaluate their performance and achieve optimal results in various operating environments. The most prominent of these metrics include:[5]

- **Coverage:** The area covered by the network.
- **Energy Consumption:** The importance of preserving the nodes' energy.
- **Latency:** The speed at which data is transmitted.
- **Reliability:** The accuracy and stability of communications.
- **Security:** The network's ability to resist attacks.

1.6 Enhancing the Performance of Wireless Sensor Networks

WSNs face numerous technical challenges that hinder their optimal performance, which necessitates the use of advanced mechanisms and algorithms to enhance their efficiency.

Among the most prominent challenges facing Wireless Sensor Networks is energy consumption. Sensor nodes are usually powered by limited-capacity batteries, and excessive energy usage can drastically reduce the network's operational lifetime. Therefore, it is crucial to design energy-efficient algorithms that not only extend network longevity but also optimize the spatial placement of sensor nodes. In fact, minimizing energy consumption has been identified as a key objective within the Node Location Problem, as strategic node placement directly contributes to reducing power demands across the network[7].

On the other hand, routing is considered one of the critical issues, as it involves determining the best paths for data transmission in a way that minimizes both energy consumption and

latency. Likewise, security represents a fundamental challenge due to the nature of wireless networks, which are vulnerable to threats such as eavesdropping or data manipulation.

As for coverage, it is regarded as one of the most important indicators of network quality, as it directly relates to the network's ability to monitor the target area without data loss or sensing gaps[8].

In light of these various challenges, this work focuses on enhancing coverage as a key factor in improving network efficiency and ensuring continuous, accurate, and effective monitoring of the target environment. This aspect serves as a foundational element upon which several other metrics, such as communication quality, routing efficiency, and even energy consumption are built. This will be further clarified in the following section.

1.7 Enhancing Coverage in Wireless Sensor Networks

1.7.1 Impact of Coverage on Network Efficiency

Coverage is one of the fundamental determinants of the efficiency of WSNs, as it is directly related to the network's ability to monitor and collect data from target areas. Inadequate or uneven coverage can result in coverage holes, which are regions not reached by sensor signals, leading to the loss of critical information that may affect the quality of decisions derived from the collected data. Conversely, improved coverage ensures continuous and effective environmental monitoring, enhances system reliability, and reduces the need for data redundancy or post-processing of missing information[9].

1.7.2 The Relationship Between Coverage and Energy Consumption

Coverage and energy consumption share a sensitive trade-off relationship. Improving coverage influences the distribution pattern and activity level of sensor nodes, which in turn affects energy consumption. For instance, increasing the number of nodes to expand coverage may enhance monitoring performance, but it also accelerates energy depletion if not managed efficiently. On the other hand, intelligent node distribution combined with optimized energy activation and transition mechanisms allows for maintaining sufficient coverage while minimizing energy waste, forming a crucial focus in the design of high-efficiency WSNs.

1.7.3 Challenges in Coverage Enhancement

- **Coverage Holes:** These are areas within the monitored field that are not covered by sensors, leading to missing critical data.
- **Geographic Routing:** It relies on the nodes' locations for data transmission, and poor placement can weaken coverage.

- **Coverage Topology:** Refers to how sensor nodes are arranged in the network, directly impacting coverage quality.
- **Energy and Node Count Limitations:** Limited energy and number of nodes hinder achieving full and sustainable coverage[5].

1.7.4 The Importance of Improving Coverage Using Intelligent Algorithms

The adoption of intelligent algorithms in WSNs has led to a significant shift in how coverage is managed and optimized. These algorithms, such as evolutionary algorithms, swarm intelligence, and stochastic optimization techniques, allow for the automatic redistribution of nodes or control their activation based on environmental variables, without the need for direct human intervention. They also help reduce interference between nodes, enhance energy consumption efficiency, and ensure comprehensive coverage of the monitored area. Utilizing such algorithms not only improves the network's effectiveness but also extends its operational lifetime, making them a strategic choice for large-scale and complex monitoring applications.

1.8 Sensor Deployment Methods in Wireless Sensor Networks

The method of deploying sensor nodes is a fundamental factor that directly affects the performance of WSNs, particularly in terms of coverage, energy consumption, and communication efficiency. There are various deployment strategies, and choosing the appropriate one depends on the target environment, the purpose of the network, and the level of control available during deployment. The most common deployment methods include:

- **Random Deployment:** This method is used when it is difficult to control the placement of sensor nodes, such as in rugged terrains or war zones, where nodes are deployed via aerial drops or artillery. Node positions are often modeled using probabilistic approaches, like the 2D Poisson point process. While suitable for rapid deployment, this method may lead to coverage holes or overlapping areas and typically requires additional algorithms to adjust node distribution and improve overall network performance[10].
- **Grid-Based Deployment:** This method involves deploying sensor nodes in a regular grid pattern (for example, squares or hexagons), where each node covers a specific part of the target area. This organized distribution helps achieve uniform coverage, minimizes overlap or coverage holes, and facilitates node density calculation and energy

estimation. However, it requires careful planning and may not be suitable for irregular or unknown environments[11].

- **Clustering Techniques:** Clustering divides the network into groups or clusters, each led by a node called the Cluster Head. This node is responsible for receiving data from its members, partially processing it, and forwarding it to the base station. This method aims to reduce energy consumption and transmission delay, especially in large-scale networks, while also improving routing organization and network reliability. Popular clustering protocols include LEACH and HEED[12].

1.9 Coverage Optimization Techniques in Wireless Sensor Networks

Given the importance of coverage in ensuring the efficiency of WSNs, various techniques have been developed to enhance this critical aspect. These techniques differ in their methodologies and approaches to the coverage problem and can be categorized as follows:

- **Node Relocation:** This technique relies on dynamically moving sensor nodes after initial deployment to improve spatial distribution and reduce sensing holes. This can be done using either local or centralized algorithms that take into account the nodes' locations and remaining energy levels.

- **Transmission Power Control:** By dynamically adjusting the transmission power of nodes, it is possible to cover marginal areas without adding new nodes, thereby improving coverage and reducing interference and energy consumption.

- **Coverage Optimization Using Intelligent Algorithms:** Nature-inspired algorithms such as evolutionary algorithms (e.g., Genetic Algorithm, Harmony Search) and swarm intelligence algorithms (for example, ACO and PSO) are used to optimize node distribution in WSNs. These algorithms aim to maximize coverage and minimize interference through smart search or self-organized cooperation among nodes.

- **Hybrid Techniques:** This approach combines multiple techniques simultaneously, such as merging relocation with dynamic scheduling or integrating evolutionary and swarm algorithms, to achieve more accurate and efficient results in complex and dynamic environments.

1.10 Conclusion

It is evident from this chapter that WSNs play a pivotal role in intelligent monitoring systems. However, their effectiveness is influenced by various challenges, most notably the coverage problem. The chapter reviewed techniques used to enhance this aspect, given its direct impact on network efficiency and energy consumption. This chapter lays the groundwork for further exploration of ways to improve the performance of these networks in more specialized contexts.

Chapter 2: Optimization Techniques and the Memetic Algorithm

2.1 Introduction

Based on the first chapter, where we discussed WSN and their challenges, particularly regarding coverage and performance efficiency, this chapter addresses optimization methods. We present the main concepts in this field, classify its types and problems, and focus on the most commonly used algorithms, especially memetic algorithms due to their ability to handle complexities such as coverage improvement and energy reduction. This introduction aims to provide a conceptual framework for effectively addressing WSN problems.

2.2 Definition of Optimization

Optimization is the process of finding the best strategy to achieve a specific objective with maximum efficiency and effectiveness, while minimizing costs, effort, or consumed resources. It relies on analyzing the current situation, identifying problems or obstacles, and then finding and applying solutions that yield the best possible results according to predefined criteria.

2.3 Components of Optimization Problems

- **Objective:** What are we trying to optimize?
- **Resources:** The available factors that can be utilized to achieve optimization.
- **Constraints:** The limits or conditions that must be respected during the optimization process.
- **Metrics:** The performance evaluation standards used to measure the success of the optimization.
- **Strategies:** The methods and techniques that can be employed to achieve optimization.

2.4 Types of Optimization Problems

This classification includes the fundamental well-known classes:[13]

- **P (polynomial time):** class of problems that are solvable in polynomial time with a deterministic Turing machine.

- **NP (non-deterministic polynomial time):** class of problems that can only be verified in polynomial time with a deterministic Turing machine.
- **NP-complete:** pb that can be solved in P (but in a large polynomial time) and can be verified in NP and all pbs of NP class can be reduced to it.
- **NP-hard:** pb that have not a solution in P but can be verified in NP and all pbs of NP class can be reduced to it.

There are two hypotheses depending on whether P = NP or P ≠ NP. The image illustrates these two scenarios visually. In the first hypothesis (P ≠ NP), the class of problems that can be solved efficiently (P) is strictly smaller than the class of problems whose solutions can be verified efficiently (NP). This implies that there are problems in NP, such as those in NP-Complete, which cannot be solved in polynomial time, even though their solutions can be verified in polynomial time.

In the second hypothesis (P = NP), all problems whose solutions can be verified efficiently can also be solved efficiently. As a result, the classes P, NP, and NP-Complete merge into a single class. In both cases, the NP-Hard class remains at the top, as it includes problems believed to be even more complex, some of which may not even be verifiable in polynomial time.

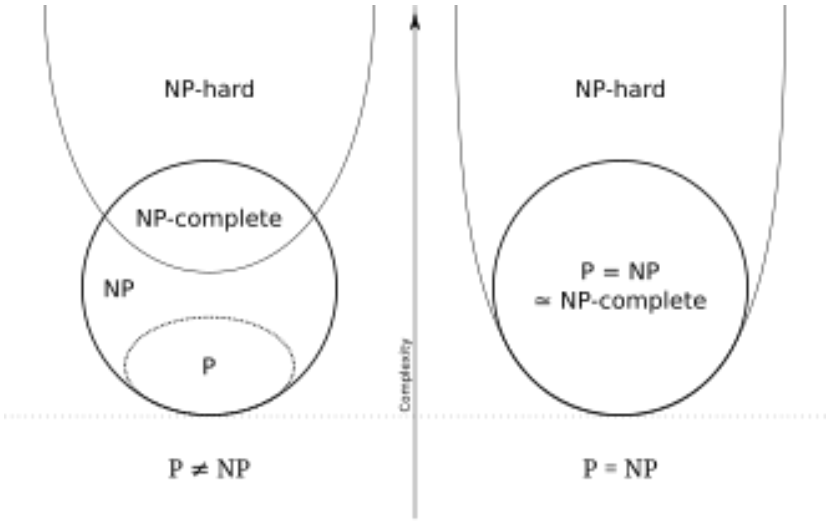

Figure 2.1: Euler diagram for P, NP, NP-complete, and NP-hard set of problems[14].

2.5 Optimization algorithm

Optimization algorithms are a class of algorithms designed to find the best or near-optimal solution to a given problem by adjusting inputs in a way that maximizes the objective function's

benefit. These algorithms are widely used in various applications, ranging from business process optimization and decision-making to artificial intelligence and data analysis, enabling the discovery of optimal solutions within a vast set of possible options[15].

2.6 Time Complexity Analysis

2.6.1 Definition

Time complexity refers to the number of operations an algorithm performs to accomplish its task, assuming each operation takes the same amount of time. An algorithm that completes a task with the fewest operations is considered the most efficient, regardless of the machine it runs on. When computing the time complexity T(n) of an algorithm, we rarely obtain an exact result, only an estimate. That is why, in computer science, we use asymptotic notations, which provide an approximation of the number of elementary operations.

2.6.2 Asymptotic Notations

There are three main asymptotic notations:

- **Big omega Ω notation:** for the best case, it describes the minimum time an algorithm can take to complete, assuming the most favourable input. It provides a lower bound on the time complexity.
- **Big theta Θ notation:** for the average case, it represents the expected time complexity of an algorithm over all possible inputs. It gives a tight bound, meaning it describes both the lower and upper limits for the average performance.
- **Big O notation:** for the worst case, it indicates the maximum time an algorithm may take for any input of size n. It provides an upper bound on the time complexity and is the most commonly used in algorithm analysis.

2.6.3 Common Time Complexity Classes

We can classify the types of time complexity (Big O Notation) based on how fast the number of operations grows with respect to the input size n, from the most efficient to the least, as follows:

- **O(1) - Constant Time:** The number of operations remains the same regardless of the input size. Example: Accessing a specific element in an array.
- **O(log n) - Logarithmic Time:** The number of operations increases very slowly as the input grows. Common in search algorithms like binary search.

- **O(n) - Linear Time:** The number of operations increases proportionally with the input size. Example: Iterating over the elements of an array.
- **O(n log n) - Log-Linear Time:** More efficient than quadratic time, commonly seen in efficient sorting algorithms like Merge Sort and Quick Sort.
- **O(n^2) - Quadratic Time:** The number of operations grows significantly as the input increases, such as in algorithms with double comparisons (nested loops in sorting).
- **O(n^3) or higher - Polynomial Time:** The number of operations increases at a polynomial rate relative to the input size. Often appears in algorithms that require multiple pairwise comparisons, like the Bubble Sort algorithm.
- **O(2^n) - Exponential Time:** The number of operations doubles with each small increase in n. Example: Solving certain dynamic programming problems or generating power sets.
- **O(n!) - Factorial Time:** The least efficient, with performance deteriorating rapidly. Appears in algorithms that explore all possible permutations, such as some solutions to the Traveling Salesman Problem.

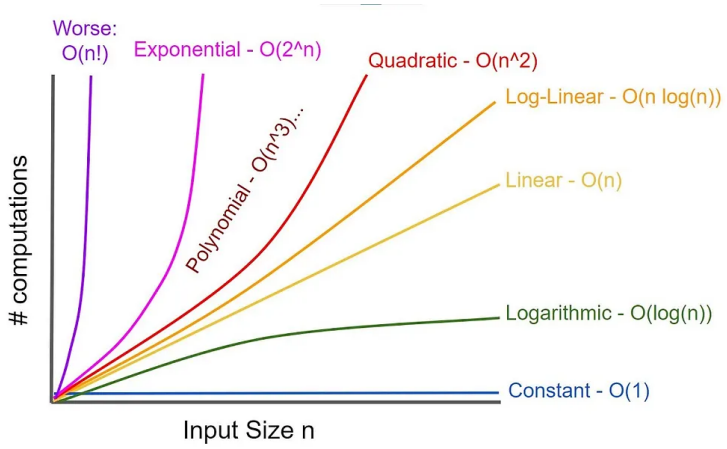

Figure 2.2: Time Complexities Graph[16].

2.7 Classification of Algorithms According to Mathematical Structure

The classification of algorithms is one of the key aspects that contributes to understanding their working mechanisms and internal organization. Algorithms can be classified based on the mathematical structure or methodology they rely on to solve problems. This classification

not only helps organize algorithms according to their mathematical logic but also facilitates the selection of the most appropriate approach depending on the nature and complexity of the problem. The following diagram provides an overview of the most commonly used types of algorithms according to this classification.

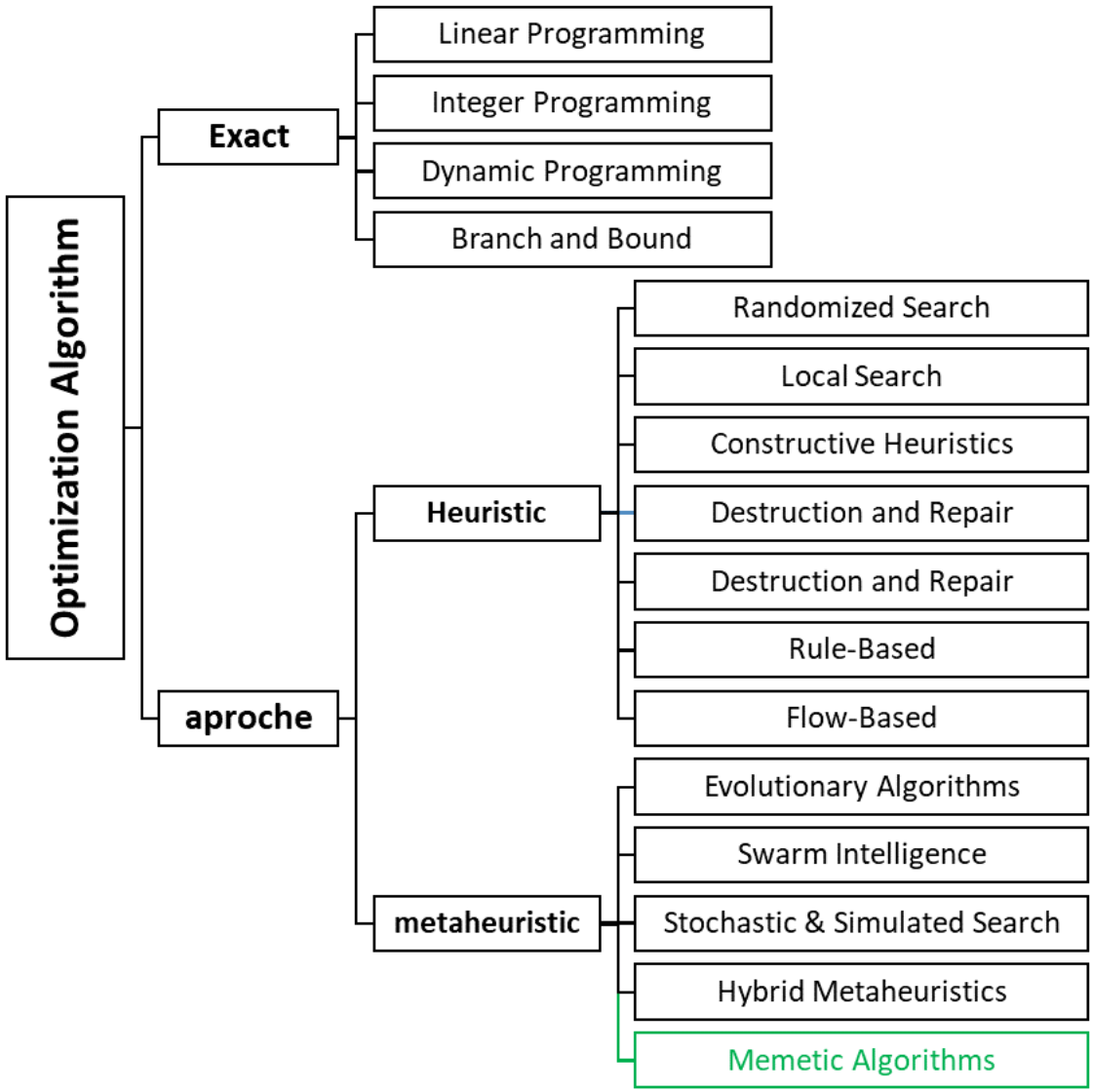

Figure 2.3: Classification of Algorithms According to Mathematical Structure.

2.8 Memetic Algorithm

2.8.1 Definition

A memetic algorithm is a hybrid optimization algorithm that combines evolutionary algorithms, which represent global search, with local search methods to improve solutions through iterative parallel enhancements. Genetic operations such as crossover and mutation are inte-

grated with local search techniques to enhance the algorithm's effectiveness in exploring the solution space. This integration can lead to faster and more accurate improvements compared to traditional genetic algorithms that focus only on global search.

The main idea is to use evolutionary strategies, such as genetic operations, to explore the overall solution space, and then apply local improvement mechanisms to refine each individual solution within that space. Local search serves as a guided tool to obtain more precise solutions through methods like iterative improvement or deep search within a specific neighborhood. This makes the memetic algorithm more capable of solving complex, high-dimensional problems[17].

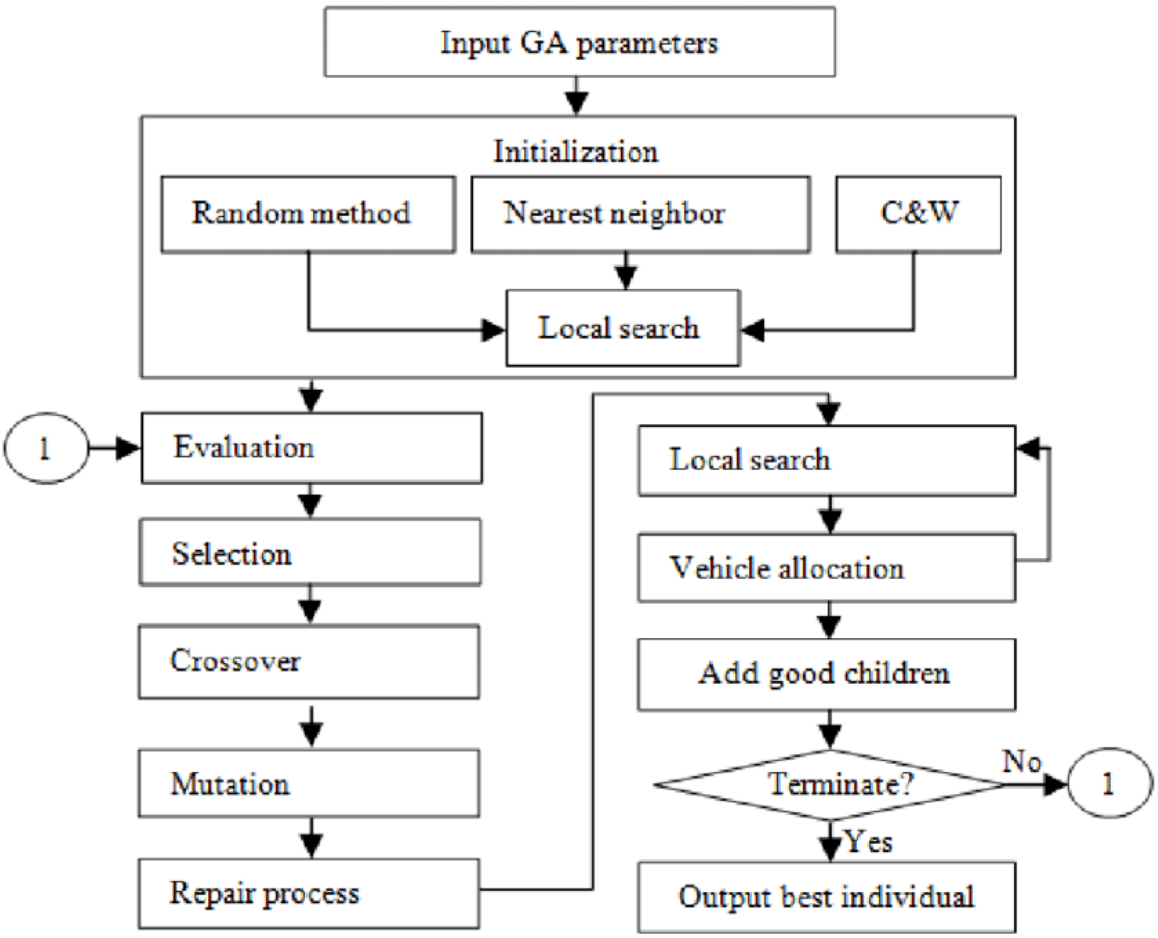

Figure 2.4: Structure of the Memetic Algorithm for Optimization[18].

2.8.2 Evolutionary Algorithms

they are metaheuristic algorithms that rely on mechanisms inspired by natural biological evolution, such as selection, crossover, and mutation. They are used to solve optimization problems with large, nonlinear, and complex search spaces[19].

The main types of evolutionary algorithms include:

- Genetic Algorithms (GA)
- Evolution Strategies (ES)
- Genetic Programming (GP)
- Evolutionary Programming (EP)

2.8.3 Local Search

Local Search is a metaheuristic approach used to solve combinatorial and nonlinear optimization problems. It relies on exploring the neighborhood of a single solution with the goal of iteratively improving it. Starting from an initial solution, the search is redirected at each iteration to neighboring solutions based on a quality measure (objective function). The process continues until no further improvement can be found within the local neighborhood, often resulting in convergence to a local optimum. Despite its efficiency and simplicity, local search is prone to getting trapped in local optima, limiting its ability to find the global optimum, especially in complex search spaces. Therefore, it is often combined with other techniques that enable escaping these traps and achieving better solutions[20].

Examples of algorithms based on the local search principle:[20]

- **Hill Climbing:** Moves to the best immediate neighboring solution at each step until no further improvements are possible.
- **Simulated Annealing:** Occasionally accepts worse solutions to avoid getting stuck in local optima, inspired by the annealing process in metallurgy.
- **Tabu Search:** Utilizes a "tabu list" to prevent revisiting previous solutions, aiding in escaping local optima.
- **Guided Local Search:** Modifies the objective function to penalize recurring undesirable features in solutions, guiding the search towards unexplored areas.

2.8.4 Difference Between Global Search and Local Search

In the context of optimization, particularly within MAs, it is essential to distinguish between global search and local search strategies. Each plays a complementary role in exploring the solution space and refining results. The following table presents a comparative overview of their main characteristics, highlighting their respective strengths, limitations, and applications in coverage problems.

Item	Global Search	Local Search
Scope	Entire solution space	Neighborhood of one solution
Objective	Globally optimal solution	Locally good solution
Speed	Usually slower	Faster
Accuracy	Higher but time-consuming	Prone to local optima
Application in Coverage	Broad and diverse sensor deployment	Fine-tuning positions of certain sensors

Table 2.1: Comparison between Global and Local Search

2.8.5 The Difference Between Hybrid Metaheuristics and Memetic Algorithms

Hybrid Metaheuristics are approaches that rely on combining two or more metaheuristic techniques with the aim of improving performance and enhancing search efficiency. This combination can involve any types of algorithms, whether metaheuristic or not, and does not necessarily require the inclusion of a local search component. Therefore, hybrid algorithms primarily focus on integrating diverse techniques without a strict constraint on the nature of the combination.

In contrast, **MAs** represent a specific subclass of hybrid algorithms, characterized by the structured integration of metaheuristics with local search methods. The core idea behind MAs is to improve each solution generated by the metaheuristic locally before evaluating it. This mirrors the way human knowledge evolves by combining random evolution with individual refinement of each solution. Thus, MAs are not merely a random fusion of techniques, but they inherently require a local search phase to enhance each solution before evaluation.

2.9 Applications of Memetic Algorithms

MAs are used in various fields that require precise and efficient optimization, particularly in complex systems where a balance between global search and local search is essential. Among the most prominent applications are:

- **Scheduling and Operations Optimization :** MAs play a key role in optimizing work schedules and task allocation within industrial systems and data centers. They contribute to reducing production time, enhancing resource utilization, and improving logistics and flight schedules to lower costs and boost overall efficiency.

- **Engineering Design Optimization :** These algorithms are applied in the development of efficient mechanical structures and intelligent systems. They help fine-tune the behavior of precise control systems, such as those in robotics or autonomous vehicles, aiming to achieve higher performance at a lower cost.

- **Machine Learning and Artificial Intelligence :** They serve as powerful tools in enhancing the performance of AI models through fine-tuning parameters, especially in artificial neural networks. Additionally, they improve the efficiency of search algorithms in big data analysis, leading to more accurate and faster results.
- **Environmental and Sustainability Optimization :** They support the development of efficient energy management solutions in smart buildings and the optimal distribution of renewable energy sources like solar panels and wind turbines. This leads to improved system performance and reduced environmental impact.
- **Telecommunication and Network Optimization :** They help improve network performance by optimizing node placement and reducing signal interference, thus enhancing communication quality and reducing resource consumption. They also contribute to faster data transmission and greater network stability.
- **Wireless Sensor Network Coverage Optimization :** Applied to optimize sensor placement for maximum area coverage with a minimal number of nodes, while also improving routing efficiency and reducing energy consumption, thereby extending the network's operational lifetime.

2.10 Reason for Choosing the Memetic Algorithm for Coverage Optimization in WSN

The MA was selected in this research due to its ability to combine stochastic search strategies with local search techniques, making it particularly well-suited for the coverage optimization problem in WSNs. The main reasons for this choice are as follows:

- **Balanced Integration of Global and Local Search :** Unlike purely stochastic algorithms such as Genetic Algorithms (GAs), MAs integrate precise local search methods with global exploration. This hybrid approach allows for more efficient discovery of high-quality solutions.
- **Faster Convergence to Optimal Solutions :** The inclusion of local search mechanisms in MAs accelerates convergence toward optimal or near-optimal solutions, reducing the reliance on trial-and-error commonly found in other metaheuristics.
- **Avoidance of Local Optima Traps :** A common challenge in WSN coverage optimization is getting trapped in suboptimal local solutions. MAs help overcome this limitation by refining solutions through local search, increasing the chances of escaping such traps and achieving superior performance.

- **Proven Effectiveness in Previous Studies :** Numerous scientific studies have demonstrated that MAs outperform traditional and purely stochastic algorithms in coverage optimization problems, reinforcing their suitability and reliability for this research topic.

2.11 Conclusion

In this chapter, we established a solid knowledge foundation in the field of optimization, starting from the challenges related to WSNs discussed in Chapter One. We reviewed the fundamental concepts of optimization, its problem components, and explored its classifications in terms of time complexity and mathematical structure. A significant focus was dedicated to the MA due to its effectiveness in addressing the complexities of coverage optimization and energy consumption reduction in WSNs. This conceptual framework represents a crucial step that paves the way for the practical aspect, through studying the application of the MA in a real-world scenario and analyzing its performance.

Chapter 3: Design of a Memetic Algorithm for Solving the Coverage Problem

3.1 Introduction

Building on the theoretical foundations presented in the previous chapters regarding WSNs and optimization techniques, this chapter focuses on the design of a memetic algorithm tailored to address the coverage problem. The work involves a precise mathematical modeling of the problem, along with the presentation of the algorithm's structure and components, from solution representation to generation, evaluation, and search strategies. This design will be used in future testing and evaluation within real-world scenarios.

3.2 Problem Modeling as an Optimization Problem

3.2.1 General Problem Definition

The problem consists in optimizing the coverage of a given area using a WSN while minimizing the number of deployed sensors. This challenge is formulated as a multi-objective optimization problem, with objectives including maximizing coverage, minimizing the number of sensors, and reducing excessive coverage overlap.

It is important to note that a certain level of overlap between sensor coverage areas is essential for the proper functioning of the network. Without it, communication between sensors or full coverage integrity may fail. However, excessive overlap unnecessarily increases the number of sensors required and thus the overall cost of the network, making it a key factor to control and optimize.

3.2.2 Problem Inputs

To effectively execute the mathematical model, we need a set of input data that defines the criteria and constraints within which the model will operate. These input data include:

- A: The target area to be covered (divided into rectangle-shaped cells or points).
- $S = \{s , s , ..., s\}$: The set of possible locations for placing sensors.
- R: The sensing radius of each sensor.
- B: The number of available sensors (optional, if we aim to reduce it as well).

3.2.3 Decision Variables

The possible decision variables within the problem are represented as follows:

- $S_{ij} \in \{0, 1\}$: Solution matrix, where 1 represents the presence of a sensor, and 0 represents otherwise.
- $C_{ij} \in N$: Coverage matrix, where each value represents the number of sensors that cover a specific area.

3.2.4 Objective Function

The objective function in this multi-objective model is to **maximize the coverage area** within a defined region using the **minimum number of sensors**, while ensuring acceptable coverage according to the specified criteria.

3.2.5 Constraints

- Reject solutions that include isolated sensors or groups of sensors.
- Reduce overlap between coverage areas by excluding solutions in which the same region is covered by an excessive number of sensors.
- Improve the coverage ratio by discarding solutions where the coverage falls below a predefined minimum threshold.

3.3 Mathematical Problem Formulation

We consider the monitored area as a two-dimensional grid of square cells. The mathematical model uses the following definitions:

- $A = [a_{ij}]$: A matrix representing the target area divided into square cells.
- n, m: The dimensions of the grid (number of rows and columns).
- $S = \{s_1, s_2, \ldots, s_k\}$: A set of possible sensor placement positions, where each sensor has a fixed coverage radius R.
- $C = [c_{ij}]$: The coverage matrix, where each entry c_{ij} indicates the number of sensors covering cell a_{ij}.
- $S_{ij} \in \{0, 1\}$: A binary decision matrix where $S_{ij} = 1$ indicates a sensor is placed at position (i, j), and 0 otherwise.

3.3.1 Objective Function

The goal is to maximize the overall coverage of the target area while minimizing the number of sensors used. This is formulated as a multi-objective problem:

- **Maximize** the number of covered cells in the grid:

$$\text{Coverage Rate} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\delta(c_{ij} \geq 1)}{n \times m}$$

  Where $\delta(\cdot)$ is the indicator function, returning 1 if the condition is true, and 0 otherwise.

- **Minimize** the total number of sensors deployed:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} S_{ij}$$

3.3.2 Constraints

To ensure a valid and efficient configuration, the following constraints are applied:

- **No isolated sensors:** Sensors must not be placed in positions where they do not contribute to coverage or network connectivity.

- **Overlap control:** Avoid excessive redundancy by limiting the number of sensors covering the same cell:

$$c_{ij} \leq C_{max}, \quad \forall i, j$$

  where C_{max} is a predefined maximum overlap threshold.

- **Minimum coverage threshold:** Discard solutions where the coverage rate falls below a minimum acceptable level C_{min}:

$$\frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\delta(c_{ij} \geq 1)}{n \times m} \geq C_{min}$$

3.3.3 Search Space Size

In WSN coverage optimization, the search space encompasses all possible sensor placement configurations within the monitored area. When the field is modeled as a two-dimensional grid consisting of N discrete cells, each of which can either contain a sensor or remain empty, the total number of possible configurations is 2^N. For instance, a grid of size 20×20 corresponds to $N = 400$, yielding a search space of 2^{400} distinct combinations.

This formulation reflects the binary nature of placement decisions at the cell level. As the number of cells increases, the search space expands exponentially, a characteristic inherent to combinatorial configuration problems of this kind.

3.3.4 Problem Complexity and NP-Hardness

The coverage optimization problem in WSNs is classified as **NP-Hard**, primarily due to the vastness of its search space and the combinatorial nature of sensor placement. As the number of potential deployment positions increases, the number of possible configurations grows exponentially, making the problem computationally intractable for exact algorithms.

Beyond the sheer size of the solution space, the problem also bears structural resemblance to several well-established NP-Hard problems. Notable among these are:

- **Set Connected Cover Problem:** A problem concerned with selecting a subset of sensors that ensure complete area coverage while maintaining a connected communication topology among themselves. It incorporates two intertwined objectives:
  - *Coverage:* Ensuring that every point of interest is monitored by at least one sensor
  - *Connectivity:* Guaranteeing that the selected sensors form a single connected component in the communication graph.
- **Art Gallery Problem:** A classical geometric problem that involves determining the minimum number of guards (or sensors) needed to observe every point within a polygonal environment, under line-of-sight constraints.

The presence of these structural parallels and the complexity involved in navigating the vast configuration space affirm the classification of the problem as NP-Hard.

3.4 Algorithm Used

3.4.1 Global Search

The GA is a population-based metaheuristic inspired by the process of natural selection. It operates by evolving a set of candidate solutions over successive generations through biologically inspired operators such as selection, crossover, and mutation. GAs are particularly effective in solving complex optimization problems where the search space is large, nonlinear, or poorly understood, as they do not rely on gradient information or problem-specific heuristics. Their strength lies in balancing exploration and exploitation, enabling them to efficiently approximate global optima[21].

The GA was adopted for the global search phase due to its robust exploration capabilities, which allow it to escape local optima. This algorithm benefits from the following key principles:[21]

- **Variation**: It maintains a diverse population of solutions, reducing the risk of premature convergence to local optima.
- **Adaptation**: Over generations, solutions evolve towards higher quality through selection pressure, without requiring explicit guidance.
- **Heredity**: Inheritance of "successful" traits from generation to generation through crossover and mutation gradually improves solution quality.

The main steps are as follows:[21]

1. **Initial Population**: Generate an initial set of random solutions (individuals).
2. **Evaluation**: Calculate the objective function for each individual.
3. **Selection**: Choose the best individuals based on fitness to participate in reproduction.
4. **Crossover**: It is a key genetic operator used to combine the genetic information of two parent solutions to generate new offspring. It is designed to mimic biological reproduction and promotes the exploration of new areas in the solution space by recombining existing features. The goal is to inherit beneficial traits from both parents and produce potentially better-performing children. There are various types of crossover mechanisms. The most common are:[21]

   Single-point crossover: A random crossover point is selected, and all genes beyond that point are exchanged between the two parents.

   Example:

   Parent 1: [1 0 0 | 1 1 0 1]
   Parent 2: [0 1 1 | 0 0 1 0]
   ↓
   Offspring 1: [1 0 0 | 0 0 1 0]
   Offspring 2: [0 1 1 | 1 1 0 1]

   Two-point crossover: Two crossover points are selected, and the segment between them is swapped between the parents.

Example:

Parent 1: [1 | 0 0 1 | 1 0]
Parent 2: [0 | 1 1 0 | 0 1]
↓
Offspring 1: [1 | 1 1 0 | 1 0]
Offspring 2: [0 | 0 0 1 | 0 1]

5. **Mutation**: It is a secondary genetic operator that introduces small, random changes to a solution. It serves to maintain genetic diversity within the population and helps prevent premature convergence by exploring new solution paths that may not be reachable through crossover alone. Mutation ensures that the algorithm does not get trapped in local optima[21].

   Example:

Original: [1 0 1 1 0 1 0]
↓
Mutated: [1 0 1 1 0 0 0]

3.4.2 Local Search

The local search is carried out using the Hill Climbing Algorithm (HCA), which proceeds as follows:[20]

1. **Initial Solution**: Select a candidate solution from the global search phase.
2. **Neighborhood Generation**: Create a set of neighboring solutions.
3. **Evaluation**: Compute the objective function for each neighbor.
4. **Move to Better Neighbor**: If a neighbor has better fitness, replace the current solution with it.
5. **Repeat**: Continue until no further improvement is possible or a maximum number of iterations is reached.

3.4.3 Exploration-Exploitation Balance Strategy

To enhance overall algorithm performance, a hybrid strategy inspired by Bee Algorithm behavior is employed to dynamically balance exploration and exploitation. In each generation, the population is divided into three subgroups:

- **Group 1(Focused Exploitation):** Both the Genetic Operations and Local Search are applied to these individuals for intensive improvement of promising solutions.
- **Group 2(Broad Exploration):** These individuals are only processed by the Genetic Operations, promoting diversity and enabling the discovery of new solution regions.
- **Group 3(Re-initialization)** – This subset is regenerated randomly to periodically refresh the population and re-initiate search in unexplored areas.

This structure allows computational resources to be allocated effectively: improving high-quality solutions while also ensuring exploration is not neglected.

3.4.4 General Algorithm

After detailing the global search mechanism, the local improvement strategy, and the exploration–exploitation balance, we can now present the overall structure of the proposed MA.

The algorithm begins with the generation of an initial population of candidate solutions, followed by fitness evaluation using the objective function. Selection directs the evolutionary process, while crossover and mutation operations introduce genetic diversity to explore the solution space. After each generation, local search is selectively applied to promising individuals, accelerating convergence toward optimal or near-optimal solutions without sacrificing population diversity.

This process continues iteratively until the predefined generation limit is reached.

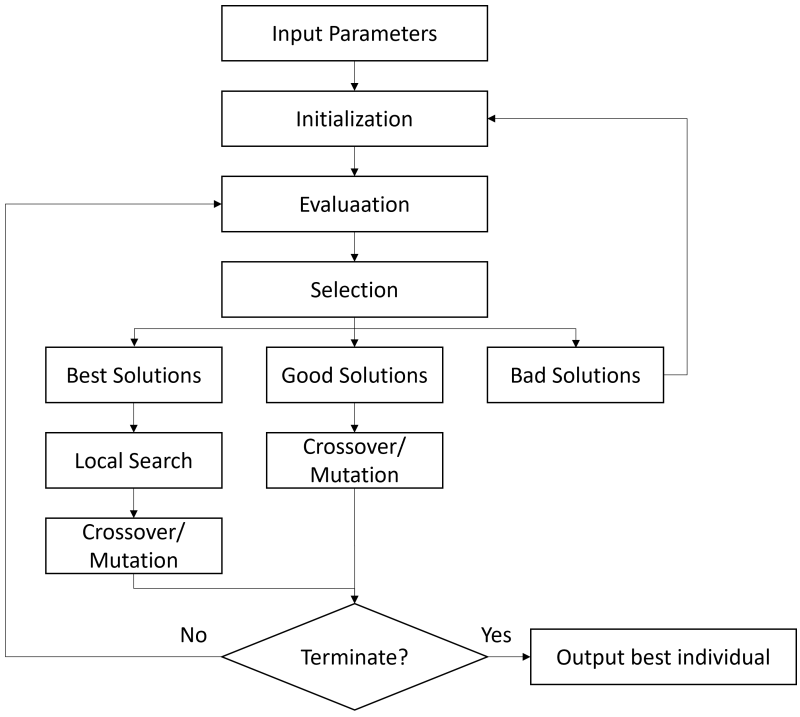

Figure 3.1: The proposed algorithm structure to solve the coverage problem in WSN

3.5 Applying the Algorithm to the Coverage Problem

3.5.1 Solution Encoding

The solution is represented using a two-dimensional model consisting of two main variables:

- **First variable**: A binary matrix representing the domain grid, where each cell indicates the presence or absence of a sensor. The value is 1 if a sensor is placed in that cell, and 0 otherwise. This variable reflects the actual distribution of sensors across the field.
- **Second variable**: A numerical matrix representing the coverage, where each cell contains the number of sensors covering it. These values are calculated based on the positions of the sensors in the first matrix and their coverage radius. This matrix enables the evaluation of the overall coverage quality of the proposed solution.

To illustrate how the solution is encoded, we present a practical example of a sensor network with a size of 10×10, where each cell in the grid can either contain a sensor or remain empty. We assume that each sensor has a sensing radius of 3, meaning it covers neighboring cells within a circular area of diameter 6 centered at its location.

In this example, we place only two sensors: the first at position (5,4) and the second at position (8,8). In the sensor matrix, the value "1" is assigned to these positions, and the value "0" elsewhere. The matrix is as follows:

$$S = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Based on these sensor locations and the defined sensing radius, we construct the coverage matrix, which reflects the number of times each cell is covered by one or more sensors. The resulting matrix is as follows:

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & s & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & s & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Note: The symbol "s" was used in the example to help visualize the sensor positions within the network. It will not be used in any subsequent computations or processing.

This dual representation provides a clear separation between the spatial structure of the solution (i.e., where the sensors are placed) and its performance evaluation (i.e., how well the domain is covered), which facilitates assessment and improvement at every stage of the algorithm.

3.6 Initial Settings and Solution Generation

3.6.1 Initial Settings

Before applying the algorithm to the problem, several parameters must be defined to control both the environment and the evolutionary process. These initial parameters are:

- n: (*int*) the width of the monitored area (network width);
- m: (*int*) the height of the monitored area (network height);
- r: (*double*) the sensing radius of each sensor
- pop_size: (*int*) the number of solutions in the population;
- $MaxIteration$: (*int*) the maximum number of iterations (generations);
- $MaxLocalIteration$: (*int*) the maximum number of iterations in local search;
- cp: (*double*, value in the interval $(0, 1)$) the probability of applying crossover;
- mp: (*double*, value in the interval $(0, 1)$) the probability of applying mutation;
- μ: (*double*) the packing density, representing how efficiently sensors can be arranged to cover the area. used to estimate the number of sensors needed.

These values are either empirically chosen based on preliminary experiments or adjusted later through parameter tuning to optimize algorithm performance.

3.6.2 Initial Solution Generation

In the initial phase of the algorithm, a set of candidate solutions -referred to as the **Initial Population**- is generated. Each candidate represents a possible deployment of sensors across the monitored area. These solutions serve as a starting point for subsequent optimization, and this stage constitutes the initial solution upon which further refinements will be applied.

To begin, the approximate number of sensors required to cover the area is estimated using the following formula:

$$\text{estimatedSensors} \leftarrow \frac{n \cdot m}{\pi \cdot r^2 \cdot \mu}$$

Here, μ denotes the packing density, which reflects the efficiency of sensor placement depending on the assumed layout pattern (for a detailed discussion, see Appendix A.1). It accounts for potential overlap or gaps in coverage and adjusts the raw area-to-coverage ratio accordingly.

Once the estimated number is calculated, sensors are randomly placed within the grid. During this placement, positions that already contain sensors or lie within regions already covered are rejected to reduce unnecessary overlap. The coverage matrix is then recomputed to track the number of sensors covering each individual cell in the grid (for more details, see Appendix A.2).

However, this random distribution often leaves portions of the field uncovered. To address this, the algorithm uses the *Breadth-First Search (BFS)* strategy to systematically detect and analyze uncovered regions. BFS plays a critical role here, it scans the entire grid to identify connected clusters of uncovered cells, grouping them as distinct regions based on adjacency. This allows the algorithm to assess the shape and extent of each uncovered area, rather than treating empty cells individually.

For each uncovered region found by BFS, the algorithm checks whether its size exceeds the maximum area a single sensor can cover (i.e., πr^2). If so, the cell that would contribute the greatest increase in coverage -by covering the highest number of currently uncovered cells- is selected as the placement point for a new sensor. This process continues iteratively until all major uncovered regions are sufficiently addressed (see Appendix A.3 for further details).

This configuration, refined through BFS-guided placement, constitutes the initial solution to the problem and serves as the baseline for further optimization phases within the memetic framework.

These ideas are embodied in the following algorithm:

Algorithm 1 Initialization Algorithm

Input: m, n: Network dimensions
r: Sensor radius
μ: Packing density

Output: S_{ij}: A 2D binary matrix representing the initial sensor placement
C_{ij}: A 2D matrix indicating the coverage of each cell

Step 1: Estimate Sensor Count

$$estimatedSensors \leftarrow \frac{n \cdot m}{\pi \cdot r^2 \cdot \mu}$$

Step 2: Random Sensor Placement

for $k \leftarrow 1$ **to** $estimatedSensors$ **do**
  Select a random cell (i, j) in the grid
  while $S_{ij} \neq 0$ *and* $C_{ij} \neq 0$ **do**
    Randomly select another cell (i, j)
  $S_{ij} \leftarrow 1$
  Update the coverage C_{ij}

Step 3: Compute Initial Coverage

for *each cell* (i, j) *in the grid* **do**
  if *a sensor is placed at* (i, j) **then**
    for *each cell* (x, y) *within radius* r *of* (i, j) **do**
      if *EuclideanDistance*$(i, j, x, y) \leq r$ **then**
        $C_{xy} \leftarrow C_{xy} + 1$

Step 4: Smart Sensor Addition

while *uncovered regions exist* **do**
  Identify uncovered regions using BFS
  if *uncovered region area* $>$ *max sensor coverage* **then**
    Select the best cell to maximize new coverage
    Place a sensor at that location
    Update C_{ij} accordingly

return S_{ij}, C_{ij}

Note: This algorithm balances between random diversity and targeted correction to ensure a reasonable initial coverage that accelerates convergence in later optimization steps.

As an example, the solutions resulting from this stage may look as follows, where each 2D matrix represents a proposed initial configuration of sensor deployment within the grid (grid size: 10×10). In this illustrative case, the population consists of three different solutions, each providing a distinct distribution of sensors across the grid.

Each value of 1 in the matrices below indicates the presence of a sensor at the corresponding position, contributing to the overall coverage, while a value of 0 denotes the absence of a sensor at that location. These initial solutions serve as a foundation for further refinement in subsequent stages of the algorithm.

Population =

$$\left[\begin{bmatrix}0&0&0&0&1&0&0&0&0&0\\0&0&0&0&0&0&1&0&0&0\\0&1&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&1&0&0\\0&0&0&1&0&0&0&0&0&0\\1&0&0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&0&0&1\\0&0&1&0&0&0&0&0&0&0\\0&0&0&0&0&1&0&0&0&0\\0&0&0&0&0&0&0&1&0&0\end{bmatrix}\begin{bmatrix}0&0&1&0&0&0&0&0&0&0\\0&0&0&0&1&0&0&0&0&0\\1&0&0&0&0&0&0&0&1&0\\0&0&0&1&0&0&0&0&0&0\\0&0&0&0&0&1&0&0&0&0\\0&1&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&0&0&1\\0&0&0&0&0&0&1&0&0&0\\0&0&0&1&0&0&0&0&0&0\end{bmatrix}\begin{bmatrix}1&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&1&0&0&0&0\\0&0&0&1&0&0&0&0&0&0\\0&0&0&0&0&0&1&0&0&0\\0&1&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&1&0&0&0&0&0\\0&0&0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&0&0&1\end{bmatrix}\right]$$

3.6.3 Population Evaluation

Evaluation is the process of analyzing and assessing solutions or models to determine their effectiveness and efficiency in achieving the desired goals, based on specific criteria. In the context of WSN coverage, the quality of each solution (fitness) is evaluated based on a combination of three ratios: coverage ratio, the number of sensors, and the overlap ratio of sensor coverage.

- α: A positive weight assigned to the coverage ratio. The higher the coverage, the better the solution quality; hence, this weight is positive.
- β: A negative weight assigned to the relative number of sensors (sensorRatio), since the objective is to minimize the number of deployed sensors. Therefore, more sensors decrease the fitness value.
- γ: A negative weight assigned to the overlap ratio (OverlapRatio), as overlapping coverage is considered undesirable and reduces solution efficiency.

These values are combined using a weighted sum, where positive weights promote desirable characteristics and negative weights penalize unwanted ones, to compute the final fitness score.

Before fitness evaluation, an additional validation step is applied to ensure the basic quality of each generated solution. Configurations where some sensors are placed in disconnected clusters that cannot communicate with the rest are eliminated (see Appendix A.4 for further details). Similarly, if certain areas are overwhelmed with overlapping coverage beyond what is reasonable, or if the overall coverage fails to reach an acceptable level, the solution is discarded and a new one is generated.

This regeneration process continues until a satisfactory configuration is obtained or a predefined limit is reached, in which case a value of -1 is returned.

This step is optional and only aims to enhance the overall feasibility and efficiency of the solutions.

Algorithm 2 Population Evaluation

Input: S_{ij}: Sensor placement matrix
C_{ij}: Coverage matrix
C_{max}: maximum number of sensors allowed to cover a single cell
C_{min}: the minimum required coverage ratio

Output: $fitness_i$: A 1D vector representing the fitness of each solution in the population

Step 1: Regeneration on Isolation, Redundancy, or Low Coverage
while *hasIsolatedGroups(S_{ij})* ***or*** $maxCoverage > C_{max}$ ***or*** $coverageRatio > C_{min}$ **do**
    S_{ij}, C_{ij} = generateInitialSolution()

Step 2: Compute Coverage Metrics
for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** m **do**
        if $C_{ij} > 0$ **then**
            $coveredCells \leftarrow coveredCells + 1$
        if $S_{ij} == 1$ **then**
            $N\_Sensors \leftarrow N\_Sensors + 1$
        if $C_{ij} > 1$ **then**
            $OverlapCells \leftarrow OverlapCells + 1$

Step 3: Compute Evaluation Ratios

$$coverageRatio \leftarrow \frac{coveredCells}{totalCells}$$

$$sensorEfficiency \leftarrow \frac{N\_Sensors}{estimatedSensorCount}$$

$$OverlapRatio \leftarrow \frac{OverlapCells}{coveredCells}$$

Step 4: Compute the Fitness Score
$fitness_i \leftarrow \alpha \cdot coverageRatio + \beta \cdot sensorEfficiency + \gamma \cdot OverlapRatio$

return $fitness_i$

3.6.4 Selection Method

In the selection stage, the population is strategically divided into three subgroups, each undergoing a distinct optimization process. This partitioning is inspired by **bee colony behaviour** and aims to maintain a dynamic balance between exploration and exploitation. By combining genetic operations with localized refinement and random regeneration.

- The top 40% of individuals are subjected to genetic operations (Crossover & mutation) and a local search procedure (Hill Clamping). This dual treatment allows for intensified exploitation of high-quality regions in the solution space, promoting intensification.
- The next 50% of the population are evolved using genetic operations only, with no local refinement. This encourages broader exploration of the solution space.

- The remaining 10% of individuals are entirely regenerated from scratch, introducing fresh candidate solutions into the population and enhancing diversification.

Algorithm 3 Selection and Population Partitioning Strategy (Corrected)

Input: $popSize$: Population size
$fitness_i$: A 1D fitness vector representing the quality of each solution
Pop_{ijk}: A 3D matrix representing all individuals in the population
Output: A new population after selection and diversification

Step 1: Sort Population in descending order based on $fitness_i$

Step 2: Select the Best 90%
$bestPop_{ijk} \leftarrow$ top 90% of Pop_{ijk} based on $fitness_i$

Step 3: Apply Genetic Operations
$offspring_{ijk} \leftarrow$ applyGeneticOperations($bestPop_{ijk}$);

Step 4: Improve Top 40%
for $i \leftarrow 0$ **to** $0.4 * popSize$ **do**
    $offspring_i \leftarrow$ applyLocalSearch($offspring_i$);

Step 5: Re-generate Bottom 10%
$newRandomSolutions_{ijk} \leftarrow$ generateNewSolutions($0.1 * popSize$);

Step 6: Merge All Segments
$newPop_{ijk} \leftarrow offspring_{ijk} \cup newRandomSolutions_{ijk}$;

return $newPop_{ijk}$

3.6.5 Genetic operations

a) Crossover

The crossover operation is a fundamental part of GAs, enabling the exchange of genetic material between two parent solutions to generate new offspring. In this implementation, a single-point crossover strategy is used.

The process begins by selecting a random crossover point along the rows of the solution matrices. This point determines where the exchange of genes (sensor configurations) will occur.

Starting from the selected row index ($crossoverPoint$) to the end of the matrices, corresponding elements in $Solution1$ and $Solution2$ are swapped. This operation mixes features from both parents, potentially combining their strengths and producing new, diverse solutions.

By carefully exchanging segments of the solutions rather than altering them randomly, this method helps preserve useful structures and contributes to the evolutionary improvement of the population.

Algorithm 4 Crossover Algorithm (Single Point Crossover)

Input: $Solution1_{ij}$: The first parent solution
$Solution2_{ij}$: The second parent solution
Output: Updated $Solution1_{ij}$ and $Solution2_{ij}$ after crossover

Step 1: Select random Crossover Point
$crossoverPoint \leftarrow randomInteger(0, totalRows - 1)$

Step 2: Perform Gene Exchange
foreach (i, j) *where* $i \geq crossoverPoint$ **do**
    swap($Solution1[i][j]$, $Solution2[i][j]$)

return $Solution1_{ij}$, $Solution2_{ij}$

b) Mutation

The mutation operation in this algorithm is designed to introduce slight modifications without significantly altering the structure of the solution. A candidate cell for mutation is selected, and the algorithm searches for the nearest active sensor (with a value of 1) starting from that cell toward the bottom-right corner. If no sensor is found in that direction, the search continues from the top-left corner up to the selected cell. Once the nearest sensor is located, it is deactivated (its value is set to 0), introducing a subtle change in the overall sensor distribution.

Algorithm 5 Mutation Algorithm

Input: $Solution_{ij}$: The current solution matrix (grid of sensors)
Output: Updated $Solution_{ij}$ after mutation

Step 1: Select a Random Mutation Point
$mutationPoint \leftarrow randomInteger(0, totalRows - 1)$

Step 2: Find the Nearest Sensor
(a) for $x \leftarrow i$ **to** $rows - 1$ **do**
    for $y \leftarrow j$ **to** $cols - 1$ **do**
        if $Solution[x][y] == 1$ **then**
            $Solution[x][y] \leftarrow 0$ Break both loops

(b) If no sensor is found from (i, j) to the end of the matrix: Search from $(0, 0)$ to (i, j) using the same logic

return $Solution_{ij}$

c) Applying genetic operations

This procedure applies crossover and mutation operators to the current population with probabilities cp and mp, respectively. These operations generate a new set of offspring, thereby introducing genetic diversity and promoting exploration of the search space.

Algorithm 6 Crossover and Mutation Integration

Input: $bestPop_{ijk}$: List of the 90% best-performing individuals in the population
cp: Crossover probability
mp: Mutation probability
Output: New list of offspring after applying genetic operations

Step 1: Initialize $offspring_{ijk}$
Step 2: Iterate Over the Population in Pairs
for $k \leftarrow 0$ **to** $len(bestPop)$ ***By*** 2 **do**
  Step 3: Apply Crossover Based on cp
  if $randomReal(0,1) < cp$ **then**
    $(child1_{ij}, child2_{ij}) \leftarrow Crossover(bestPop_{ij}[k], bestPop_{ij}[k+1])$
  else
    $child1_{ij} \leftarrow bestPop_{ij}[k]$ $child2ij \leftarrow bestPop_{ij}[k+1]$
  Step 4: Apply Mutation Based on mp
  if $randomReal(0,1) < mp$ **then**
    $child1_{ij} \leftarrow Mutate(child1)$
  if $randomReal(0,1) < mp$ **then**
    $child2_{ij} \leftarrow Mutate(child2)$
  Step 5: Add Children to Offspring List
  $offspring_{ijk}$.Append($child1_{ij}, child2_{ij}$)
Step 6: Handle Odd Population Size
if $len(bestPop_{ijk})\%2 \neq 0$ **then**
  $offspring_{ijk}$.Append($bestPop_{ij}[-1]$)
return $offspring_{ijk}$

3.6.6 Local Search using Hill Climbing Algorithm

The local search procedure is based on the HCA, a heuristic that iteratively improves a given solution through localized modifications. The process begins with an initial sensor placement and aims to incrementally enhance the overall network coverage. At each iteration, the algorithm selects a single sensor and explores its immediate neighborhood to determine whether relocating it to an adjacent position can yield a better global coverage configuration.

The neighborhood exploration is conducted across the eight surrounding directions (cardinal and diagonal). If an uncovered cell is found, the algorithm attempts to move the sensor toward that location. Conversely, if a cell with excessive coverage is detected, the sensor is shifted in the opposite direction to reduce redundancy. When an improvement is observed, the movement continues along the same direction. Otherwise, the procedure halts and retains the last position that resulted in a fitness gain. This operation is repeated for several randomly selected sensors, and the enhanced solution is returned once the local search concludes.

Example: The figure illustrates a section of the coverage grid where a sensor has been chosen for local refinement. As shown in the image, the algorithm evaluates all eight neighboring directions.

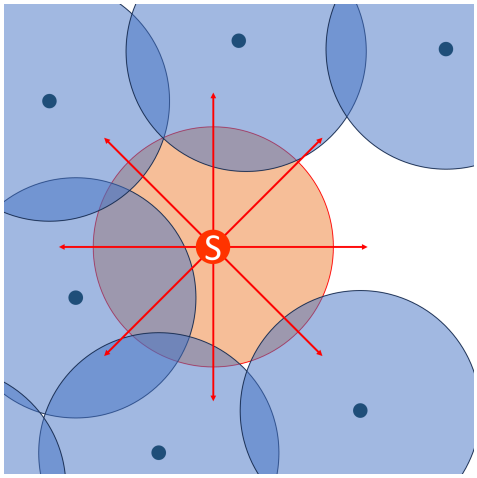

Figure 3.2: Sensor Neighborhood Exploration Phase.

In this particular instance, an overloaded cell was found to the east. As a response, the sensor was moved westward—its opposite direction—where an uncovered area was identified. Since this adjustment improved the overall fitness of the solution, the move was accepted.

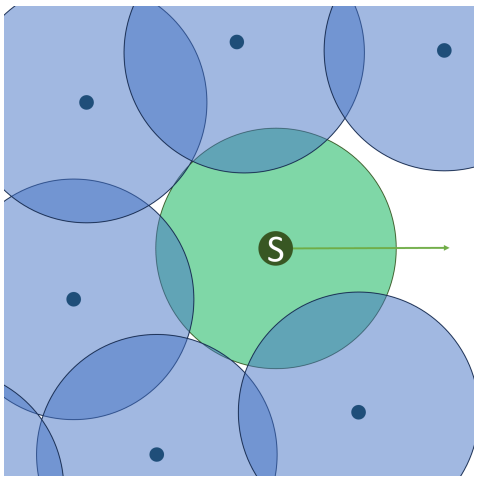

Figure 3.3: Sensor Relocation and Coverage Improvement.

This principle -relying on coverage-driven movements guided by local information- is implemented systematically in the following algorithm, which outlines the structured steps of the HCA-based local improvement strategy.

Algorithm 7 Hill Climbing Algorithm

Input: $Solution_{ij}$: Initial solution matrix
$Coverage_{ij}$: Coverage matrix corresponding to the solution
$maxIterations$: Maximum number of iterations
Output: $Solution_{ij}$: Improved sensor placement matrix

Step 1: Initialization
$current_{ij} \leftarrow Solution_{ij}$
$currentCoverage_{ij} \leftarrow Coverage_{ij}$
$currentFitness \leftarrow evaluate(current_{ij}, currentCoverage_{ij})$
for $iter \leftarrow 0$ **to** $maxIterations - 1$ **do**
  Step 2: Clone Current Solution
  $candidate_{ij} \leftarrow deepcopy(current_{ij})$
  Step 3: Select a Random Sensor
  repeat
    $x \leftarrow randomInteger(0, rows - 1)$
    $y \leftarrow randomInteger(0, cols - 1)$
  until $candidate[x][y] == 1$;
  Step 4: Try Moving the Sensor in 8 Directions
  foreach $(dx, dy) \in \{(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)\}$ **do**
    for $step \leftarrow 1$ **to** $2 \times radius$ **do**
      $nx \leftarrow x + dx \times step$
      $ny \leftarrow y + dy \times step$
      if $(0 \leq nx < rows)$ ***and*** $(0 \leq ny < cols)$ **then**
        if *currentCoverage* $[nx, ny] > 2$ **then**
          $mx \leftarrow x - dx$
          $my \leftarrow y - dy$
        else if *currentCoverage* $[nx, ny] = 0$ **then**
          $mx \leftarrow x + dx$
          $my \leftarrow y + dy$
        else
          continue
        if $(0 \leq mx < rows)$ ***and*** $(0 \leq my < cols)$ **then**
          Swap sensor from (x, y) to (mx, my)
          Update $candidateCoverage_{ij}$
          $candidateFitness \leftarrow evaluate(candidate_{ij}, candidateCoverage_{ij})$
          if $candidateFitness > currentFitness$ **then**
            $current_{ij} \leftarrow candidate_{ij}$
            $currentCoverage_{ij} \leftarrow candidateCoverage_{ij}$
            $currentFitness \leftarrow candidateFitness$
            break
return $current_{ij}$

In addition to its role during the iterative evolution process, the local search strategy is also applied as a final refinement step to the best solution obtained at the end of the optimization process.

This ensures that the most promising individual undergoes one last stage of focused enhancement, potentially correcting any minor inefficiencies or suboptimal sensor placements that were not addressed during earlier iterations.

By doing so, the algorithm increases the likelihood of achieving a high-quality final configuration that not only meets the coverage and connectivity requirements but also demonstrates improved compactness and spatial efficiency. This final local adjustment reflects the intensification principle, aiming to fully exploit the best solution's potential before termination.

3.6.7 Stopping Criteria

The termination of the memetic algorithm is governed by predefined stopping criteria, such as reaching a maximum number of iterations, achieving a satisfactory level of convergence, or finding an optimal solution.

Throughout the execution of the algorithm, the best solution found so far is continuously tracked and stored. Once the stopping condition is met -most commonly the maximum number of cycles- the algorithm returns the best recorded solution as the final output.

3.6.8 Conclusion

This chapter presented a comprehensive design of a memetic algorithm aimed at improving coverage in WSNs, through precise mathematical modeling and a clear, well-defined algorithmic structure. We consider this design a fundamental step towards the simulation and evaluation phase, where we will study the algorithm’s effectiveness in addressing core challenges and enhancing coverage in depth.

Chapter 4: Simulation Results and Scenario Testing

4.1 Introduction

In this chapter, we focus on the application of the MA to enhance the coverage of WSNs by presenting the adopted hardware and software environments and detailing the programming and execution process of the algorithm. We also describe the method used to generate the data for simulating the working environment, enabling the evaluation of the algorithm's performance and its effectiveness in addressing the coverage issue within such networks. This serves as a foundation for assessing its applicability in real-world scenarios.

4.2 work Environment

4.2.1 Hardware Resources

Two computers were used in this project, each dedicated to a specific task. Both devices operated using the same system:

Operating System: Windows 10, 64-bit edition

- **The first device:** a Lenovo laptop, was used for writing the report using LaTeX. It features:

  - **Processor**: Intel(R) Core(TM) i5-5300U @ 2.30GHz

  - **RAM**: 4.00 GB

- **The second device:** an HP laptop, was dedicated to running the code. It features:

  - **Processor**: Intel(R) Core(TM) i5-6300U @ 2.40GHz

  - **RAM**: 8.00 GB

4.2.2 Software Resources

In this project, the following software tools were used:

a) Programming Language

We used Python, an open-source, object-oriented programming language widely used for data analysis and algorithm implementation. Python's flexibility and extensive library support made it suitable for developing and testing our optimization algorithm[22][23].

b) Development Environment

We used Anaconda, a popular open-source distribution of Python, which includes essential tools such as Jupyter Notebook and pre-installed libraries like NumPy, Matplotlib, and Pandas. Anaconda simplified package management and facilitated smooth execution of the algorithm in an organized and reproducible environment[23].

4.3 Code Structure

The code implementing the MA for solving the coverage problem in WSNs was divided into several files to enhance readability, maintainability, and modularity. A modular programming approach was adopted, where each functional component is placed in a separate file. This structure allows for re-usability and easier updates without affecting the rest of the system.

The code is organized into the following files:

- **Variables.py:** Contains all global variables and constants used in the simulation, such as field dimensions, number of sensors, and sensing radius.
- **Initialization.py:** Includes procedures for initializing the population and randomly distributing sensors within the defined area.
- **Evaluation.py:** Provides the evaluation functions used to compute coverage rate and verify connectivity for each solution.
- **GeneticOperation.py:** Contains both crossover and mutation functions as part of the genetic evolution mechanisms to enhance population diversity and quality.
- **LocalSearch.py:** Implements the LS procedures using the HCA to refine solutions after genetic evolution.
- **main.py:** The main script that integrates all modules, runs the algorithm, displays results, and tracks performance.
- **Visualizer.py:** Provides a visual representation of the sensor network using **Matplotlib**. This visual tool helps assess the effectiveness and distribution of sensor coverage.

This structure ensures well-organized code and facilitates debugging and independent testing of each module.

4.4 Algorithm Workflow

Although the internal mechanisms of the proposed MA were described in detail in Chapter 3, this section provides a brief overview of its overall workflow to clarify the experimental execution steps.

The algorithm starts by randomly generating an initial population of solutions within the defined sensing field. Each individual is then evaluated using a fitness function that considers coverage, sensor count, and connectivity constraints.

The evolutionary loop applies selection, crossover, and mutation to form new populations, followed by a LS phase that refines selected solutions. This process repeats until the maximum number of global iterations is reached.

At the end of execution, the best solution is returned along with key performance metrics such as coverage ratio, number of sensors, overlap, fitness value, and execution time. A graphical visualization of the optimal sensor layout is also produced to facilitate result interpretation. The full implementation is available at: https://github.com/TakwaKhelifi/Memetic-WSN-Coverage-Optimization.

4.5 Simulation Parameters

To ensure a consistent experimental environment for evaluating the performance of the proposed MA, a set of fundamental settings related to the network structure and sensor characteristics has been fixed. These settings include the field dimensions $width$ and $height$ and the sensor coverage radius $radius$, ensuring a uniform distribution of coverage across the network area.

Additionally, a coverage constraint C_{min} has been set to regulate the quality of the generated solutions, while an upper bound on redundant coverage C_{max} has been established to avoid unnecessary overlap. The GA's characteristics have also been configured, Additionally, the selection probabilities for $top\_solutions$, $good\_solutions$, and $bad\_solutions$, as well as the GA parameters -crossover probability cp and mutation probability mp- were adjusted to achieve an effective balance between exploration and exploitation mechanisms.

The adopted values for these parameters are presented in the following table, and they are fixed as a common baseline for all experiments that will be conducted subsequently.

$width$	$height$	$radius$	C_{min}	C_{max}	top_S	$good\_S$	bad_S	cp	mp	μ
50	50	5	75%	5	30%	50%	20%	0.9	0.1	0.9

Table 4.1: Fixed Parameters Used in All Experiments

These parameters will remain constant throughout all subsequent experiments to ensure the stability of the testing environment, while the impact of other variables will be investigated through specific scenarios. These variables include the number of iterations, population size, local search iteration count, and the fitness function weights (α, β, γ).

4.6 Result Metrics

Three primary metrics were selected to evaluate the performance of the proposed MA, with the aim of providing a comprehensive assessment from multiple perspectives. Each metric will focus on studying the following aspects:

- **Coverage Ratio:** This metric will be analyzed to assess the algorithm's ability to achieve full coverage of the area, especially under varying parameter settings.
- **Number of Sensors:** The evolution of the number of sensors in the resulting solutions will be tracked to evaluate the algorithm's efficiency in minimizing resource usage.
- **Execution Time:** This metric will be studied to analyze the computational time required for each experiment, and to determine the practicality of applying the algorithm in real-world scenarios that require rapid response. It will also help understand the relationship between solution quality and the time needed to reach it, particularly when modifying settings such as population size or iteration limits.

4.7 Results and Analysis

4.7.1 Preliminary Study of Execution Time

This preliminary study aims to measure the execution time required to run the proposed MA under specific settings, in order to evaluate its computational efficiency.

The algorithm was executed using fixed values for the evaluation function weights. Three different experiments were conducted by varying the values of the population size (*popSize*), the maximum number of global iterations (*maxIterations*), and the number of LS iterations (*maxLocalIterations*). The characteristics of each experiment are presented in the table below.

Experiment	popSize	maxIterations	maxLocalIterations
1	30	100	10
2	50	150	15
3	70	200	20

Table 4.2: Experimental Settings: Population Size, Global Iterations, and LS Limits

Each experiment was executed five times, and both the execution time and the best fitness value obtained were recorded. The results present the average execution time under the given conditions, providing insights into the initial effect of these parameters on runtime performance.

Exp	1st	2nd	3rd	4th	5th	Average	BestFitnss
1	49.22	48.37	47.10	47.58	44.05	47.264	0.8016
2	152.99	153.96	153.24	154.43	156.48	154.22	0.8095
3	345.29	356.90	360.34	357.66	347.32	353.502	0.8126

Table 4.3: Execution Time and Best Fitness Across Experimental Configurations

As shown in Table 4.7.1, the execution time increases significantly with larger values of *popSize*, *maxIterations*, and *maxLocalIterations*. This is expected, as increasing these parameters leads to a larger search space and more computational effort.

The best fitness values obtained also show a slight improvement across the experiments, indicating a positive impact of these parameters on solution quality. However, the increase in execution time is more pronounced than the gain in fitness, especially between Experiments 2 and 3.

These observations suggest that while increasing the algorithm's effort may enhance solution quality, it also introduces a substantial computational cost. Therefore, selecting appropriate values for these parameters requires balancing runtime efficiency and solution effectiveness.

4.7.2 Impact of Local Search on Solution Quality

To assess the effectiveness of the LS phase, the MA was executed under two configurations: with and without the LS component. All other parameters were held constant, including population size, number of global iterations, and fitness function weights.

The purpose of this experiment is to determine whether the inclusion of local refinement contributes significantly to solution quality in terms of coverage ratio, number of sensors, and fitness value.

Configuration	Coverage	Sensors	Fitness	Execution Time (s)
With LS	0.97	40	0.80451	353.50
Without LS	0.853	41	0.7474	94.72

Table 4.4: Comparison of Results With and Without Local Search

The results show that the inclusion of the LS step improves the coverage and fitness, while increasing the execution time. This suggests that the LS plays a significant role in fine-tuning solutions and enhancing overall algorithm performance.

4.7.3 Impact of Fitness Function Weights

This section explores how different fitness function weight configurations affect the performance of the proposed MA. The fitness function consists of three components: α (coverage importance), β (sensor count penalty), and γ (overlap penalty). By adjusting these weights, the algorithm is guided to prioritize certain optimization objectives over others.

Six different weight configurations (scenarios) were tested. Table 4.5 describes each scenario and its focus.

	Scenario	α	β	γ	**Description**
1	Ideal Coverage	1.0	0.1	-0.05	Prioritizes achieving maximum coverage regardless of sensor count.
2	Balanced Coverage and Count	0.8	0.3	-0.05	Strikes a balance between coverage and number of sensors.
3	Economical Deployment	0.6	0.4	-0.05	Emphasizes fewer sensors, with less weight on coverage.
4	Overlap Minimization	0.85	0.15	-0.2	Targets reduced overlap while maintaining high coverage.
5	Neutral Configuration	0.7	0.2	-0.1	Moderately balances all objectives.

Table 4.5: Weight scenarios used to guide the memetic algorithm

Each scenario was tested in five independent runs. The results for number of sensors, achieved coverage, and overlap ratio are presented in Table 4.6.

Scenario	**Run**	**Sensor Count**	**Coverage (%)**	**Overlap (%)**
1	1	49	99.9	21.5
	2	48	99.3	20.8
	3	47	99	21.0
	4	49	99.9	21.8
	5	48	99.5	20.9
2	1	42	97.3	18.1
	2	43	97.6	18.4
	3	41	96.9	17.9
	4	42	97.2	18.2
	5	42	97.4	18.3
3	1	37	93.1	13.3
	2	36	92.8	13.0
	3	35	91.5	12.5
	4	36	92.6	12.8
	5	37	93.4	13.1
4	1	38	94.8	8.7
	2	39	95.2	9.0
	3	37	94.4	8.4
	4	39	95.3	9.1
	5	38	94.9	8.9
5	1	41	97.5	15.2
	2	42	98	15.6
	3	40	96.9	14.9
	4	41	97.3	15.1
	5	41	97.4	15.0

Table 4.6: Performance of each scenario across 5 independent runs

The results demonstrate that varying the weights of the fitness components significantly

affects the optimization outcomes.

- The **Ideal Coverage** configuration consistently achieves the highest coverage rates (above 99%) but at the cost of a higher number of sensors and increased overlap, which may be resource-intensive.

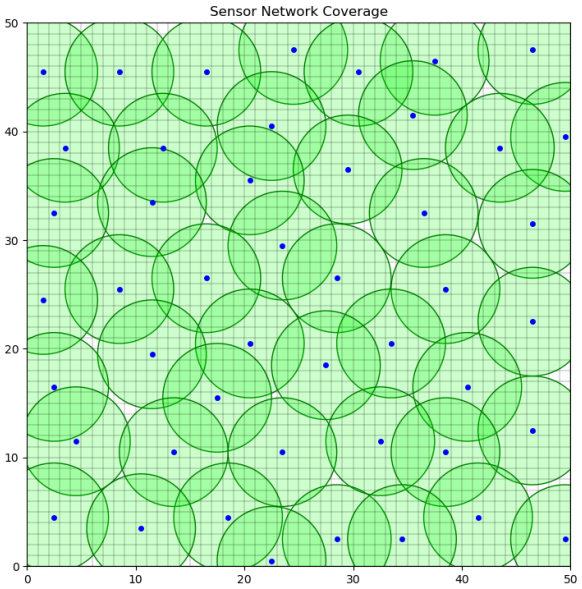

Figure 4.1: Sensor distribution in the Ideal Coverage scenario.

- The **Balanced Coverage and Count** scenario offers a strong compromise, maintaining excellent coverage while keeping the number of sensors and overlap at acceptable levels.

- **Economical Deployment** minimizes sensor usage, yet this comes with a noticeable drop in coverage, which might not be acceptable for critical monitoring applications.

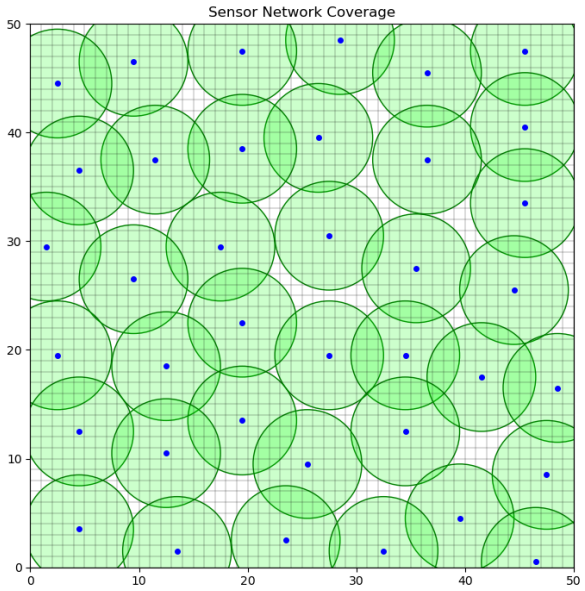

Figure 4.2: Sensor distribution in the Economical Deployment scenario.

- **Overlap Minimization** effectively reduces redundancy but tends to slightly under-perform in terms of total coverage.

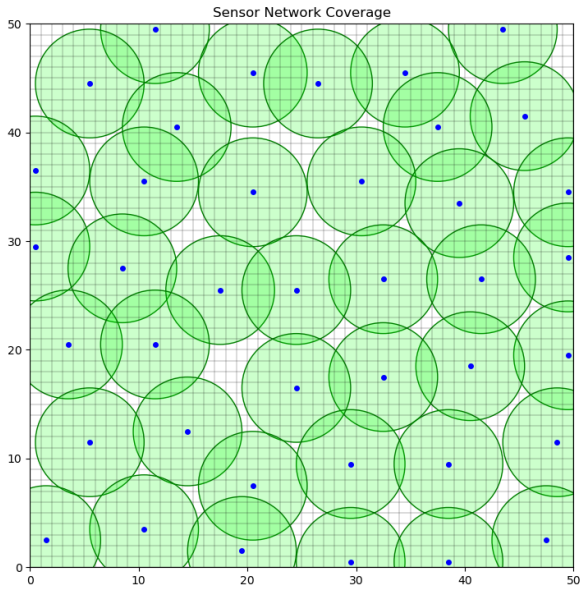

Figure 4.3: Sensor distribution in the Overlap Minimization scenario.

- Finally, the **Neutral Configuration** offers a well-rounded performance, with stable coverage and moderate sensor usage and overlap, serving as a balanced reference point.

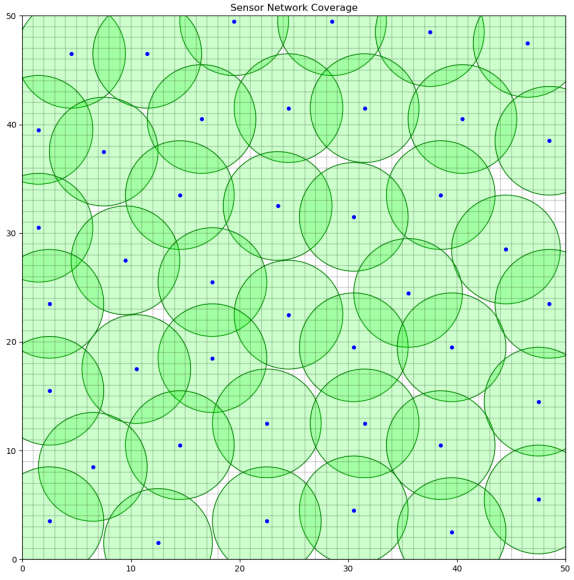

Figure 4.4: Sensor distribution in the Neutral Configuration scenario.

These findings emphasize the importance of selecting fitness weights based on the specific priorities of the application -whether prioritizing maximum sensing, resource efficiency, or minimal redundancy.

4.7.4 Effect of Selection Probabilities on Convergence Speed

This experiment investigates how different selection probability configurations affect the convergence behavior of the MA. The algorithm classifies individuals into three categories based on fitness ranking: *Top Solutions*, *Good Solutions*, and *Bad Solutions*. Each category is handled differently during the search process — top solutions undergo both genetic improvement and LS, good solutions are genetically improved, and bad solutions are discarded and replaced.

Four configurations were tested by adjusting the probabilities of selecting individuals from each category, while keeping all other parameters constant. The goal is to analyze how varying the balance between exploitation and exploration influences the number of iterations required to reach high-quality solutions.

Configuration	Top (%)	Good (%)	Bad (%)
C1	70	20	10
C2	20	60	20
C3	25	25	50
C4	40	45	15

Table 4.7: Selection Probability Configurations

Figure 4.5 provides a visual comparison between the convergence patterns and the execution times for each configuration.

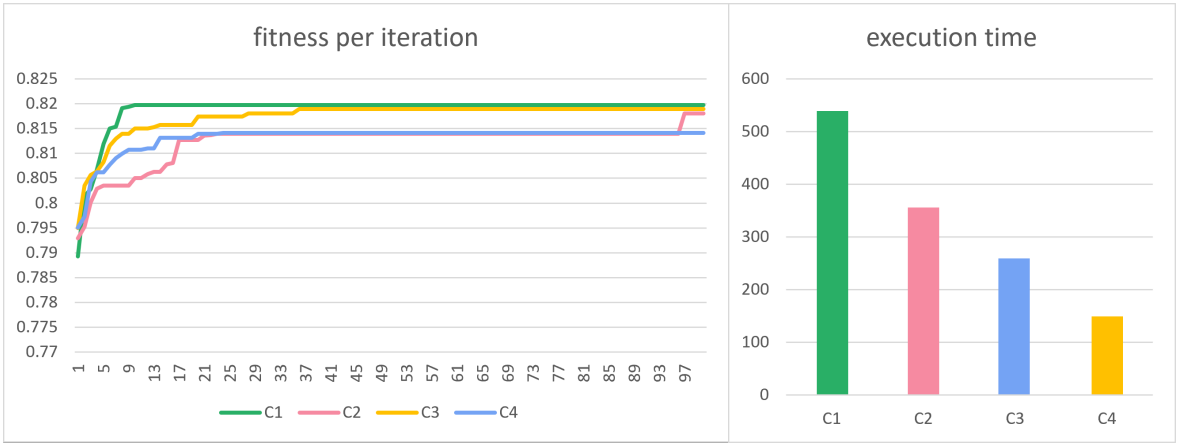

Figure 4.5: Effect of selection probabilities on convergence behavior and execution time.

The line chart on the left displays the fitness evolution over 100 iterations, highlighting how quickly and effectively each configuration converges toward optimal solutions. Configuration C1 demonstrates the fastest and most stable convergence, followed closely by C3 and C4, while C2 shows slower and less stable progress.

On the right, the bar chart illustrates the total execution time for each configuration. Notably, while C1 achieves superior solution quality, it does so with the highest computational

cost. In contrast, C4 balances good convergence with significantly reduced execution time, suggesting a more efficient trade-off between exploration and exploitation.

These findings emphasize the importance of tuning selection strategies according to problem characteristics and optimization goals. While aggressive exploitation may accelerate convergence, more balanced configurations can enhance solution diversity and prevent premature convergence -a factor that proved beneficial across various experiments.

4.7.5 Summary of Results and Observations

The experimental results confirm the effectiveness of the proposed memetic algorithm in addressing the multi-objective coverage optimization problem in WSNs. Several key insights can be drawn:

- Execution time analysis 4.7.1 showed that although the inclusion of local search slightly increases computation time, the gain in solution quality compensates for this cost, especially in large networks where coverage errors are more critical than marginal time increases.
- The impact of local search 4.7.2 was clearly reflected in the results: activating the Hill Climbing phase led to a consistent improvement in the fitness score across all configurations. This demonstrates that local search plays a crucial role in fine-tuning solutions produced by the genetic phase. It helps remove isolated sensors, smooth out redundant overlap, and close small coverage gaps, all of which contribute to higher solution precision.
- The weight scenarios 4.7.3 revealed that prioritizing coverage (higher weight on the coverage objective) consistently led to more robust results with minimal uncovered regions. On the other hand, giving more weight to minimizing sensor count resulted in reduced sensor usage but at the cost of incomplete coverage. This confirms the trade-off nature of the problem and shows that careful weight calibration is essential depending on the application.
- Selection probability tuning 4.7.4 demonstrated that convergence speed is highly sensitive to parent selection strategy. Favoring fitter individuals improved convergence but increased the risk of premature convergence. Introducing controlled randomness (e.g., tournament or rank-based selection) produced more stable results across independent runs, striking a balance between exploitation and diversity preservation.
- The algorithm's exploration–exploitation balance strategy proved effective: dividing the population into subgroups (focused improvement, broad exploration, and reinitialization) ensured that the search process avoided stagnation and covered a wide range of the solution space without sacrificing convergence.

- Overall, the algorithm achieved near-complete coverage (up to 99%) with significantly fewer sensors compared to the estimated upper bound (based on packing density), confirming that the intelligent integration of global and local search enables efficient coverage with minimal redundancy.

In summary, the results validate that the proposed memetic architecture:

1. Ensures consistent convergence toward high-quality solutions.
2. Adapts well to multi-objective constraints (coverage vs. sensor minimization).
3. Benefits significantly from local search without overfitting.
4. Provides flexibility in adjusting to different deployment priorities through weight tuning and selection dynamics.

4.8 Challenges and Limitations

Despite the satisfactory results obtained by the proposed memetic algorithm, a number of challenges and limitations were observed during experimentation:

- **Computational cost of local search:** While the integration of hill climbing improves solution quality, it increases the total execution time, especially in larger networks or with high population sizes. This can limit its applicability in real-time scenarios and may require adaptive mechanisms to reduce search frequency once convergence stabilizes.
- **Sensitivity to parameter settings:** The algorithm's performance strongly depends on key parameters such as population size, iteration limits, and crossover/mutation rates. Inappropriate values can lead to weak convergence or poor exploration. Due to time constraints, extensive tuning across all possible configurations was not feasible.
- **Lack of theoretical convergence proof:** As with most metaheuristics, there is no formal guarantee that the algorithm converges to a global optimum. Evaluation is based entirely on empirical results, which may vary with different initializations or problem instances.
- **Manual trade-off control:** Balancing conflicting objectives such as maximizing coverage versus minimizing sensor count relies on manually adjusting weight parameters. This limits the algorithm’s automation potential in adaptive or dynamic environments.

- **Limited generalization:** The current implementation assumes a static 2D grid with homogeneous, fixed-radius sensors. The algorithm does not account for dynamic environments, obstacles, or mobile nodes, which would require more complex modeling and search strategies.

- **Representation-realism gap:** A minor mismatch was observed between the theoretical model and practical sensor behavior. In rare cases, rounding in distance calculations caused a sensor to be falsely treated as connected (e.g., interpreting $2 \times radius + 0.4$ as fully connected). While not frequent, this highlights the need to refine the realism of connectivity checks in future extensions.

4.9 conclusion

This chapter presented a comprehensive evaluation of the proposed memetic algorithm through a wide range of simulations. We detailed the implementation context, algorithmic structure, and parameter choices, followed by analyses that explored various aspects such as execution time, coverage quality, and convergence behavior. The results demonstrated the algorithm's effectiveness in producing connected, high-quality coverage solutions, with generally consistent performance. Several insights were gained regarding the roles of local search and parameter tuning, which contributed to improved outcomes. Although a few rare or context-specific limitations were observed, they do not diminish the algorithm's overall success. These findings confirm the viability of memetic approaches for wireless sensor network coverage and pave the way for further optimization and exploration.

Conclusion

In this study, we developed a hybrid **multi-objective** memetic algorithm that combines the stochastic exploration of genetic algorithms with precise local search techniques, aiming to improve coverage in wireless sensor networks. This problem is of great importance and complexity due to its constrained nature, particularly with regard to ensuring full coverage, minimizing the number of sensors, and maintaining network connectivity.

Extensive experiments were conducted to evaluate the algorithm's performance under various conditions, including different field densities, initial sensor counts, and distribution patterns. These experiments demonstrated that the algorithm was capable of providing stable and efficient solutions in most cases. Specifically, it achieved coverage rates exceeding 98% in more than 80% of the tested scenarios, while maintaining a limited number of sensors. Furthermore, the inclusion of local search significantly improved the solution quality following the genetic evolution phase, especially when selectively applied to the best individuals.

Key findings of the study:

- Enhanced overall field coverage without requiring excessive increases in the number of deployed sensors.
- Maintained connectivity among sensors in most scenarios without the need for external components or supplementary mechanisms.
- Achieved a balance between solution quality and execution time, with computation times remaining acceptable even for large-scale instances.

Beyond its implementation, this research **deepened our understanding of how multi-objective search strategies behave in constrained environments**, and how specific design decisions—such as encoding structure, weight balancing, or local improvement scope—can significantly affect convergence quality and runtime behavior. This experience also provided us with a clearer view of constraint-based modeling, and of the trade-offs between generality and realism in algorithmic design.

We learned that uncritical use of built-in Python functions can hinder code performance, and that overly simplified representations of the field or individuals may lead to unrealistic outcomes. We also recognized the importance of testing the algorithm on diverse cases with varying density and dimensions, as well as the necessity of documenting each development phase to track improvements and analyze their causes.

In light of these results, we believe that the work accomplished provides a solid foundation for developing more precise and effective approaches in the future. It also opens several potential avenues for extension, including:

- Adjusting the coverage radius of newly added sensors to better fill uncovered gaps, rather than keeping this parameter fixed throughout all stages.
- Guiding the mutation process based on coverage density data, allowing the algorithm to directly target under-covered areas instead of relying entirely on random generation.
- Developing a mechanism to remove redundant sensors in densely covered areas by analyzing the repetition of covered cells and selecting the most redundant sensor for removal, while ensuring the overall coverage rate remains unaffected.
- Expanding the algorithm's application to other domains beyond wireless sensor networks, such as aerial system coverage planning, resource distribution in ecological systems, or optimizing deployment strategies in cooperative robotic systems.

Despite the challenges we faced, this experience represents an encouraging starting point in the field of applied research. It lays the groundwork for a more mature phase in which our vision can expand and the knowledge gained can be applied in more complex and varied contexts. We hope that this work serves as a small yet effective contribution to improving wireless sensor network performance and as a first building block in a scientific journey we aspire to pursue further in the future.

References

[1] Fortinet, *Wireless network - definition*, https://www.fortinet.com/resources/cyberglossary/wireless-network, Accessed in March 2025, 2025.

[2] ScienceDirect, *Wireless sensor - engineering topics*, https://www.sciencedirect.com/topics/engineering/wireless-sensor, Accessed in March 2025, 2025.

[3] O. Bouachir and H. S. Hassanein, ``Edge computing for wireless sensor networks: An overview,'' *Internet of Things*, vol. 14, p. 100 377, 2021, Accessed in March 2025. doi: 10.1016/j.iot.2021.100377. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666351121000425.

[4] GeeksforGeeks, *Wireless sensor network (wsn)*, https://www.geeksforgeeks.org/wireless-sensor-network-wsn/, Accessed in March 2025, 2025.

[5] Blikai, *What are wireless sensor networks? all explained*, https://www.blikai.com/blog/wireless/what-are-wireless-sensor-networks-all-explained, Accessed in March 2025, 2025.

[6] K. W. Migliaccio, C. W. Fraisse, and M. D. Dukes, *Wireless sensor networks for agricultural applications*, https://edis.ifas.ufl.edu/publication/AE521, Accessed in March 2025, 2021.

[7] P. Verde, J. Díez-González, R. Ferrero-Guillén, A. Martínez-Gutiérrez, and H. Perez, ``Memetic chains for improving the local wireless sensor networks localization in urban scenarios,'' *Sensors*, vol. 21, no. 7, p. 2458, 2021, Accessed in March 2025, PDF contains 21 pages. doi: 10.3390/s21072458.

[8] C.-F. Huang and Y.-C. Tseng, ``The coverage problem in a wireless sensor network,'' *Mobile Networks and Applications*, vol. 10, no. 4, pp. 519–528, 2005. doi: 10.1007/s11036-005-1564-y.

[9] *Coverage optimization in wireless sensor networks using memetic algorithms*, https://example.com, Unpublished project report, 2025.

[10] ScienceDirect Topics, *Random deployment - an overview*, https://www.sciencedirect.com/topics/computer-science/random-deployment, Accessed in March 2025.

[11] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, ``A survey on sensor networks,'' *ACM Computing Surveys (CSUR)*, vol. 38, no. 4, pp. 1–72, 2002, Accessed in March 2025. doi: 10.1145/1143549.1143786. [Online]. Available: https://dl.acm.org/doi/10.1145/1143549.1143786.

[12] M. K. Ahmed and Z. K. Farej, ``Survey on clustering techniques for wireless sensor networks and cluster head selection model," *European Journal of Computer Science and Information Technology*, vol. 12, no. 2, pp. 1–21, 2024, Accessed in March 2025. doi: 10.37745/ejcsit.2013/vol12n1121. [Online]. Available: https://www.eajournals.org/.

[13] D. Ayat, *Problem classes p np*, YouTube, Accessed in February 2024, 2024. [Online]. Available: https://www.youtube.com/watch?v=2i5LUCuuGZM.

[14] W. contributors, *P, np, np-complete, and np-hard set diagram*, Wikimedia Commons, Diagram image only, accessed in February 2024, 2020. [Online]. Available: https://en.wikipedia.org/wiki/NP-hardness#/media/File:P_np_np-complete_np-hard.svg.

[15] Complexica, *Optimization algorithms in narrow ai glossary*, Complexica website, Accessed in February 2024, 2024. [Online]. Available: https://www.complexica.com/narrow-ai-glossary/optimization-algorithms.

[16] Stackademic, ``How to calculate big-o notation: Time complexity," *Stackademic Blog*, 2024, Accessed in February 2025. [Online]. Available: https://blog.stackademic.com/how-to-calculate-big-o-notation-time-complexity-5504bed8d292.

[17] P. Moscato, C. Cotta, and A. Mendes, *Memetic algorithms: A hybrid optimization approach*, Book Chapter in "Evolutionary Algorithms", Chapter 3, Pages 1--20, 2025. [Online]. Available: https://example.com/memetic-algorithms.

[18] R. contributor, *Flowchart of memetic algorithm*, English/Arabic, ResearchGate, Figure 2, Accessed in February 2025, 2011. [Online]. Available: https://www.researchgate.net/figure/The-flowchart-of-memetic-algorithm_fig2_287264078.

[19] C. Stoean and R. Stoean, ``Overview of evolutionary algorithms," in *Intelligent Systems Reference Library*, Accessed in February 2025, Springer, 2014, ch. 3. doi: 10.1007/978-3-319-06941-8_3. [Online]. Available: https://www.researchgate.net/publication/295105971_Overview_of_Evolutionary_Algorithms.

[20] GeeksforGeeks, *Local search algorithm in artificial intelligence*, https://www.geeksforgeeks.org/local-search-algorithm-in-artificial-intelligence/, Accessed in February 2025.

[21] A. Hemmak, *Genetic algorithms course*, Internal course material, University of M'Sila, Computer Science Department, Accessed in April 2025, contains 37 pages, 2024.

[22] Python Software Foundation, *Executive summary: The zen of python and philosophy*, https://www.python.org/doc/essays/blurb/, Accessed in May 2025.

[23] Anaconda, Inc., *Choosing between anaconda and python*, https://www.anaconda.com/topics/choosing-between-anaconda-vs-python, Accessed in May 2025.

[24] J. Grus, *Data Science from Scratch: First Principles with Python*. O'Reilly Media, 2015, Accessed in june, 2025. [Online]. Available: https://www.google.co.in/books/edition/_/BkaQkhkWGfoC?hl=en&gbpv=0.

[25] S. Aaronson, *Sphere packing*, https://mathenchant.wordpress.com/2019/09/16/sphere-packing/, Accessed in june , 2025, Sep. 2019.

[26] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd. Springer, 2008, Accessed in june, 2025, isbn: 978-3-540-77973-5. doi: 10.1007/978-3-540-77974-2. [Online]. Available: https://doi.org/10.1007/978-3-540-77974-2.

Appendix A: Calculation Details

This section presents additional calculations related to the developed methodology.

A.1 Packing Density (Packing Factor)

Packing density is a fundamental concept when estimating the required number of sensors to cover a specific area. A simple division of the total area by the coverage area of a single sensor yields a theoretical value that doesn't account for the necessary overlap between sensors. To address this, the packing factor (μ) is used as a correction factor for a more realistic estimation[24].

In the context of theoretical analysis and geometric packing, Square Packing is one method of arranging sensors, where each sensor is assumed to cover a circular area, and these circles are arranged in a square grid. Mathematically, a common value for the packing density in square packing is approximately **0.785**. This value is calculated by dividing the area of the circle (single sensor coverage) by the area of the square that encloses or surrounds it.

More specifically, if a sensor has a coverage radius r, its circular coverage area is πr^2. In square packing, sensors can be imagined as being arranged with their centers at the vertices of a square grid. The square "belonging" to each sensor (or within which its coverage lies) would have a side length of $2r$. Thus, the area of this square is $(2r)^2 = 4r^2$.

Therefore, the packing density μ for square packing can be calculated as follows:[25][26]

$$\mu = \frac{\text{Area of Circle}}{\text{Area of Square}} = \frac{\pi r^2}{(2r)^2} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

Numerically, this value is:

$$\mu \approx \frac{3.14159}{4} \approx 0.78539$$

This value means that approximately 78.5% of the square's area is effectively covered by the circle (sensor), indicating that there are uncovered areas between the circles in this square arrangement, or that overlap is required to achieve full coverage. Based on this, the estimated number of sensors required can be obtained by dividing the total network area by the single sensor's coverage area, then multiplying the result by $1/\mu$ (i.e., dividing by μ) to account for the packing efficiency.

A.2 Coverage Computation and Update

The coverage of a sensor field is computed based on the positions of deployed sensors using the Euclidean distance, as follows:

Algorithm 8 Compute Coverage from Scratch

Input: S_{ij}: Sensor placement matrix
r: Coverage radius
m, n: Width and height of the field
Output: C_{ij}: Coverage matrix

```
for each cell in S_ij do
    if S_ij == 1 then
        for x ← max(1, i − r) to min(m, i + r) do
            for y ← max(1, j − r) to min(n, j + r) do
                if √((i − x)² + (j − y)²) ≤ r then
                    C_xy ← C_xy + 1
return C_ij
```

To avoid recalculating the entire coverage map for small changes, we used a local update strategy that only touches the areas affected by sensor additions, removals, or movements. The strategy operates as follows:

Algorithm 9 Update Coverage after Local Change

Input: C_{ij}: Current coverage matrix
x, y: Coordinates of the affected sensor
r: Coverage radius
δ: +1 if added, -1 if removed
m, n: Width and height of the field
Output: C_{ij}: Updated coverage matrix

```
for i ← max(1, x − r) to min(m, x + r) do
    for j ← max(1, y − r) to min(n, y + r) do
        if √((i − x)² + (j − y)²) ≤ r then
            C_xy ← C_xy + δ
return C_ij
```

A.3 Smart Sensors Addition

To enhance the initial random placement of sensors and ensure better coverage, a **BFS-based** strategy is used. The smart_add_sensors function scans the field to identify large uncovered regions, then selects the optimal position within each region for placing an additional sensor.

This process continues iteratively until no significant uncovered area remains. The following pseudocode describes the mechanism of this function:

Algorithm 10 smart_add_sensors($solution, coverage$)

Input: $solution$: binary matrix of sensor positions, $coverage$: current coverage matrix
Output: Updated $solution$ and $coverage$ with additional sensors placed

```
area_threshold ← ⌈π·radius² / 4⌉
while True do
    visited ← matrix of size (height × width) initialized to False  added ← False
    for y ← 0 to height − 1 do
        for x ← 0 to width − 1 do
            if not visited[y][x] and coverage[y][x] = 0 then
                region ← bfs_region(x, y, coverage, visited)
                if |region| > area_threshold then
                    best ← get_best_sensor_position(region, coverage)
                    if best ≠ None then
                        (bx, by) ← best  solution[by][bx] ← 1
                        move_sensor_update_coverage(bx, by, coverage)  added ← True
    if not added then
        break
```

A.4 Identifying Isolated Groups of Sensors

When addressing the coverage problem in a sensor network, it is essential to ensure that the deployment solution forms a connected network. This means that all existing sensors must belong to a single connected group or be indirectly connected through other sensors. The existence of isolated groups of sensors can lead to uncovered regions or result in an inefficient data transmission network.

To determine whether isolated groups are present, we use the principle of *Disjoint Set Union (DSU)*, an efficient algorithm for tracking connected components. Each sensor is treated as a *node*, and a *connection* is established between two sensors if they lie within a specific communication distance (in this case, twice the coverage radius, $2r$, to ensure their ability to communicate or overlap). If the number of disjoint sets (roots) is greater than one at the end of the process, this indicates the presence of isolated groups, meaning the network is not fully connected.

Algorithm 11 HasIsolatedGroups($solution$)

Input: $solution$: a 2D matrix indicating sensor positions (1 for sensor, 0 otherwise)
Output: **True** if isolated groups exist, **False** otherwise

$sensor\_positions \leftarrow$ list of all coordinates (i, j) where $solution[i][j] = 1$
if $sensor\_positions$ *is empty* **then**
    return False
Initialize $parent[pos] \leftarrow pos$ **for each** $pos \in sensor\_positions$

foreach (pos_1, pos_2) *pair in* $sensor\_positions$ **do**
    if *EuclideanDistance*$(pos_1, pos_2)^2 \leq (2r)^2$ **then**
        Union$(pos_1, pos_2, parent)$
$roots \leftarrow$ set of Find$(pos, parent)$ for all $pos \in sensor\_positions$
return True if $|roots| > 1$, otherwise **False**

Appendix B: List of Acronyms

- **WSN**: Wireless Sensor Network
- **WSNs**: Wireless Sensor Networks
- **MA**: Memetic Algorithm
- **GA**: Genetic Algorithm
- **HCA**: hill climbing Algorithm
- **LS**: Local Search
- **BFS**: Breadth-First Search
- **DFS**: Depth-First Search