Algerian People's Democratic Republic
Ministry of Higher Education and Scientific Research
University of Ain Temouchent - Belhadj Bouchaib Faculty of Science and Technology

Faculty of Science and Technology
Department of Mathematics and Computer Science

Master Thesis
For the Master's Degree in Computer Science
Option: Networks and Data Engineering (NDE)

Thesis title:

Enhancing Variable Neighborhood Search (VNS) Performance for Relay Node Deployment through Machine Learning

Presented by:
M. Ahmed Nour ABDESSELAM
Defended on: June 24, 2024

In front of the jury composed of:

President: Dr. Djamila BOUHALOUAN	U.B.B.A.T
Examiner: Dr. Jalel MERAD BOUDIA	U.B.B.A.T
Supervisor: Dr. Ali BENZERBADJ	U.B.B.A.T
Co-Supervisor: Dr. Piroška Stanić Molcer	VTŠ, Serbia

Dedication

I dedicate this humble work:
To the woman who endured without letting me suffer, who spared no effort to make me happy: my beloved mother. To the man deserving of my respect, who has always been there for me: my dear father. I hope they will find in this work all my gratitude. I hope that one day, I could repay them for all they have done for me. May God grant them happiness and a long life.

To my dear sisters Abir and Samah, to my younger brother Sami, to my maternal uncle, to all my teachers, and to all my friends. Thank you for your love and encouragement.

Ahmed Nour ABDESSELAM

Acknowledgements

First and foremost, I am profoundly thankful to my supervisor, Dr. Ali BENZERBADJ, for his unwavering support, invaluable guidance, and constructive feedback throughout every stage of this research. His expertise and dedication have been instrumental in shaping the direction of this work.

I am also indebted to Dr. Piroška Stanić Molcer, for her insightful comments, encouragement, and willingness to share her expertise in Machine Learning.

I would also like to extend my heartfelt gratitude to the staff and professors at Subotica Tech College of Applied Sciences (VTŠ) for their invaluable assistance and support throughout my work. Their expertise and encouragement have played a significant role in the successful completion of this thesis.

My sincere appreciation goes to the members of my thesis committee, namely Dr. D. BOUHALOUAN and Dr. J. MERAD BOUDIA, for their valuable feedback and suggestions, which have significantly enriched the quality of this study.

I wish to express my heartfelt thanks to my family for their unwavering love, encouragement, and understanding throughout this journey. Their support has been a constant source of strength and motivation.

Last but not least, I would like to thank all my friends and colleagues who have offered their encouragement, assistance, and moral support during this challenging yet rewarding endeavor.

In conclusion, I am deeply appreciative of everyone mentioned above and others who have contributed in various ways to the completion of this thesis. Your support and guidance have been invaluable, and I am truly grateful for the opportunity to undertake this research.

Ahmed Nour ABDESSELAM

Contents

List of Figures

List of Tables

List of Algorithms

General Introduction

Nowadays, monitoring sensitive areas using Wireless Sensor Networks (WSNs) is considered a highly promising research field. The miniature nature of sensor nodes, which makes them easily concealable, has led to the extensive use of these networks for securing strategic sites.

In this work, we employ a WSN to monitor a fenced sensitive site, such as an oil or nuclear facility or any other critical building (governmental or otherwise). Our monitoring approach involves deploying sensor nodes along the perimeter of the site. These nodes, called Sentinel Nodes (SSNs), are responsible for generating an alert whenever they detect an intrusion. These alerts must then be relayed to the sink node using Relay Nodes (RNs). The sink node is responsible for transmitting the alerts to a decision center via a high-speed connection (internet, cellular, or satellite).

On one hand, the miniature aspect of the nodes means they have scarce resources in terms of energy, processing power, storage, communication range, and data rate. On the other hand, the critical nature of the surveillance application discussed in this work requires that alerts be delivered in a timely manner to ensure the security of the monitored site. Therefore, routing the alerts to the sink node must involve the minimal number of hops to conserve energy and reduce end-to-end latency in the WSN.

The proposed multi-objective optimization solution is a near-optimal deterministic deployment of the WSN nodes. This deployment should be done at minimal cost (minimal number of deployed RNs) and minimal Network Diameter (ND), under the following constraints: full coverage of the site's borders using SSNs and ensure connectivity (at least one path from each SSN to the sink node).

This thesis focuses on enhancing the performance of Variable Neighborhood Search (VNS) for relay node deployment in WSNs through the application of machine learning techniques. Specifically, it explores how integrating Reinforcement Learning (RL) with combinatorial optimization can lead to intelligent systems capa-

ble of learning from experience and adapting to new situations, while still providing optimal or near-optimal solutions. The proposed solution has been compared to Basic VNS (BVNS). The experimental results are very encouraging.

The structure of the thesis is as follows:
Chapter I introduces WSNs, detailing their components, architecture, and the challenges and constraints influencing their design. The subsequent chapter II delve into the combination of meta heuristics and machine learning problem for multi objective optimization problems. Finally, the chapter III of the thesis presents a novel approach integrating meta-heuristic algorithms with reinforcement learning techniques to solve the deployment problem in WSNs, aiming to optimize the placement of sensor and relay nodes for improved network efficiency.

Chapter I

Overview of Wireless Sensor Networks

I.1 Introduction

Wireless Sensor Networks (WSNs) have emerged as a transformative technology, revolutionizing the way we collect, process, and disseminate data in various domains.

WSNs are composed of a large number of small, low-power, and low-cost sensor nodes that communicate wirelessly to monitor and collect data from the environment. These nodes, composed of sensors, processors and transceivers, obtain information on the environment such as temperature, pressure, humidity or pollutant, and send this information to a base station. The latter sends the info to a wired network or activates an alarm or an action, depending on the type of data monitored.

The deployment of sensor nodes is challenging as it significantly impacts energy, a scarce resource in WSNs. Finding the optimal deployment is crucial since improper placement can greatly affect energy efficiency, potentially reducing the overall effectiveness and longevity of the network. Energy efficiency is critical since sensor nodes often rely on limited energy sources, such as batteries. Ensuring the longevity and effectiveness of WSNs requires careful consideration of energy matters.

The applications of WSNs are vast and diverse, ranging from environmental monitoring, structural health monitoring, and disaster management to smart homes, agriculture, and healthcare, or tracing human and animal movement in forests and borders. WSNs have the potential to revolutionize living standards, safety, and the environment while being more economically beneficial.

This chapter will provide an overview of WSNs including their architecture, types, applications, constraints and challenges, energy harvesting sources, and their deployment.

I.2 Components of Wireless Sensor Networks

WSNs consist of vital components that collaborate to facilitate sensing, data gathering, and communication across diverse applications. Familiarizing oneself with these components is crucial for comprehending the complexities involved in designing, deploying, and operating WSNs. This section explores the essential elements that form the foundation of a standard WSN architecture [26].

I.2.1 Sensor nodes

A sensor node is composed of four basic components: a sensing unit, a processing unit, and a transceiver unit, as shown in Figure I.1. It may also include additional components depending on the application, such as a location-finding system, a power generator, and a mobilizer. In our project, we have deployed two types of nodes: sentinel sensor nodes and relay nodes. Sentinel sensor nodes are equipped with sensing units to monitor the perimeter, while relay nodes primarily serve to forward data between sensor nodes and the central processing unit, ensuring robust communication throughout the network.

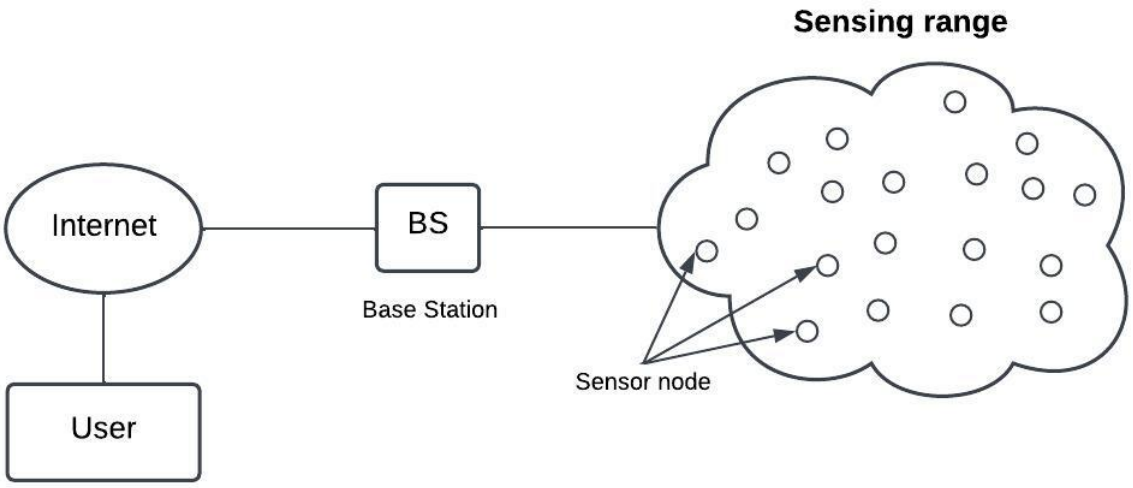


Figure I.1: A basic working model of WSN

I.2.1.1 Sensing unit

Sensing units are usually comprised of two sub-units as shown in Figure I.2: sensors and analogue to digital converter (ADC) which is responsible on converting the

analogue signals produced by the sensors.

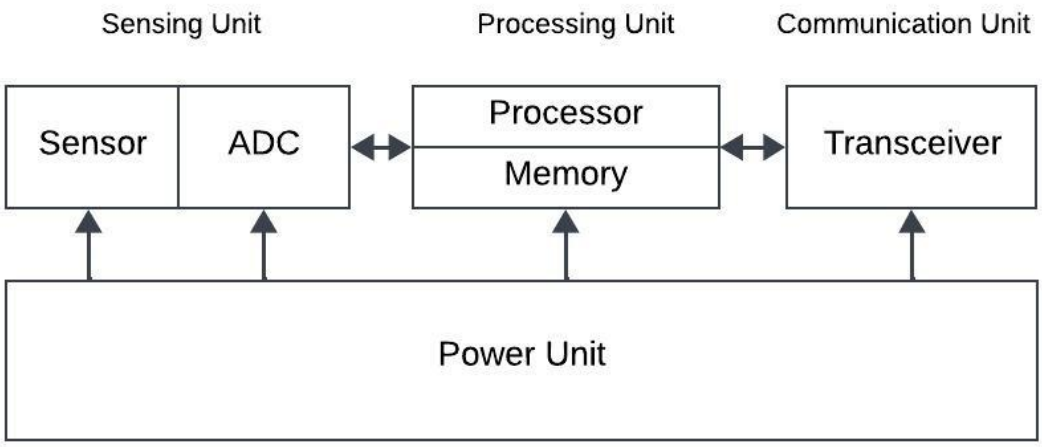


Figure I.2: The components of a sensor node

I.2.1.2 Processing unit

typically in the form of a micro-controller or microprocessor, serves as the computational brain of the sensor node. It executes algorithms for data processing, decision-making, and communication protocols, often optimized for energy efficiency due to the resource-constrained nature of WSNs, the processing unit is usually associated with a small storage unit as shown in the Figure I.2.

I.2.1.3 Communication unit

Sensor nodes are equipped with wireless communication interfaces. These interfaces may support various communication standards such as Zigbee, Bluetooth Low Energy (BLE), Wi-Fi, or proprietary protocols, depending on factors like data rate, range, and power consumption requirements.

Each sensor node has a communication range (Rc) and a sensing range (Rs). Figure I.3 shows the areas defined by these two spans. Communication range (Rc) is the range in which the sensor node can communicate with other nodes. As for the Sensing Range (Rs), it is the range in which the sensor node can detect the event. Usually in WSNs applications,the Communication range is larger than the Sensing range [45].

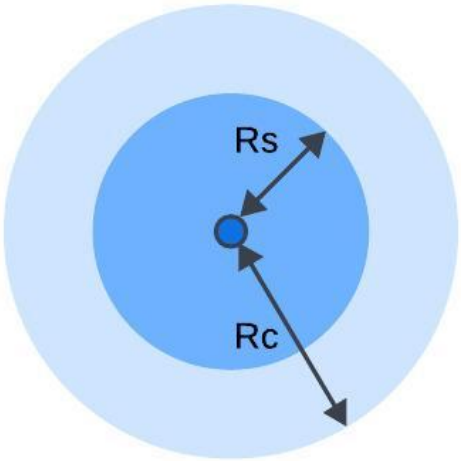


Figure I.3: Communication range and Sensing range

I.2.1.4 Mobilizer (optional)

The mobilizer component of a sensor node, which includes actuators or motors, is designed to enable physical movement or positional adjustments of the node. This capability is particularly vital in applications where the network's configuration needs to be dynamically restructured. For instance, in a surveillance system, mobile sensor nodes can reposition themselves to enhance coverage or avoid obstacles, thereby ensuring optimal monitoring of the area. The ability to move allows the network to adapt to changing environmental conditions and operational requirements, significantly improving the flexibility and robustness of the sensor network [18].

I.2.1.5 Power Management Unit

The power management unit of a sensor node plays a crucial role in maintaining efficient energy usage and prolonging the node's operational life. This unit typically incorporates power conditioning circuits, responsible for regulating voltage and managing power distribution within the node. By ensuring stable power supply and suitable voltage for the node's components, these circuits prevent damage and enhance performance. Additionally, the power management unit employs sleep/wake scheduling techniques to conserve energy. By transitioning the node into a low-power sleep mode when not actively engaged in sensing or communication tasks, these techniques significantly reduce energy consumption, thereby extending battery life and overall operational period of the sensor node [18].

Furthermore, energy harvesting techniques can be considered a viable solution

for power management. These techniques can provide a continuous power supply, reducing the dependency on batteries and further prolonging the operational life of the sensor node. Integrating energy harvesting with power management not only enhances sustainability but also ensures uninterrupted operation in remote or inaccessible locations.

Energy harvesting techniques consist of obtaining energy from external resources such as solar, wind, and heat, involves converting this energy into electrical energy, which can be either stored or used immediately. They directly extract energy from the environment, offering an alternative power source in situations where connecting devices to a power grid is undesirable or impractical due to cost or logistical constraints. The components utilized in energy harvesting systems may vary depending on the energy source.

I.2.1.5.1 Energy Harvesting sources

In [30, 11], different types of energy that can be harvested are discussed. Figure I.4 depict some of the energy sources that can be used. A brief overview for some of the mentioned energy harvesting sources and energy harvesters are listed below:

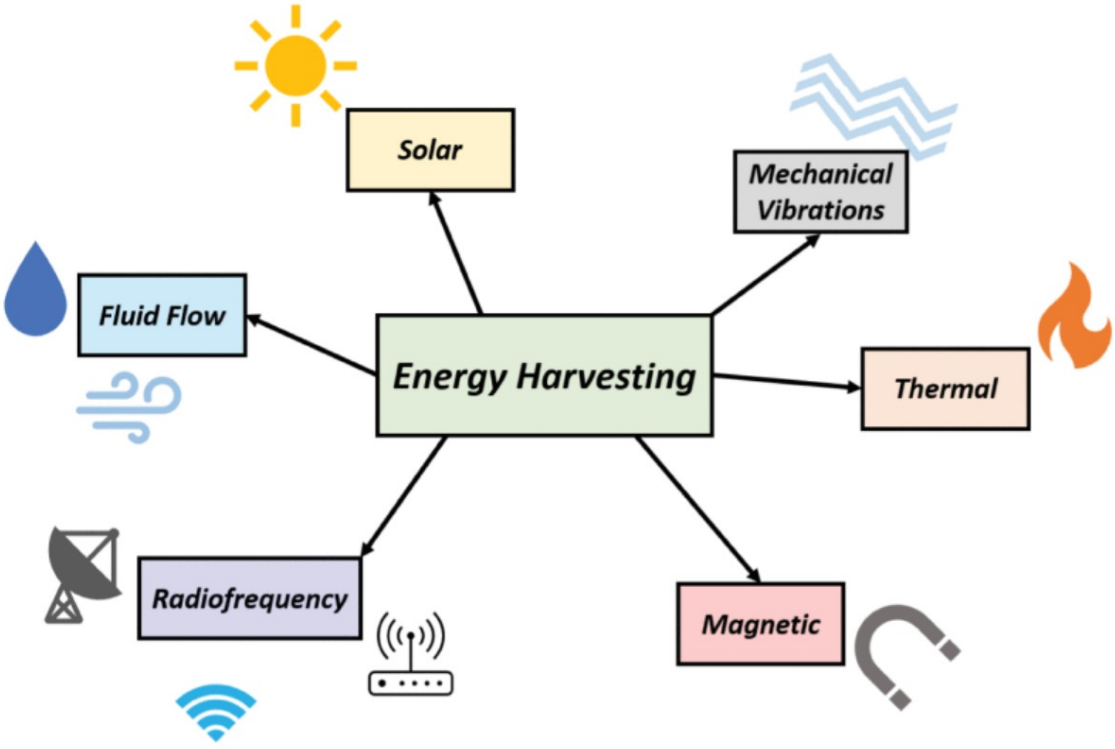


Figure I.4: Sources of energy that can be used [74].

Solar Energy Harvesting Solar energy harvesting is one of the most common energy harvesting techniques utilized in WSNs. It is a sustainable, clean and safe method of harvesting energy. The solar energy harvesting industry is growing rapidly and the costs for PV-cells have dropped significantly over the last 30 years while the availability has increased. Even though the technology has come a long way, there are still challenges to overcome in terms of maximizing the energy output from the solar irradiance [55].

Thermal Energy Harvesting Thermoelectric energy harvesting utilizes thermoelectric power generators (TEGs), composed of a thermopile made of two different conductors, typically semiconductors, forming a thermocouple. This setup is placed between hot and cold plates, creating a temperature difference. According to the Seebeck effect, this temperature disparity induces an electrical voltage between the junctions of the thermocouple, generating heat flow through the generator. This process converts thermal energy into electrical power, which is proportional to the temperature difference. As long as the temperature gradient persists, continuous energy harvesting occurs.

Mechanical Energy Harvesting Describes the process of converting mechanical energy into electricity through various means such as vibrations, mechanical stress, pressure, strain on the sensor's surface, high-pressure motors, waste rotational movements, fluids, and force. The principle behind mechanical energy harvesting involves converting the energy from the displacements and oscillations of a spring-mounted mass component inside the harvester into electrical energy. Mechanical energy harvesting can be categorized into three types: *piezoelectric, electrostatic*, and *electromagnetic* [11].

Wind Energy Harvesting Wind energy harvesting involves converting airflow (e.g., wind) energy into electrical energy. This is achieved by using a properly sized wind turbine to harness the linear motion of the wind and generate electrical energy. There are miniature wind turbines capable of producing sufficient energy to power WSN nodes [11].

I.2.2 Base Station (Sink)

Sink nodes in WSNs as shown in Figure I.5 serve as centralized points where data gathered from multiple sensor nodes are aggregated before being transmitted to external systems. These specialized nodes have higher energy reserves and computational capacity compared to regular sensor nodes. They play a critical role in minimizing propagation latency and conserving overall network energy usage.

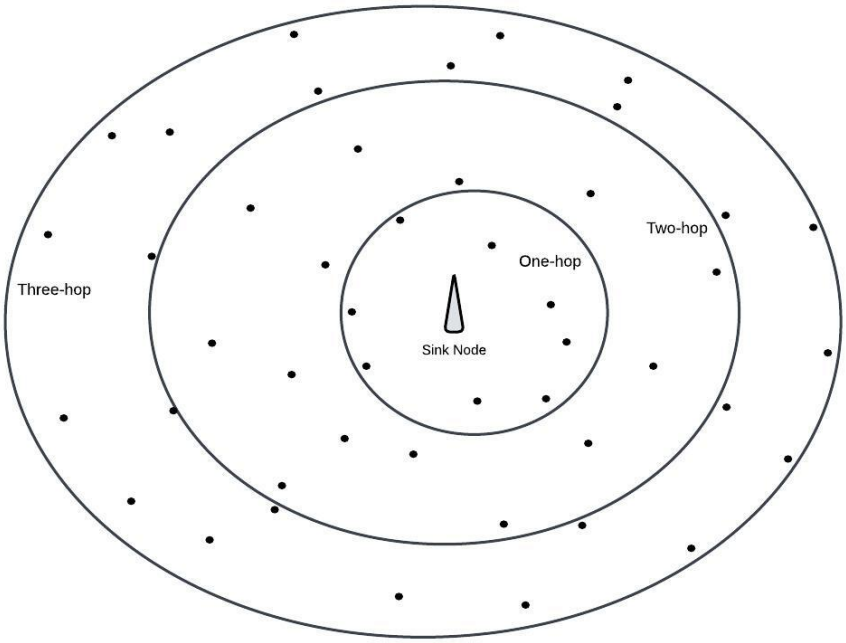


Figure I.5: Sink Node

The optimal placement of sink nodes within a WSN is a significant consideration because it directly impacts the efficiency and performance of the entire system. Researchers often explore strategies to determine the best positions for sink nodes, taking into account factors like distance from sensor nodes, energy conservation, and network coverage. Additionally, there exist various types of sink nodes, including fixed (static) and mobile sinks as illustrated in Figure I.6, each offering unique benefits depending on specific application requirements [45].

I.3 Types of Wireless Sensor Networks

There are several types of WSNs as depicted in Figure I.7, generally differing based on the application domain. Depending on their deployment environment, they face various constraints and challenges [59].

I.3.1 Terrestrial WSN

Terrestrial Wireless Sensor Networks typically comprise a large number of low-cost nodes, ranging from hundreds to thousands, deployed on land within a specified area, often in an ad-hoc manner (e.g., nodes dropped from an airplane). In these

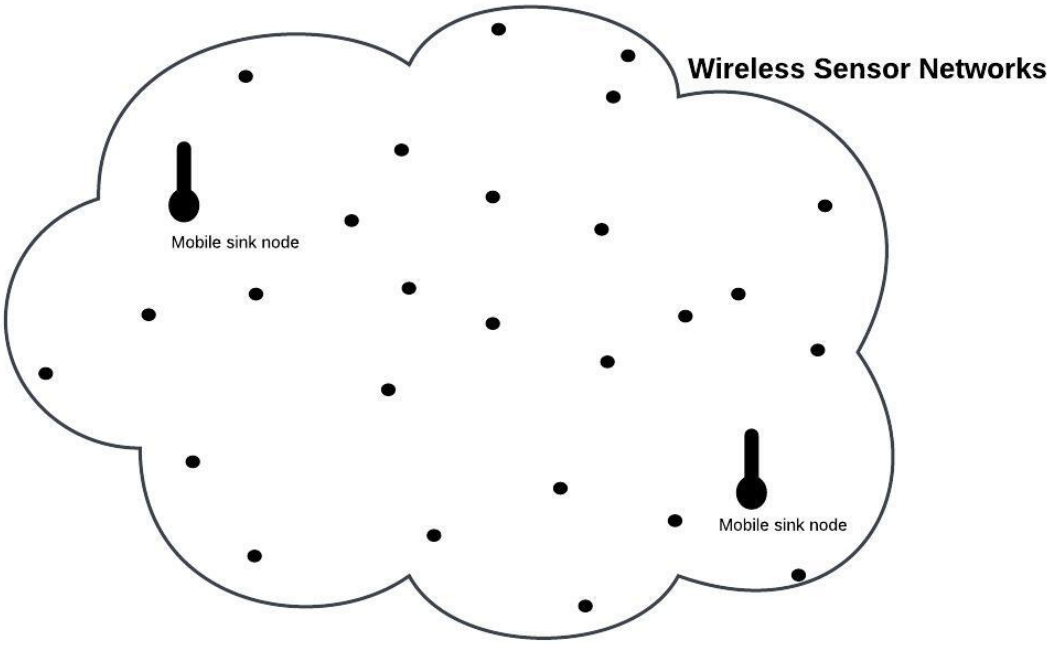


Figure I.6: Sensor nodes with mobile sink node architecture

networks, sensor nodes must efficiently transmit data back to a base station, especially in dense environments. Due to limited and usually non-rechargeable battery power, terrestrial sensor nodes may utilize secondary power sources like solar cells to prolong their operation. Energy conservation strategies include multi-hop optimal routing, limited transmission range, in-network data aggregation, and low duty-cycle operations. Terrestrial WSNs find applications in environmental sensing and monitoring, industrial monitoring, and surface explorations [58].

I.3.2 Underground WSN

Underground WSNs consist of sensor nodes deployed in caves, mines, or underground areas to monitor subterranean conditions. To transmit information from these underground nodes to the base station, additional sink nodes are positioned above ground. These networks are more expensive than terrestrial WSNs because they require specialized equipment to ensure reliable communication through soil, rocks, and water. Wireless communication poses a challenge in such environments due to high attenuation and signal loss. Additionally, it is challenging to recharge

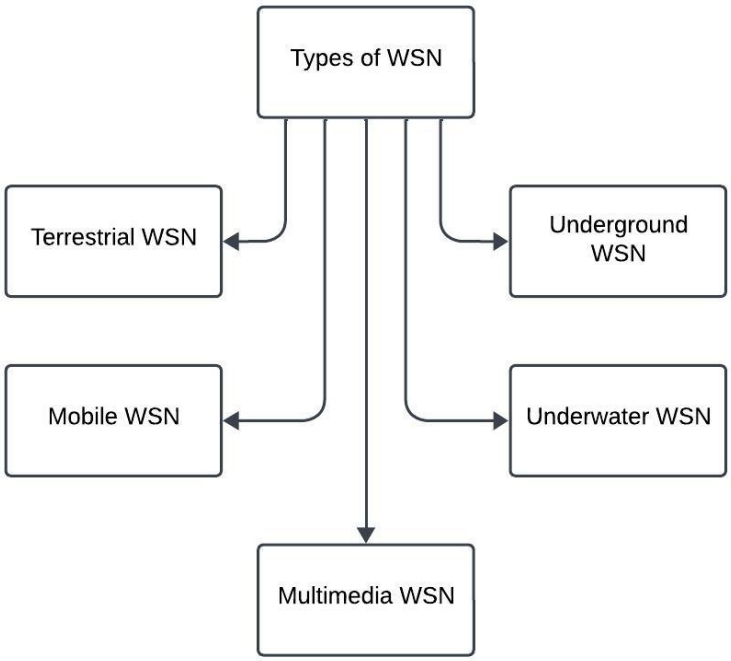


Figure I.7: Types of WSNs

or replace the batteries of nodes buried underground, highlighting the importance of designing energy-efficient communication protocols to prolong their lifespan. Underground WSNs find applications in agriculture monitoring, landscape management, underground monitoring of soil, water, or minerals, and military border monitoring [58].

I.3.3 Underwater WSN

Underwater WSNs are comprised of sensors deployed underwater, typically in ocean environments. Due to their high cost, only a limited number of nodes are deployed, and autonomous underwater vehicles are utilized to explore or gather data from them. Underwater wireless communication relies on acoustic waves, which pose several challenges such as limited bandwidth, long propagation delay, high latency, and signal fading. These nodes must self-configure and adapt to the extreme conditions of the ocean environment. Equipped with limited, non-rechargeable batteries, energy-efficient underwater communication and networking techniques are essential. Applications of underwater WSNs include pollution monitoring, undersea surveillance and exploration, disaster prevention and monitoring, seismic monitoring, equipment monitoring, and underwater robotics [58].

I.3.4 Multi-media WSN

The setup involves low-cost sensor nodes equipped with cameras and microphones, deployed according to a pre-planned strategy to ensure comprehensive coverage. These multimedia sensor devices are capable of storing, processing, and retrieving various types of multimedia data, including video, audio, and images. They must address numerous challenges such as high bandwidth requirements, significant energy consumption, quality of service (QoS) provision, data processing and compression techniques, and cross-layer design.

Developing transmission techniques that support high bandwidth while maintaining low energy consumption is essential for delivering multimedia content such as video streams. Although QoS provision is challenging in multimedia WSNs due to variable link capacity and delay, achieving a certain level of QoS is crucial for reliable content delivery. Multimedia WSNs enhance existing WSN applications such as tracking and monitoring by providing richer data streams for analysis and decision-making [58].

I.3.5 Mobile WSN

Mobile WSNs are composed of sensor nodes that have mobility capabilities, enabling them to move around and interact with the physical environment. These mobile nodes can reposition and organize themselves within the network while also sensing, computing, and communicating. Unlike static WSNs, which use fixed routing, mobile WSNs require dynamic routing algorithms to accommodate the mobility of nodes.

Mobile WSNs encounter various challenges, including deployment, mobility management, localization with mobility, navigation and control of mobile nodes, maintaining adequate sensing coverage, minimizing energy consumption during movement, ensuring network connectivity, and managing data distribution.

Primary applications of mobile WSNs include environmental and habitat monitoring, underwater surveillance, military surveillance, target tracking, and search and rescue operations. Compared to static nodes, mobile sensor nodes can achieve higher degrees of coverage and connectivity due to their ability to move within the environment [58].

I.4 Applications of Wireless Sensor Networks

WSN has a wide range of applications ranging from home automation to commercial industrial applications [26]. Some of the important applications of WSN are given in Figure I.8:

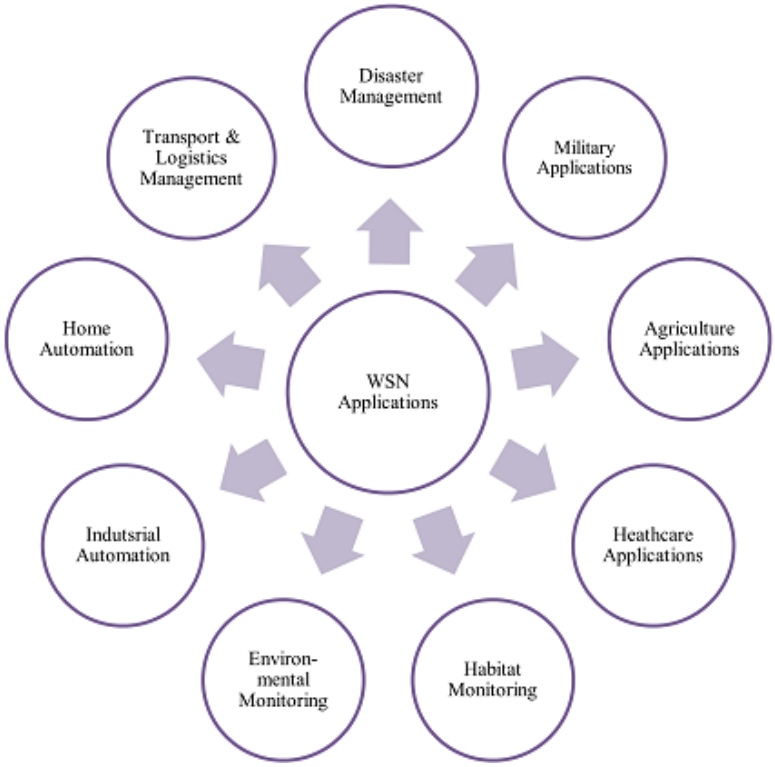


Figure I.8: WSNs applications [54].

I.4.1 Military applications

The military area is not always the simplest the first field of human pastime that used WSNs, however, it is also considered to have the initiation of sensor network research. The main subcategories of the military applications of WSNs are battle-field surveillance, combat monitoring and intruder detection in fenced areas such as prisons and military barracks.

I.4.2 Health applications

In healthcare, WSNs play a crucial role in remote patient monitoring, assisted living systems as shown in Figure I.9. WSN-enabled devices can continuously monitor vital signs, detect falls, and provide timely alerts in case of emergencies, improving patient care and safety.

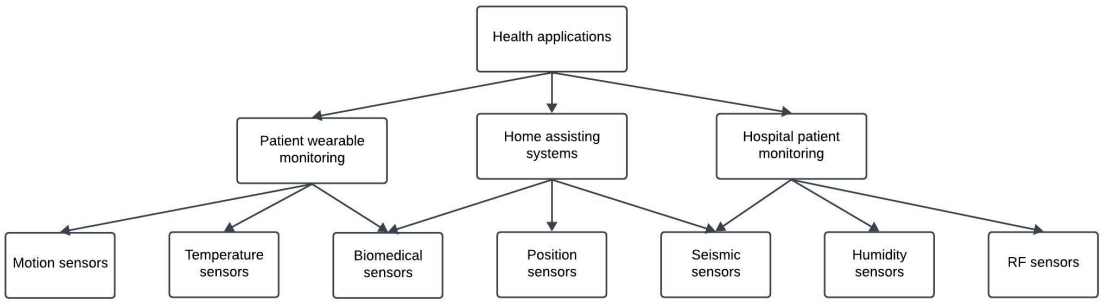


Figure I.9: Subcategories of WSNs based Health Applications and the types of sensors used

I.4.3 Environmental applications

WSNs are extensively used for environmental monitoring in areas such as agriculture, forestry, and wildlife conservation. They can monitor parameters like temperature, humidity, soil moisture, and air quality, enabling real-time data collection for research and decision-making.

I.4.4 Urban applications

WSNs can be used to solve the various urban problems, for example, coordination of the specialised vehicles like ambulance, fire tenders, rescue vehicles, police automobiles, logistics of public transportation, traffic management, monitoring chemical/physical environmental parameters, building security and many other.

I.4.5 Smart Cities

WSNs play a vital role in transforming traditional cities into intelligent, connected, and sustainable "smart cities" through continuous monitoring and analysis of various facets of urban living. These technologies enable smart cities to optimize resources, reduce costs, and enhance the overall well being of citizens.

I.4.6 Industrial Automation

WSNs are deployed in industrial environments for condition monitoring, predictive maintenance, and process optimization. They enable real-time monitoring of equipment health, temperature, pressure, and other parameters, helping industries enhance operational efficiency and reduce downtime. Figure I.10 shows the industrial applications of WSNs and the types of the sensors used by them.

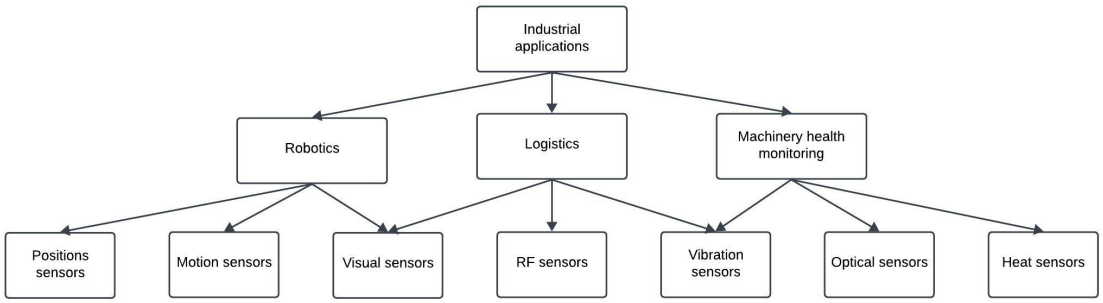


Figure I.10: WSNs based Industrial Applications and the types of sensors used by them.

I.5 Routing Structure in Wireless Sensor Networks

The sensor nodes are usually scattered in the field as shown in Figure I.1. These sensor has the capabilities to collect the data and send it back to the sink and the end user. The sink or the end user may communicate with the decision making unit node using Internet or Satellite. The routing protocols define how the nodes will communicate with each other and how the information will be scattered through the network. There are many ways to classify the routing protocols of WSN. The basic classification of routing protocols is illustrated in Figure I.11.

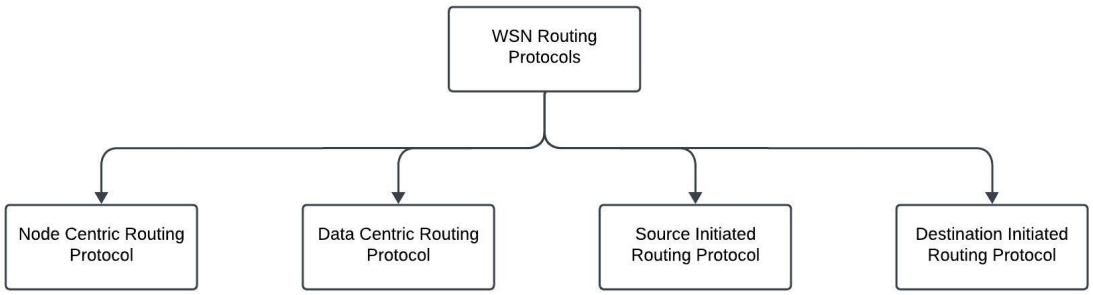


Figure I.11: Basic classification of routing protocols

I.5.1 Node centric routing protocols

Node centric protocols use numeric identifiers to specify the destination node, which is not the expected type of communication in Wireless Sensor Networks (WSNs). An example of such a protocol is Low Energy Adaptive Clustering Hierarchy (LEACH) [63].

Low Energy Adaptive Clustering Hierarchy (LEACH): LEACH is a routing protocol that aims to distribute energy equally among all sensor nodes in the network. It creates several clusters of sensor nodes, with one node designated as the cluster head that acts as the routing node for all other nodes in the cluster.

Unlike other routing protocols where the cluster head is selected before communication starts, LEACH applies randomization to select the cluster head from a group of nodes. This temporary selection of the cluster head from several nodes makes the protocol more long-lasting as the battery of a single node is not burdened for long. Sensor nodes elect themselves as the cluster head based on a probability criteria defined by the protocol. However, if there is any problem with the cluster head, the communication fails, and there is a higher chance that the battery dies earlier than the other nodes in the cluster.

I.5.2 Data Centric routing protocols

In wireless sensor networks, the data or information sensed is often more valuable than the nodes themselves. Data-centric routing techniques focus on transmitting information based on specific attributes rather than collecting data from particular nodes. In these techniques, the sink node queries specific regions to gather data with specific characteristics, requiring a naming scheme based on attributes to describe the data features. An example of a data-centric routing protocol is Sensor Protocols for Information via Negotiation (SPIN) [63].

Sensor protocols for information via negotiation (SPIN) SPIN aims to address deficiencies like flooding and gossiping present in other protocols by emphasizing the sharing of metadata (descriptors about the data) rather than the actual data sensed by the node. As shown in Figure I.12 the protocol involves three messages: ADV, REQ, and DATA. Nodes broadcast an ADV packet to announce they have data, including attributes of the data. Nodes interested in this data send REQ messages, and upon receiving them, the advertising node sends the data. This process continues as nodes share data based on interest, forming a network model that promotes efficient resource usage.

I.5.3 Source Initiated Routing Protocols

Source-initiated protocols in WSNs are those where the path setup generation originates from the source node. In contrast, destination-initiated protocols are those where the path setup generation originates from the destination node. In most WSNs, the sensed data or information is more valuable than the actual node itself. Therefore, data-centric routing techniques focus on transmitting information based

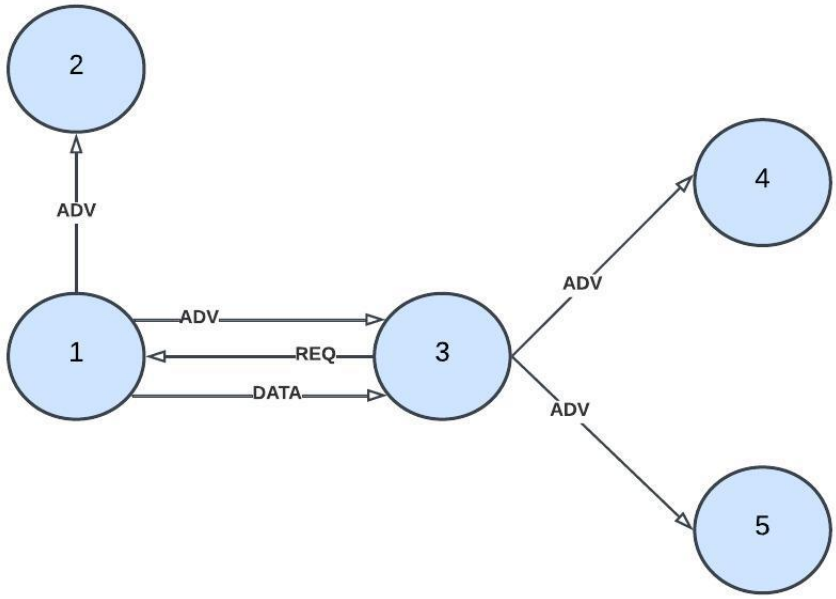


Figure I.12: SPIN Routing Protocol

on specific attributes rather than collecting data from particular nodes. In data-centric routing, the sink node queries specific regions to gather data with specific characteristics, requiring a naming scheme based on attributes to describe the data features [63].

I.5.4 Destination Initiated Routing Protocols

Destination-initiated protocols in WSNs are those where the path setup generation originates from the destination node. An example of a destination-initiated protocol is Directed Diffusion (DD) [63].

Directed Diffusion (DD) Directed diffusion is a data-centric routing technique that focuses on information gathering and disseminating it efficiently. This routing protocol is energy-efficient, leading to an increased network lifetime. In Directed Diffusion, all communication occurs node-to-node, eliminating the need for addressing within the protocol. Directed Diffusion involves a two-phase process where interest messages are initially flooded to find sources, and sources reply with exploratory data messages to establish paths towards the sink node. The subsequent transmission of data along these reinforced paths constitutes the second phase of the protocol. To address issues like energy imbalance and network partitioning due

to hotspot formation, an extension called Source Routing Directed Diffusion (SR-DD) selects paths based on residual energy in nodes, ensuring more uniform energy utilization and avoiding hotspots.

I.6 WSNs: Constraints and Challenges

Ideally in the deployment of sensor nodes is that the deployed network can respond to all the design constraints that guarantee a long lifespan of the network. In literature, most deployment methods focus on specific criteria such as as connectivity, coverage, overlap, number of nodes and consumption of energy [26, 6]

I.6.1 Constraints

I.6.1.1 Coverage

Among the significant factors in deploying a sensor network, coverage stands out as a key metric of Quality of Service (QoS). Coverage can be either complete or partial, depending on the application's requirements. It can be achieved through single-node monitoring as shown in Figure I.13 (referred to as 1-coverage) or by multiple sensor nodes as shown in Figure I.14 (known as k-coverage). In the realm of WSNs, coverage is typically categorized into three types: area coverage, point (target) coverage, and barrier coverage.

I.6.1.1.1 Zone coverage

Surface coverage, also known as area coverage, is a crucial aspect in the deployment of sensor networks. Its primary goal is to monitor a specific geographical zone, termed the area of interest, ensuring that every point within this region is effectively covered by a subset of sensor nodes as depicted in Figure I.15. The choice between total or partial coverage depends on the specific needs and objectives of the application being deployed. Achieving optimal surface coverage is essential for maximizing the efficiency and effectiveness of the sensor network in capturing and transmitting relevant data from the monitored area.

I.6.1.1.2 Target Coverage

Target coverage, also known as point coverage, is a type of coverage used to monitor specific points of interest within a capture field whose geographic position is known. Each specific point must be covered by at least one sensor node. Examples of monitoring points of interest include military applications such as monitoring

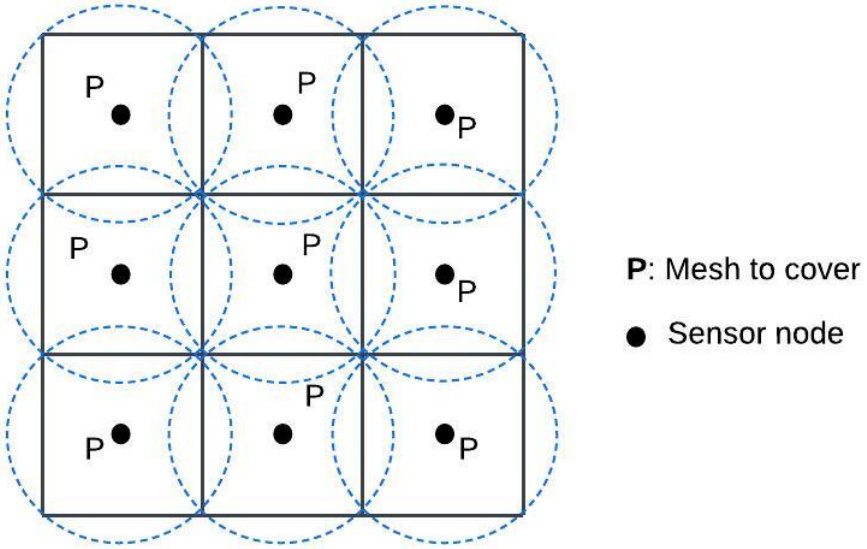


Figure I.13: 1-Coverage

enemy bases. Target coverage is particularly useful when the focus is on detecting and tracking specific objects or events rather than simply observing a general area.

I.6.1.1.3 Border Coverage

Border coverage is a strategy used to cover a specific portion of the area of interest. Sensor nodes are not tasked with monitoring events within the considered zone; rather, the focus is solely on covering the perimeter of this region to detect intruders attempting to breach it. This type of coverage is recognized as a suitable model for applications like monitoring international borders or detecting the spread of hazardous chemicals around a facility, for instance.

I.6.1.2 Connectivity

Two sensor nodes are considered connected if and only if they can communicate directly (single-hop connectivity) or indirectly (multi-hop connectivity). In WSNs, the network is deemed connected if there exists at least one path between the sink and every sensor node within the designated area. Considering connectivity is essential for effectively monitoring a given region. Merely ensuring coverage is not sufficient; sensor nodes must be capable of immediately reporting any detected events to the sink. There are two types of network connectivity: complete connectivity and in-

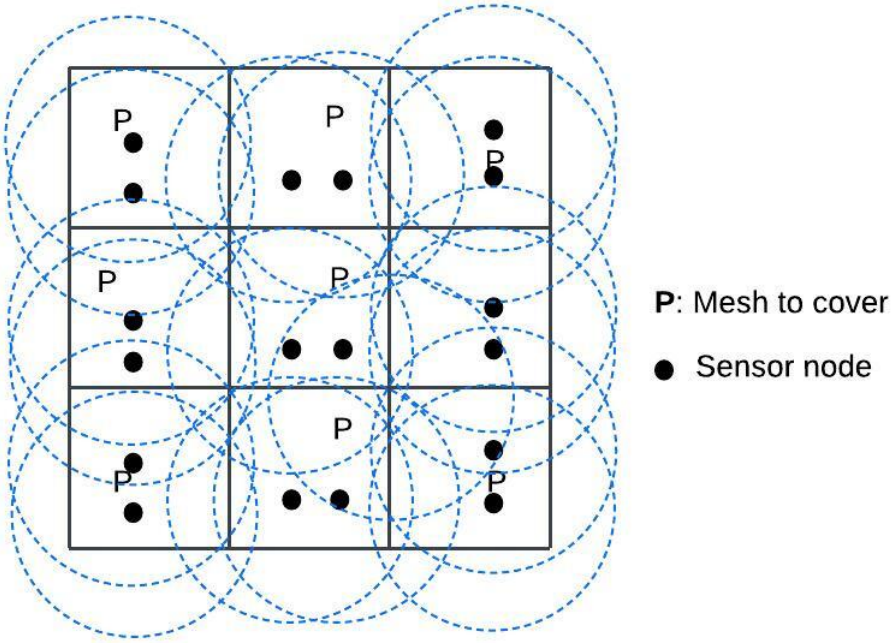


Figure I.14: K-coverage (K=2)

termittent connectivity. Complete network connectivity can also be categorized as simple (1-connectivity) if there is a single path from any sensor node to the sink, or multiple (m-connectivity) if multiple disjoint paths exist between any node and the sink.

I.6.1.3 Latency and Hop count

Latency is a comprehensive metric of network performance. It represents the time required for a node to transmit a data packet to the sink. This time, expressed in terms of slots, includes the waiting time necessary before the transmitting node can send the packet and the time required for the packet to reach the sink. It is worth noting that latency depends on the depth of a node in the routing tree.

The number of hops is an important design and performance criterion in WSNs. It is defined as the number of intermediate nodes through which packets must pass between the source node and the destination node.

Finally, we emphasize that a path with a minimal number of hops does not guarantee optimal latency, and a faster path (with low latency) does not imply a path with a minimal number of hops.

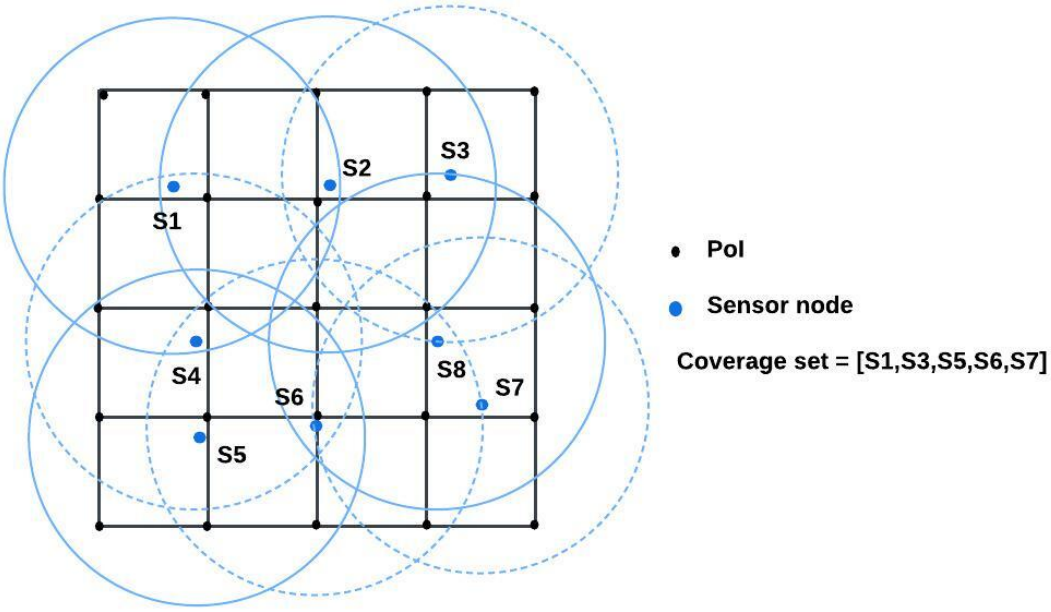


Figure I.15: Zone Coverage

I.6.1.4 Energy

One of the significant constraints in WSNs is the requirement for low-power operation, as each node is powered by a limited battery. Typically, a sensor node consumes its energy in three primary operations: sensing, communication (transmission and reception), and data processing.

I.6.1.4.1 Sensing

The energy consumed during the capture phase varies depending on the observed phenomenon. Sporadic capture consumes less energy than constant detection.

I.6.1.4.2 Communication

The energy consumed in this phase is the highest among the three consumption phases. This consumption is defined by the size of the data to be communicated and the transmission distance, as well as the signal power. When the transmission power is high, the signal will have a longer range, resulting in higher energy consumption.

I.6.1.4.3 Data processing

This phase consumes less energy than communication. In fact, the energy required to transmit 1 KB over a range of 100 m is approximately equal to that needed to execute 3 million instructions at a speed of 100 million instructions per second. Therefore, sometimes it is preferable to process the data locally, especially if they are scalar in nature (such as temperature, humidity).

I.6.1.5 Number of Nodes

The number of nodes deployed in a WSN plays a crucial role in determining the network's cost, performance, and coverage capabilities. Optimal deployment strategies need to consider the trade-off between increasing the number of nodes for improved coverage, latency and minimizing the number of nodes to conserve energy and reduce deployment costs.

I.6.1.6 Overlap

In reality, the sensing range of a node resembles a circle, not a square. Therefore, if we aim to avoid overlap between the sensing ranges of nodes, the network would end up with points that are not covered by the network at all. As a result, to increase coverage of the region of interest, we might accept slight overlap between nodes. This means there will be points within the area of interest where data will be collected by more than one node. Thus, its important to find a balance between overlap and coverage in a WSN [45].

I.6.2 Challenges

There are several challenges that face the progress of WSNs. Among these are the following [53]:

1. **Scalability** Intelligent sensor networks primarily consist of stationary nodes, with network sizes expected to reach tens of thousands of nodes or more. Scalability is a critical concern in designing these networks, as performance improvements should ideally scale proportionally with network size. Algorithms and protocols for WSNs must factor in communication costs relative to network size.

2. **Energy consumption** WSNs are commonly deployed in remote and inaccessible areas like deserts, forests, or military zones, where nodes rely on batteries with limited lifespans since recharging them isn't always feasible.

Consequently, minimizing power consumption becomes crucial. Various protocols and schemes have been proposed to address this challenge, focusing on energy-efficient Medium Access Control (MAC) protocols, data aggregation, topology management, data compression, and intelligent battery usage. Additionally, designing electronic devices and chips with lower power consumption is a key consideration in WSN design.

3. **Self-Organization** In hostile environments where WSNs are deployed, self-organization is essential for ensuring network resilience. Nodes may fail due to harsh conditions or battery depletion, necessitating periodic reconfiguration of the network to maintain functionality. The network must adapt to these changes, enabling continued operation and potentially accommodating new nodes. Even if individual nodes become disconnected, it's imperative that the majority of the network remains operational.

4. **Cost** One of the important issues that WSNs faces is the cost of deploying such networks. The expenses of wireless controllers is significantly influenced by the required memory size. Usually for WSNs designers, having access to variety of chips or wireless micro-controllers with optimized memory sizes is crucial to cater to different application needs. Larger applications such as gateway devices and third-party network layer development necessitate even larger memory sizes, sometimes exceeding 250 KB. This highlights the importance of offering a range of memory options to meet diverse application needs in the development of wireless sensor networks.

5. **Interference and Environment** Interference from nearby wireless networks like Bluetooth or wireless LANs is a common concern for wireless sensor networks (WSNs). For instance, WSNs based on standards like IEEE 802.15.4 or Zigbee often employ automatic repeat capabilities, which can mitigate the impact of Interference from Bluetooth. Similarly, WSNs with occasional transmissions and Bluetooth's frequency hopping feature generally have a low probability of frame collision. Collision avoidance schemes are utilized by wireless Local Area Networks (WLANs) to listen for clear radio-frequency (RF) channels before transmitting data. However, in heavily trafficked WLAN environments, the continuous interference may limit RF channel availability for WSNs. In such cases, it's advisable to set the WSN on a different channel. The RF environment can also be affected by surrounding building structures, which introduce high levels of attenuation and multi-path fading. Additionally, the movement of people or equipment significantly influences signal levels at specific locations. To mitigate the effects of complex building structures, additional router nodes in a mesh network can be strategically installed to bypass obstacles.

6. **Security** WSNs face several security challenges due to the characteristics of the wireless communication medium. These challenges include eavesdropping, man-in-the-middle attacks, spoofing, and distributed denial of service (DDoS) attacks. Security concerns in WSNs can be more significant than those in traditional ad hoc wireless networks because computational and energy consumption limitations often hinder the implementation of robust security solutions. As a result, advancements in the design of security mechanisms in WSNs are crucial to protect confidentiality, availability, and integrity, ensuring the proper operation of these systems. Addressing these challenges is essential to safeguard sensitive data and maintain the functionality of WSNs in various applications.

I.7 WSNs Deployment methods

WSNs facilitate numerous challenging tasks. However, they encounter various challenges, such as localization, which is a fundamental issue. Indeed, deployment significantly impacts other functionalities like coverage, connectivity, energy efficiency, and lifespan. In this section, we provide a comprehensive analysis of different deployment techniques, highlighting the key factors and objectives influencing the deployment phase.

I.7.1 Deployment techniques

The deployment of sensor nodes in a region of interest is the initial phase of constructing a WSN. Generally, There are three types of deployment as shown in Figure fig:deployment-methods: Deterministic deployment, Random deployment and Hybrid deployment. The deployment type is influenced by major factors such as the deployment area, the type of sensor nodes and the application domain.

I.7.1.1 Deterministic deployment

As illustrated in Figure I.17 This approach is used in accessible and non-hostile areas, where sensor nodes can be placed in fixed and known positions according to a predefined method. This type of deployment minimizes the number of required nodes and maximizes the coverage of the detection area.

I.7.1.2 Random deployment

In the case of random deployment as shown in Figure I.18, the deployment region are usually hostile and inaccessible. Sensor nodes are deployed via a drone or aircraft,

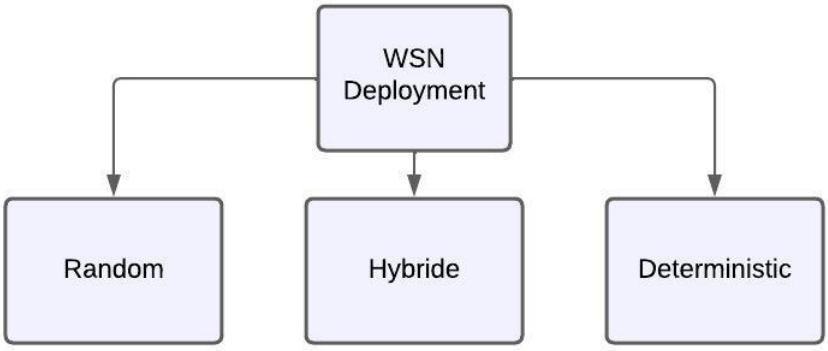


Figure I.16: Deployment methods

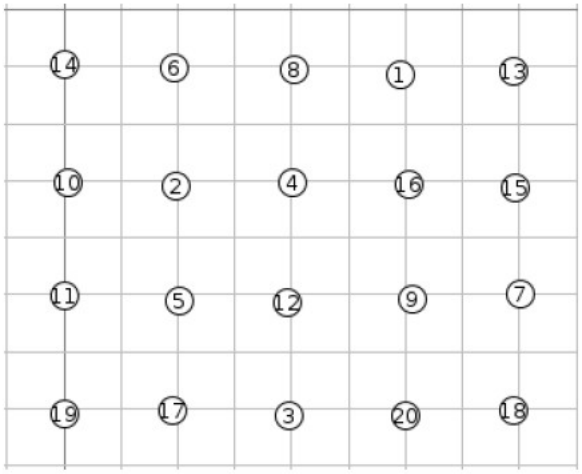


Figure I.17: Deterministic deployment example [20].

resulting in a random distribution of nodes. This type of deployment leads to sub-optimal performance. For instance, this type of deployment does not guarantee network connectivity and coverage.

I.7.1.3 Hybrid deployment

Refers to a combination of deterministic and random deployment methods. In this approach, sensor nodes are placed in a non-deterministic (random) manner within predetermined deployment zones, which are created in a deterministic manner. This means that while the overall layout or structure of the deployment zones is predetermined, the specific placement of sensor nodes within those zones is random. This hybrid approach aims to leverage the benefits of both deterministic and random

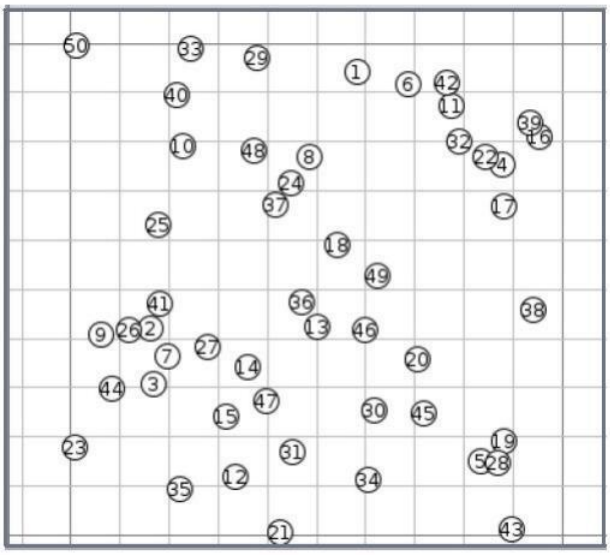


Figure I.18: Random deployment example [20].

deployment techniques.

I.7.2 Nodes placement strategies

Node placement strategies are all deterministic and vary based on one or multiple objectives that need to be achieved (maximizing coverage, maximizing the number of neighbors of a node, etc.). These methods are classified into two groups based on the approach used: force based approach, grid-based approach and geometric algorithmic approach.

I.7.2.1 Force-based approach

According to [20] The force-based deployment strategy relies on sensor mobility, using virtual repulsive and attractive forces. Sensors are compelled to move away from or towards each other to achieve complete coverage. Sensors continue to move until reaching an equilibrium state, where the repulsive and attractive forces balance each other out and eventually cancel each other.

I.7.2.2 Grid-based approach

The grid based strategy offers a deterministic deployment wherein the region of interest get discretized and the positions of sensor nodes are fixed according to a model that maintains a certain distance between nodes, dependant on the Rs (Sensing Range) and the Rc (Communication Range). This can take the form of a triangular network, hexagonal network or square grid.

Each sensor node occupies a cell (mesh), and coverage can be controlled by adjusting the inter-nodes distance d (the distance between two adjacent sensor nodes). To achieve complete coverage, the inter-node distance must be $d < \sqrt{3}$ with r being the Sensing Radius [20]

I.7.2.2.1 Triangular grid

Figure I.19 illustrates that each node has a maximum number of 6 neighbors and a minimum of 2. In terms of connectivity, this type of strategy ensures up to 6-connectivity. However, when it comes to coverage, this model only ensures coverage if $d < \sqrt{3} * Rs$, where d is the inter-node distance (resulting in an equilateral triangle) and Rs is the sensing radius. Note that Rc $\geq$ d, where Rc is the communication range of the nodes.

The minimum number of sensor nodes in this model is $\frac{2\sqrt{3}}{9} * \frac{1}{Rs^2}$ [23]. The triangular model is the most cost-effective among all types of grid-based strategies. Additionally, it provides the smallest overlapping area.

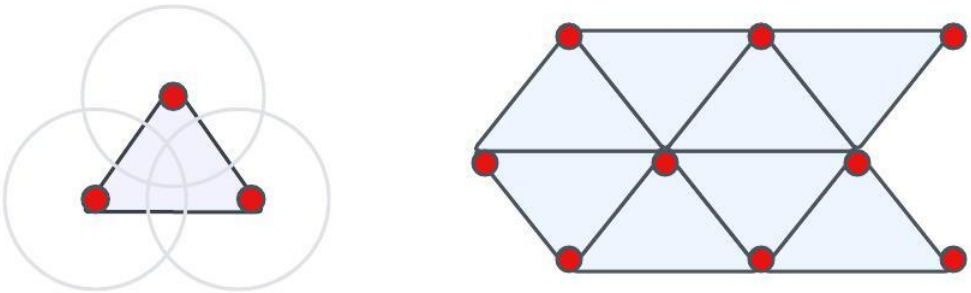


Figure I.19: Triangular model

I.7.2.2.2 Squared grid

In this kind of strategies, the area of interest is divided into squared cells, as illustrated in Figure I.20. The nodes are located either at the corners of the cell or at the center of the cell, We would like to remind that this is the strategy we utilized in the proposed solution within this thesis. In such an architecture, as stated in [23], complete coverage and connectivity are guaranteed if the inter-node distance is $d \leq \sqrt{2} \times Rs$ and $Rc \geq d$

The minimum number of sensor nodes is calculated as: $\frac{1}{2} \times \frac{1}{Rs^2}$

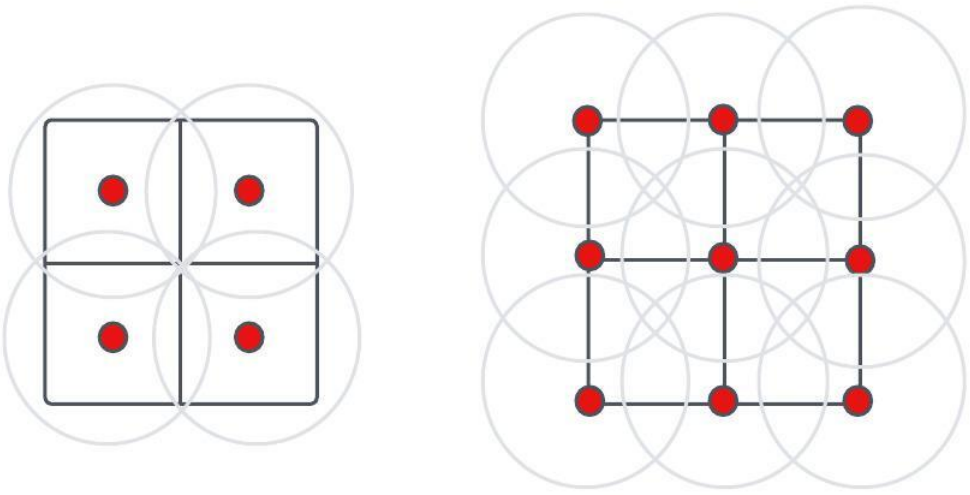


Figure I.20: Squared model

I.7.2.2.3 Hexagonal grid

This kind of strategies is considered the most costly model compared to the triangular and square grid because it has the largest overlapping area. The hexagonal model guarantees total coverage and connectivity for $d \leq Rs$ and $Rc \geq d$. As shown in Figure I.21, the nodes are placed at the vertices of the hexagon.

The minimum number of sensor nodes is calculated as [23]: $\frac{4\sqrt{3}}{9} \times \frac{1}{Rs^2}$

In addition to the type of grid, the size of the grid also plays an important role. It should be chosen based on the network density. For a highly dense network,

smaller grids help reduce coverage gaps, thus providing better results. On the other hand, in a sparsely populated network, a larger grid size is preferable as it prevents overlap in sensor detection ranges, thereby ensuring full utilization of their detection capabilities.

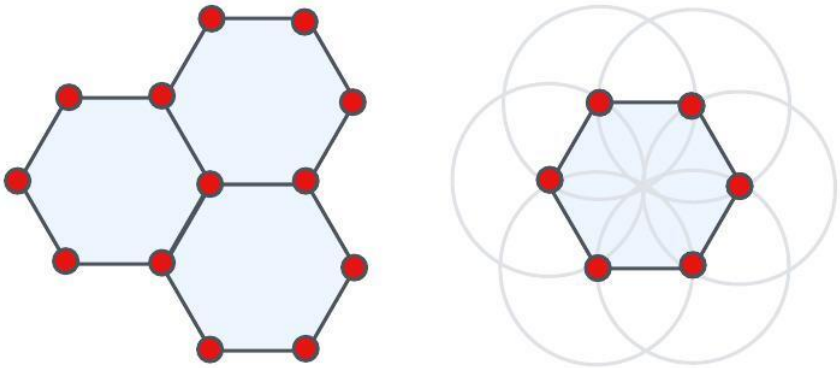


Figure I.21: Hexagonal model

I.7.2.3 geometric algorithmic approach

In this kind of strategies, the most used approaches are:

I.7.2.3.1 Vornoï diagram

The Voronoï diagram is a method of partitioning the area into a certain number of polygons based on distances from a specific discrete set of nodes, as illustrated in Figure I.22. Each node occupies only one polygon and is closer to any point within that polygon than to any other node in neighboring polygons. These polygons can be obtained by drawing the perpendicular bisector of each pair of neighboring nodes. Consequently, the edges of the polygons are equidistant from neighboring nodes. Due to these Voronoï polygons, nodes can determine coverage gaps. They then move to reduce or eliminate these gaps while maximizing the coverage rate of the considered area.

I.7.2.3.2 Delaunay triangulation

Delaunay triangulation is closely related to the Voronoï diagram. Delaunay triangulation is a method for creating a triangular mesh from a set of points, ensuring

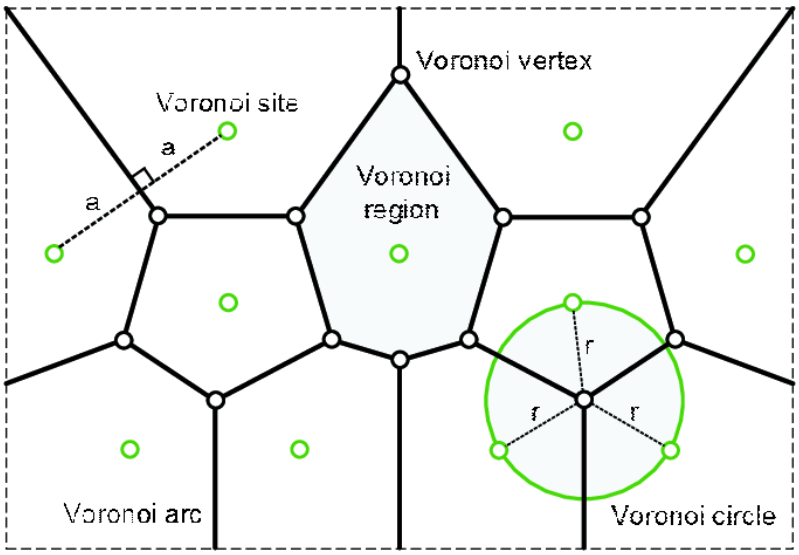


Figure I.22: Vornoï diagram in a plane [25].

that no point lies inside the circumcircle of any triangle formed by the points. This creates a triangulation that maximizes the minimum angle of all the triangles, resulting in more uniformly shaped triangles as shown in Figure I.23.

I.8 Sensitive fenced Areas

Nowadays, the use of WSNs for monitoring sensitive areas is considered as a highly promising research area and a hot topic in literature. In fact, the miniature size of sensor nodes, which allows them to be easily concealed, has facilitated their extensive use in securing strategic sites. These surveillance applications can be found in various scenarios, for instance intrusion detection systems, traffic surveillance as shown in Figure I.24.

I.8.1 Applications and Deployment

In these sensitive fenced areas, WSNs are deployed along the perimeter to detect and respond to any unauthorized intrusion attempts. The nodes used in these networks, known as Sentinel Sensor Nodes (SSNs), are strategically placed along the fence line or boundary of the site. Upon detecting an intrusion, these nodes generate an alert, which is then relayed to a central sink node using Relay Nodes (RNs). The sink node subsequently transmits the alerts to a decision-making center via high-speed communication links, such as the internet, cellular networks, or satellite connections.

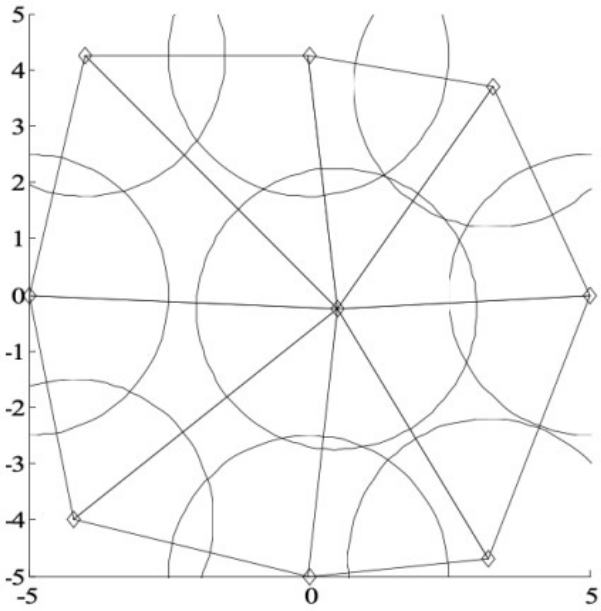


Figure I.23: Delaunay triangluation [6].

For example, in a military base, the WSN can detect and report unauthorized entry attempts or breaches, providing real-time situational awareness to security personnel. In prisons, WSNs can monitor the perimeter for any escape attempts or suspicious activities, ensuring a swift response to potential security threats. Similarly, at nuclear facilities, these networks can monitor critical areas for any signs of intrusion, ensuring the safety and security of sensitive materials.

I.8.2 Design and Implementation Considerations

The design and implementation of WSNs in sensitive fenced areas involve several key considerations:

1. **Node Placement and Density:** The nodes must be placed to ensure complete coverage of the perimeter with minimal gaps. The density of the nodes should be sufficient to provide overlapping coverage areas, ensuring redundancy in case of node failure [2, 75].

2. **Energy Efficiency:** Since sensor nodes are typically battery-powered, energy efficiency is crucial. Techniques such as duty cycling as shown in Figure I.25, where nodes alternate between active and sleep states, can help conserve energy and extend the network's operational lifespan [40].

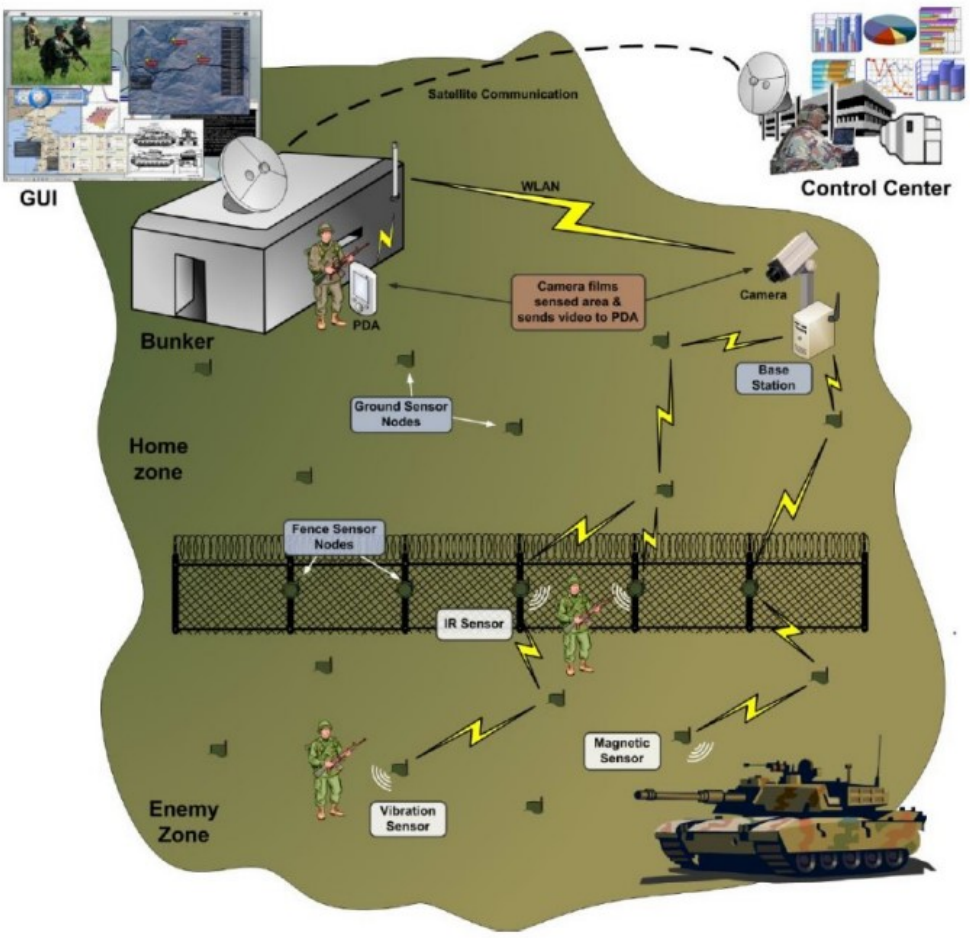


Figure I.24: Fenced military area [69].

3. **Robust Communication Protocols:** The communication protocols used must be robust and resilient to ensure reliable transmission of alerts. This involves implementing fault-tolerant routing algorithms that can adapt to node failures and ensure that alerts reach the sink node [2, 56].

4. **Security and Encryption:** The data transmitted by the sensor nodes must be secure to prevent interception and tampering. Encryption techniques should be employed to protect the integrity and confidentiality of the data [72].

5. **Scalability and Flexibility:** The network should be scalable to accommodate additional nodes as needed and flexible enough to adapt to changing security requirements or environmental conditions [40, 56].

By addressing these considerations, WSNs can effectively enhance the surveil-

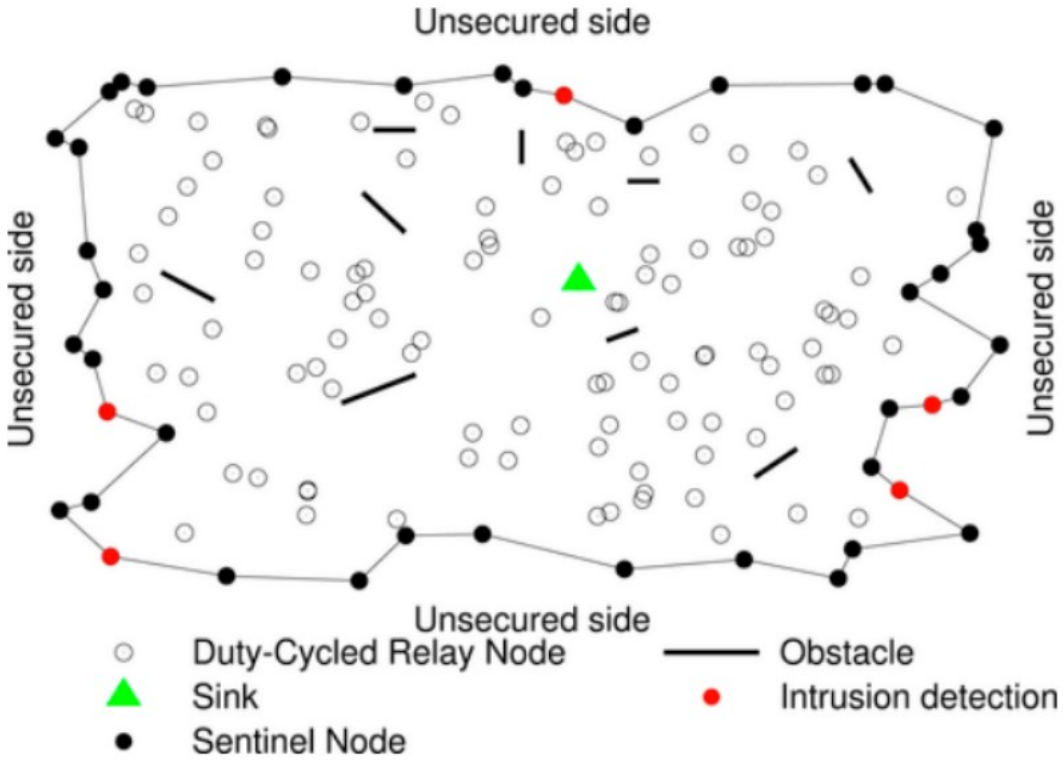


Figure I.25: Randomly deployed WSN based surveillance model [69].

lance and security measures in sensitive fenced areas, providing real-time monitoring and rapid response capabilities. The deployment of WSNs in these environments not only improves security but also reduces the need for extensive human patrols, thereby lowering operational costs and increasing overall efficiency.

In conclusion, the use of WSNs in sensitive fenced areas represents a significant advancement in surveillance technology. With their ability to provide continuous, real-time monitoring, these networks play a crucial role in safeguarding critical infrastructure and strategic sites against unauthorized intrusions and potential security threats. As research and development in this field continue to evolve, the capabilities and applications of WSNs in sensitive fenced areas are expected to expand, offering even greater levels of security and situational awareness.

I.9 Conclusion

WSNs are a continuously evolving research domain with a multitude of application contexts. In this Chapter, we delved into the realm of WSNs and surveillance

systems, exploring various aspects and considerations associated with them. In the first part of this Chapter, we introduced Wireless Sensor Networks (WSNs) by describing their architecture, characteristics, constraints and challenges influencing their design, as well as their applications. In the second part, we highlighted the various deployment strategies of WSNs in addition to the influence of deterministic placement of sensor nodes on the performance of these networks, especially in terms of coverage, connectivity, cost and network lifespan. Furthermore, we explored the application of WSNs in sensitive fenced areas, highlighting their role in enhancing surveillance and security measures.

In the next Chapter, we will present a state-of-the-art overview of combinatorial optimization techniques, both single and multi-objective as well as machine learning techniques aimed at enhancing the performance criteria of a WSNs dedicated to the surveillance of sensitive fenced areas, through appropriate node placement.

Chapter II

Machine Learning based Meta-Heuristics to Solve Combinatorial Optimization Problems

II.1 Introduction

The intersection between Machine Learning (ML) techniques and combinatorial optimization methods is an exciting domain which becomes increasingly common. Machine learning has revolutionized many fields, including computer vision, natural language processing, and robotics, by enabling algorithms to learn from data and improve their performance over time. On the other hand, combinatorial optimization techniques have been instrumental in solving complex problems in operations research, logistics, and scheduling, among others, by finding the best solution among a finite set of possibilities. In recent years, there has been a growing interest in combining these two powerful approaches to tackle challenging problems that are beyond the reach of traditional methods. By integrating machine learning techniques with combinatorial optimization methods, we can develop intelligent systems that can learn from data and adapt to new situations while still providing optimal or near-optimal solutions.

This chapter delves into the synergy between machine learning and meta-heuristics, exploring how these advanced methodologies can be leveraged to solve combinatorial optimization problems. redWe begin by reviewing works that use exact and approximate methods to solve the WSN deployment problem, then we review the research works where authors utilize ML to solve different problems related to WSN. Subsequently, we examine Combinatorial Optimization Problems (COPs) and the complexity issues they present. We then explore how to solve (COPs) using different optimization approaches, including exact and approximate methods

(heuristic/meta-heuristic algorithms), followed by a discussion on single-objective and multi-objective optimization techniques. Finally, we provide an introduction to ML, focusing on how the Reinforcement Learning (RL) paradigm can be applied to enhance meta-heuristic algorithms, thereby paving the way for innovative solutions in combinatorial optimization.

II.2 Literature Review

Numerous researchers have collaborated to enhance the Wireless Sensor Network (WSN) deployment process, aiming to make them more efficient, reliable, and high-performing, while simultaneously reducing energy consumption, lowering deployment costs, and extending the lifespan of nodes and network coverage. To achieve these objectives, researchers employed either exact or approximate methods, or a combination of traditional methods (exact and approximate) and intelligence based methods such as genetic algorithms (GAs).

II.2.1 Optimizing WSN Deployment using Exact and approximate Methods

Authors in [7] introduced a Multi-objective optimization technique tailored for WSNs employing a modified Genetic Alogrithm (GA) that emulates the movement of Particle Swarm Optimization (PSO) particles. The authors aimed to minimize the number of Relay Nodes (RNs) deployed and the associated communication costs. A series of simulations were conducted under diverse scenarios, varying parameters such as the ratio of GA chromosomes to PSO particles, as well as the number and positioning of sensors. Near-optimal outcomes were attained using a GP90 ratio, wherein 90% of the 200 chromosomes were generated utilizing GA characteristics, while the remaining 10% were treated as PSO particles. Notably, this approach surpassed alternative algorithms including Dijkstra, A-Star, GA, and PSO in terms of RN usage and communication cost efficiency. The employed method generates two distinct topology designs. One design emphasizes cost efficiency by minimizing the number of RNs, while the other prioritizes reduced communication costs despite requiring more RNs. The selection between the designs depend upon the user's preference.

Authors in [60] developed an integer linear programming model to find an optimal solution. Additionally, they proposed two approximate methods: a Local Search (LS) algorithm and a Genetic Algorithm (GA). These methods aim to provide efficient solutions to the problem, although they may not guarantee optimality. Through computational experiments, the authors demonstrated that their integer

linear programming model, implemented using CPLEX, is capable of finding optimal solutions for small and medium-sized instances of the problem. Moreover, they showed that their proposed methods outperform conventional sensor deployment patterns. Overall, this study contributed to the literature on WSN deployment by presenting both exact and approximate methods for addressing the critical coverage problem, thereby offering insights into optimizing sensor deployment strategies for improved network performance.

Authors in [68] introduced a novel approach to achieving optimal network coverage in a two-dimensional Euclidean area using a GA. The objective is to strategically place a specified number of sensors to ensure comprehensive coverage. The focus is on tackling the maximum coverage problem, emphasizing the calculation of the total area covered by deployed sensor nodes. The proposed algorithm is designed to find the optimal positions for a given number of sensors, maximizing the coverage of the sensor area. By utilizing a genetic algorithm, the research aimed to enhance network coverage efficiency, particularly when the number of sensors is limited. The results of the study indicate that the proposed method effectively maximizes the coverage of the sensor area, showcasing its potential to address the challenges associated with optimal coverage in WSNs.

Authors in [69] addressed the problem associated with the deployment of WSNs for surveillance applications. Walid TOUIL proposed a Basic Variable Neighborhood Search algorithm (BVNS) meta-heuristic, with the greedy algorithm used to find initial solution of the problem. The author also used The weighted sum method to solve the multi objective problem. The obtained results, in terms of number of RNs deployed, hop count and latency, were very encouraging compared to an exact method.

II.2.2 Machine Learning Algorithms for Addressing WSNs Challenges

This section of the literature review discusses various machine learning (ML) techniques that can be utilized to address the diverse problems and challenges encountered in WSNs.

Researchers in [3] introduced a data compression algorithm with error bound guarantee for WSNs using compressing neural networks. The proposed algorithm minimizes data congestion and reduces energy consumption by exploring spatio-temporal correlations among data samples. The adaptive rate-distortion feature balances the compressed data size (data rate) with the required error bound guar-

antee (distortion level). This compression relieves the strain on energy and bandwidth resources while collecting WSN data within tolerable error margins, thereby increasing the scale of WSNs. The algorithm is evaluated using real-world datasets and compared with conventional methods for temporal and spatial data compression. The experimental validation reveals that the proposed algorithm outperforms several existing WSN data compression methods in terms of compression efficiency and signal reconstruction. Moreover, an energy analysis shows that compressing the data can reduce the energy expenditure, and hence expand the service lifespan by several folds.

Authors in [8] presented a novel tree-based data aggregation approach for periodic sensor networks, leveraging correlation matrix and polynomial regression techniques. The proposed approach involves aggregating data at two sensor levels, where each sensor node generates a polynomial function of the captured data and transmits the coefficients of regression along with the polynomial function to the aggregator and sink levels. By utilizing similarity functions and correlation matrices, the approach aims to efficiently aggregate data while maintaining accuracy and reducing energy consumption.

Authors in [62] introduced QL-MAC, a novel energy-preserving Medium Access Control (MAC) protocol derived from Q-learning. This protocol aims to extend the network lifetime by iteratively tweaking MAC parameters through a trial-and-error process, ultimately converging to a low-energy state. QL-MAC offers flexibility and adaptability to varying network conditions without the need for predetermining the system model. It provides a self-adaptive protocol capable of adjusting to topological and external changes, ensuring optimal performance under dynamic environments. By dynamically altering radio sleeping and active periods based on traffic predictions and the transmission state of neighboring nodes, QL-MAC significantly reduces energy consumption while maintaining satisfactory network performance. Experimental validation demonstrates the protocol's efficacy in off-the-shelf devices, supplemented by large-scale simulations, showcasing its potential to address energy efficiency challenges in high-density communication scenarios in Wireless Sensor Networks (WSNs).

The authors in [39] proposed a novel approach to enhance the sustainability of WSN nodes by extending their lifetimes. The authors aimed to optimize the collection interval and transmission interval for each task using machine learning techniques. Initially, they employed the wrapper method to determine the optimal combination of nodes required to perform each task efficiently. Subsequently, Simulated Annealing (SA) is applied to identify the values of these parameters that

minimize power consumption without compromising the WSN's performance significantly. To validate the effectiveness of their method, the authors conducted two sets of experiments, demonstrating a reduction in energy consumption using their framework. By leveraging supervised learning techniques, the proposed approach offers a promising solution to extend the lifespan of WSN nodes while maintaining optimal network performance.

The authors [13] investigated the effectiveness of machine learning algorithms in localizing nodes in large-scale Wireless Sensor Networks (WSNs). Unlike traditional methods that used iterative triangulation, the study approached localization as a regression problem rather than a classification one. The authors proposed novel feature vector definitions and evaluated the impacts of various network parameters, such as network size, anchor node population, transmitted signal power, and wireless channel quality, on localization accuracy. They compared random and grid placements of anchor nodes, finding that machine learning models, specifically multivariate regression and Support Vector Machine (SVM) regression with a radial basis function (RBF) kernel, offered promising accuracy. The results highlighted that these machine learning-based methods mitigated the error propagation common in traditional approaches, demonstrating significant potential for improving localization in WSNs.

II.3 Introduction to Combinatorial optimization

> True optimization is the revolutionary contribution of modern research to decision processes
>
> *George Dantzig*

Combinatorial optimization is a branch of mathematical optimization that has applications in artificial intelligence, applied mathematics, software engineering, and many other domains. Optimization problems are usually divided in two categories. Those with continuous variables and those with discrete variables which are called *combinatorial*. When working with continuous problems we are generally looking for a set of real numbers or even a function. In the combinatorial problems, we are looking for an object from a finite or possibly infinite set, typically an integer set, permutation or graph, those candidate objects are called *feasible solutions* while the optimal one is called an *optimal solution*. In our work we focus mainly on discrete optimization or *combinatorial optimization problems* and the process of finding an

optimal solutions in a well defined discrete space. Today, combinatorial optimization finds widespread application in the study of algorithms, and it holds particular relevance in our case for optimizing WSN nodes deployment. For example, consider following problem.

Problem 1 *The traveling salesman problem (TSP)* has kept researches busy for the last 100 years by its simple formulation, important applications and interesting connections to other combinatorial problems. An article related to the traveling salesman problem was treated by the Irish mathematician Sir William Rowan Hamilton in the 1800s and later an article was published by the British mathematician Thomas Penyngton Kirkman in 1855.

The salesman wishes to make a tour visiting each city exactly once and finishing at the city he starts from. A tour is a closed path that visits every city exactly once. There is a integer cost C_{ij} to travel from city i to j and the salesman wishes to make the tour with a minimal cost. The total solution's cost becomes the sum of individual costs along the edges of the tour. The travel costs are symmetric in the sense that traveling from city i to city j costs just as much as traveling from city j to city i. The Traveling Salesman Problem (TSP) can be formulated mathematically as follows:

Let $G = (V, E)$ be an undirected graph, where V is the set of vertices (cities) and E is the set of edges (roads) connecting the vertices. Each edge e_{ij} has a non-negative weight c_{ij}, representing the cost (or distance) to travel from city i to city j. This is closely related to the *Hamiltonian circuit* of the graph as illustrated in Figure II.1. If there are n cities to visit, the number of possible solutions or tours in finite. To be precise it becomes $(n-1)!$. Hence an algorithm can easily be designed that systematically examines all tours in order to find the shortest tour.

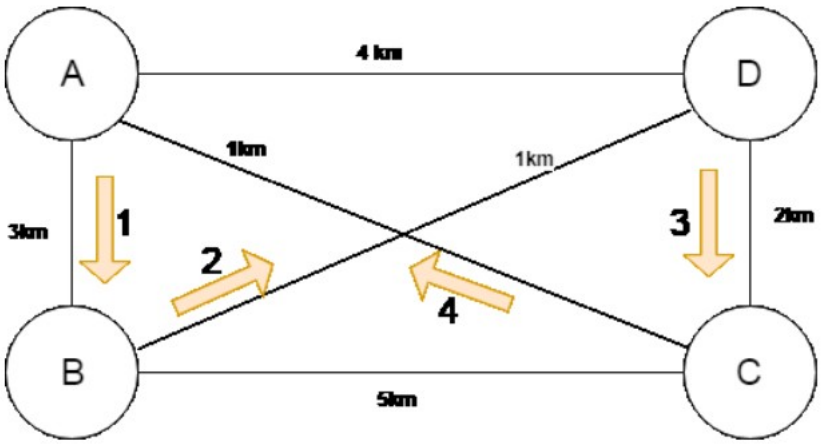


Figure II.1: Hamiltonian Cycle [6].

Formally, the TSP can be represented as an optimization problem:

$$Minimize : \sum_{(i,j)\in E} c_{ij} \cdot x_{ij}$$

Subject to:

- Each city must be visited exactly once: $\sum_{j\neq i} x_{ij} = 1, \quad \forall i \in V$
- Each city must leave exactly once: $\sum_{i\neq j} x_{ij} = 1, \quad \forall j \in V$
- Sub-tour elimination constraints to prevent loops:
  - $\sum_{(i,j)\in S} x_{ij} \leq |S| - 1, \quad \forall S \subset V,\ S \neq \emptyset$
- Binary constraint on decision variables: $x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$
  - x_{ij} is a binary decision variable, indicating whether the edge e_{ij} is included in the tour (1 if included, 0 otherwise).
  - Constraints 1 and 2 ensure that each city is visited and left exactly once.
  - Constraint 3 prevents the formation of subtours, ensuring that the tour is connected and does not contain loops.
  - Constraint 4 enforces the binary nature of the decision variables.

Problem 2 *The Knapsack problem* The classical binary knapsack problem (denoted by 0-1 KP) is one of the most linear integer programming belonging to the combinatorial optimization family. There are several applications that can be modeled as 0-1 KPs, such as cargo loading. Let us consider a simple example: suppose a traveller has a travelling bag (knapsack) that takes a maximum of c kg of items. The traveller has n items $(1, 2, 3, ..., n)$, their weights are w_i and they have values associated of p_i. In this case, the traveller should place the items so that it maximizes the value of the knapsack while note exceeding the maximum weight of the bag. The Binary Knapsack Problem can be formulated mathematically as follows:

Let n be the number of items available for selection. Each item i has a weight w_i and a value v_i. We also have a knapsack with a capacity W. Mathematically, the Binary Knapsack Problem can be formulated as an optimization problem:

$$Maximize : \sum_{i=1}^{n} v_i \cdot x_i \tag{II.1}$$

Subject to:

$$\begin{aligned} &\sum_{i=1}^{n} w_i \cdot x_i \leq W, \\ &x_i \in \{0, 1\} \quad \text{for} \quad i = 1, 2, \ldots, n \end{aligned} \tag{II.2}$$

Where:

- x_i is a binary decision variable representing whether item i is selected (1 if selected, 0 otherwise).
- The objective function maximizes the total value of the selected items.
- The constraint ensures that the total weight of the selected items does not exceed the capacity W of the knapsack.
- w_i and v_i represent the weight and value of item i, respectively.

This formulation defines the Binary Knapsack Problem as a mixed-integer linear programming (MILP) problem, which can be solved using optimization techniques such as integer programming solvers or dynamic programming algorithms.

II.3.1 Computational Complexity

Most of these problems are referred to as NP-hard problems because depending on the size of the problem and the number of objectives to optimize, there are no algorithms that provide an exact solution in polynomial time.

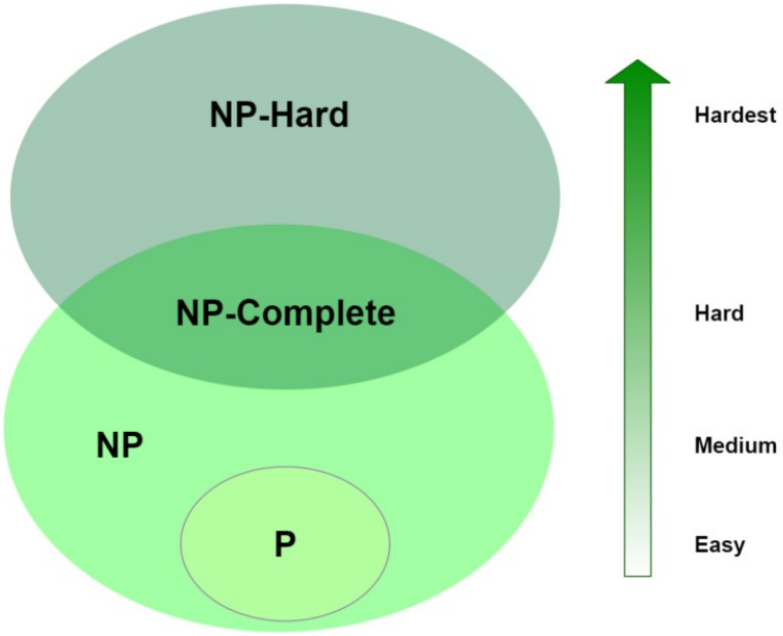


Figure II.2: Complexity classes labeled from Easy to Hard scale [69].

- **The class P** consists of decision problems solvable by deterministic algorithms within polynomial time. It encompasses tasks like searching, element uniqueness, and determining graph connectivity and acyclicity. These algorithms yield yes or no answers without explicit output statements, forming the foundation for efficient problem-solving across diverse domains [24].
- **The class NP (Non-deterministic Polynomial)** represents problems whose solutions cannot be found in polynomial time. However, they can be verified in polynomial time [21].
- **The class NP-complete:** This is a decision problem that belongs to the NP class, and the best-known resolution algorithm to date is exponential in the size of the problem in the worst-case scenario [21]. This class represents problems that can be expressed in the form of a mathematical question whose answer can only be "yes" or "no". For example, the primality test is a decision problem that involves answering the following question: given an integer n, is n prime?
- **The class NP-hard:** This is a problem whose solution cannot be verified in polynomial time, and any NP-complete problem can be polynomial reduced to this problem. NP-hard problems are therefore at least as difficult as NP-complete problems as shown in Figure II.2. In general, this class is used for optimization problems for which the solution cannot be verified in polynomial time. Examples of NP-hard problems include the traveling salesman problem and the chromatic number of a graph.

The P versus NP problem The P versus NP problem was introduced independently in 1971 by Stephen Cook and Leonid Levin. Since that time, extensive efforts have been made to find a proof for this problem, but no definitive solution has been discovered thus far it asks whether P is equal to NP or not. In other words, it asks whether every problem for which a solution can be verified in polynomial time can also be solved in polynomial time [69].

If $P = NP$, we could find solutions to search problems as easily as checking whether those solutions are good as illustrated in Figure II.3. This would essentially solve all the algorithmic challenges that we face today and computers could solve almost any task. However because there are problems for which no efficient algorithm exists, and finding a solution requires an exponential amount of time. In this case, there would always be a gap between verifying a solution and finding it thus $P \neq NP$ [69].

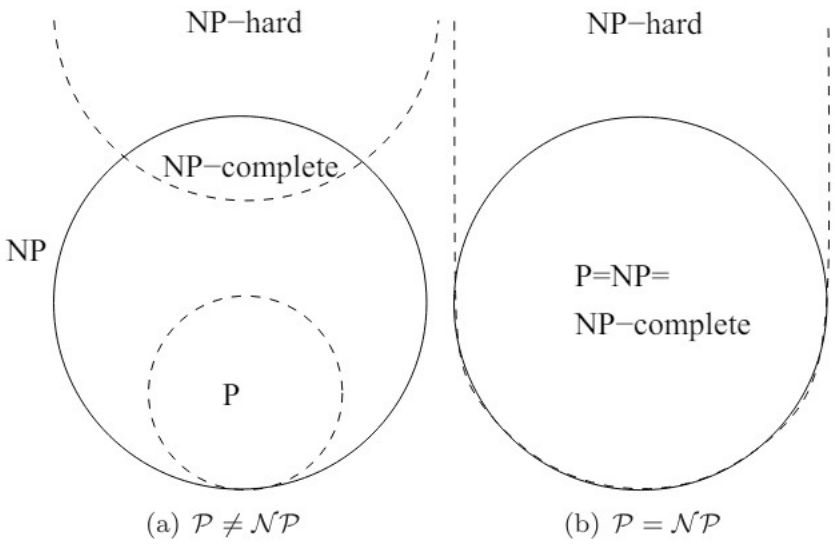


Figure II.3: The Venn diagram for P, Np, NP-complete and NP-hard set of problems [24].

II.3.2 Optimization Techniques for Combinatorial Problems

Combinatorial optimization problems present significant complexity, driving the development of diverse techniques to navigate their solution space effectively. These methods include exact approaches ensuring optimality, heuristic methods swiftly delivering satisfactory solutions, and meta-heuristic methods offering strategic guidance. Additionally, approximation algorithms provide solutions with known quality bounds. Employing these techniques empowers researchers to tackle real-world challenges across various domains effectively. Let's delve deeper into some commonly utilized techniques.

II.3.2.1 Exact Methods

Exact methods, such as branch and bound and dynamic programming, offer guaranteed optimality. Typically applied to smaller-scale problems. They efficiently explore solution spaces within reasonable time frames. Larger problems pose challenges, as solving them exactingly could lead to exponential increases in computational time.

II.3.2.1.1 Branch and Bound Branch and Bound (BB) algorithms are utilized to find optimal solutions for combinatorial, discrete, and general mathematical optimization problems. Generally, when faced with an NP-Hard problem, a branch

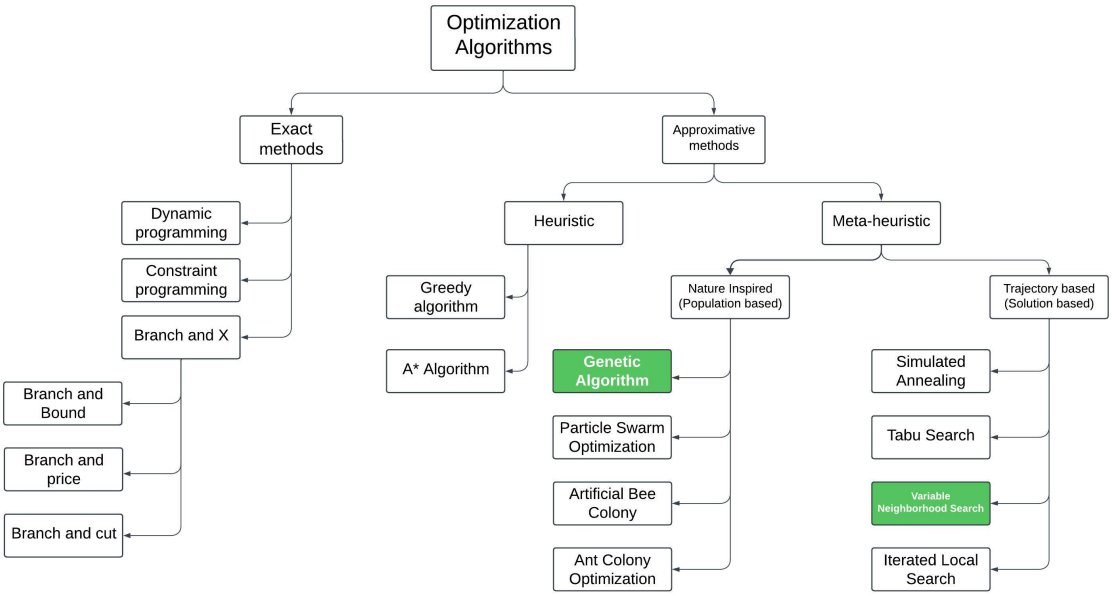


Figure II.4: Optimization methods classification

and bound algorithm systematically explores the entire search space of potential solutions and provides an optimal solution.

Branch and bound algorithms are methods for global optimization in non-convex problems. They are non-heuristic, in the sense that they maintain a provable upper and lower bound on the (globally) optimal objective value; they terminate with a certificate proving that the suboptimal point found is ϵ-suboptimal. Branch and bound algorithms can be (and often are) slow, however. In the worst case they require effort that grows exponentially with problem size, but in some cases we are lucky, and the methods converge with much less effort [14]. In these notes we describe one typical and simple example of branch and bound methods to solve the knapsack problem stated before.

0/1 Knapsack problem using Branch and Bound Given two arrays $v[]$ and $w[]$ that represent values and weights associated with n items respectively. Find out the maximum value subset(Maximum Profit) of $v[]$ such that the sum of the weights of this subset is smaller than or equal to Knapsack capacity (maximum weight of the knapsack) W. The branch-and-bound method is based on three main principles:

1. **Separation principle:** This involves dividing the problem P into sub-problems Pi based on a certain criterion. Each sub-problem has its own set of feasible solutions contained within a vertex of the tree, as shown in Figure II.5. The

union of the subsets associated with the children of a vertex must be equal to the set associated with that vertex. The optimal solution is obtained by calculating the objective function value for all non-empty leaves of the tree [6].

2. **Evaluation:** This allows defining the optimum for the current subset when the sub-problem becomes simple and can be solved directly. This method also helps avoid separating certain vertices of the tree when it is proven that the set associated with the node in question does not contain candidate solutions to optimality. This is possible thanks to the knowledge of a lower bound (respectively an upper bound) for each sub-problem. If we can find a lower bound (in the case of a minimization problem) greater than the best solution found so far, we can say that the subset does not contain the optimum [6].

3. **Traversal strategy:** To choose the next vertex to separate from the set of vertices in the tree, there are several traversal strategies [6]. These include:

   - Depth-first: This strategy favors vertices furthest from the root by applying more separations to the initial problem.
   - Breadth-first: This strategy prioritizes sub-problems obtained with the fewest separations from the starting problem (i.e., the closest vertices).
   - Best-first: This involves exploring sub-problems where the probability of finding a better solution is highest.

Figure II.5 depict the 0/1 knapsack problem using B&B method. There are also other similar algorithms, such as Branch and Price (a hybrid of branch and bound and column generation algorithms), which is used for solving problems with large solution spaces, and Branch and Cut (a hybrid of branch and bound and cutting planes algorithms), which is employed to solve mixed-integer programming problems [19].

II.3.2.1.2 Dynamic programming Dynamic programming is an algorithmic approach for investigating an optimization problem by breaking it down into smaller, simpler sub-problems. A key aspect of dynamic programming is the proper structuring of optimization problems into multiple levels and solving them in a sequential manner, one level at a time. Each level is solved using typical optimization techniques, and the solution obtained helps to define the characteristics of the next level problem in the sequence. Typically, these levels correspond to distinct time periods within the overall problem [69]. Dynamic programming is usually used to solve problems such as the knapsack problem, Hanoi tour and shortest path using Dijkstra

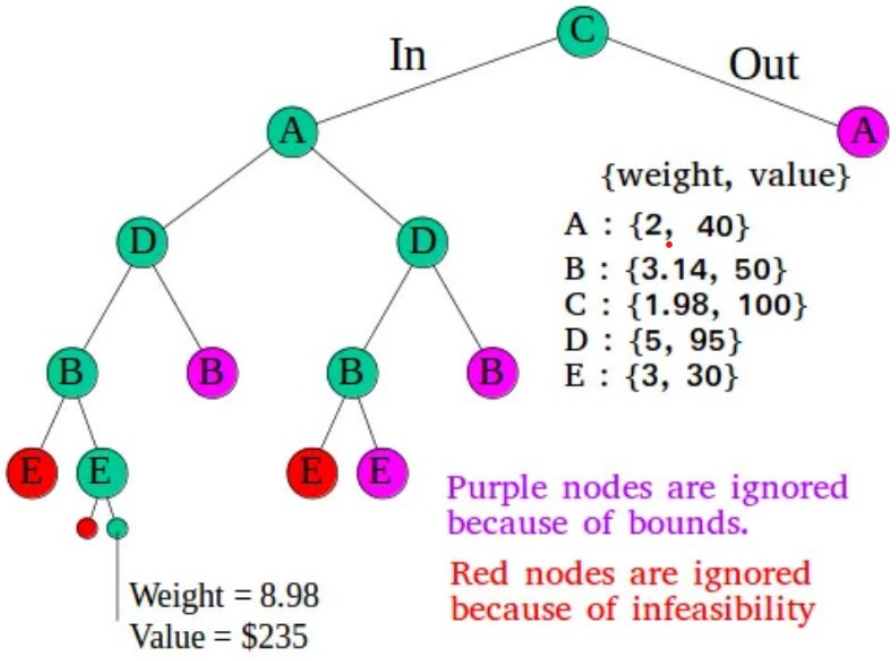


Figure II.5: 0/1 knapsack problem using B&B method [27].

Dijkstra's Algorithm

Dijkstra's algorithm is a fundamental graph algorithm used to find the shortest path from a source vertex to all other vertices in a weighted graph with non-negative edge weights. Unlike the Floyd-Warshall algorithm, Dijkstra's algorithm is specifically designed for finding single-source shortest paths. The Dijkstra algorithm 1 is as follows [73]:

Dijkstra's algorithm maintains a priority queue Q of vertices ordered by their current tentative distance from the source vertex. It repeatedly extracts the vertex with the shortest tentative distance form Q, relaxes all its outgoing edges, and updates their tentative distances accordingly. continues until all vertices have been processed, resulting in the shortest path distances stored in the array *dist* [73].

II.3.2.2 Approximate Methods

II.3.2.2.1 Heuristics

Heuristics, also known as approximate methods were first introduced by G. Polya in 1945, in order to provide good and feasible solutions in a reasonable or polyno-

Algorithm 1 Dijkstra's Algorithm

Input: A graph G with $V(G) = 1, \ldots, n$, a source vertex s, and non-negative edge weights $w : E(G) \to \mathbb{R}_{\geq 0}$
Output: An array $dist$ such that $dist[v]$ contains the length of the shortest path from source vertex s to vertex v
$dist[s] \leftarrow 0$
Initialize priority queue Q with all vertices, with $dist[v] = \infty$ for all $v \neq s$
while Q is not empty **do**
    $u \leftarrow$ Extract-Min(Q) {Extract vertex with minimum $dist$ from Q}
    for each neighbor v of u **do**
        if $dist[u] + w((u, v)) < dist[v]$ **then**
            $dist[v] \leftarrow dist[u] + w((u, v))$
            Decrease-Key($Q, v, dist[v]$) {Update priority queue}
        end if
    end for
end while
return $dist$

mial time but are not necessarily optimal. These techniques are designed to solve specific problems and do not explore the entire search space (they are not complete methods) unlike exact methods. Among these heuristics, we can mention greedy algorithms.

Greedy Algorithm

Greedy algorithms are used to solve combinatorial optimization problems typically by going through a sequence of steps, with a set of choices at each step. It makes a locally optimal choice (solving the sub-problem) in the hope that this choice will lead to a globally optimal solution. Greedy algorithms do not always provide optimal solutions, but they do so for many problems such as the coin change problem and graph coloring problems.

The Figure II.6 below depicts a comparison of the results obtained from the Greedy algorithm heuristic and an optimal algorithm in finding the shortest path from node A to node J in a weighted graph:

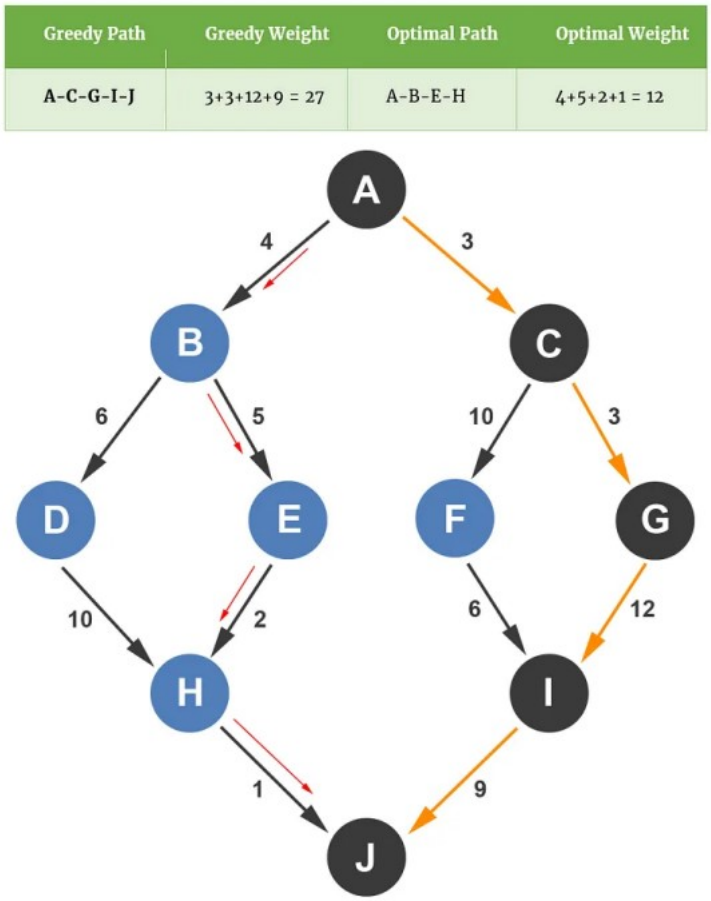


Figure II.6: Comparison between Greedy and Exact algorithms [69].

During his master's thesis [69], Walid TOUIL employed a greedy algorithm as a precursor to initializing the Variable Neighborhood Search (VNS) algorithm. The Greedy algorithm demonstrated its efficacy by yielding excellent results in the context of the VNS algorithm.

A* Algorithm

The A* algorithm, originally introduced by Peter E. Hart, Nils John Nilsson, and Bertram Raphael in 1968 [33], addresses the shortest path problem in a graph between a source node and a destination node. It serves as an extension of Dijkstra's algorithm, aiming to provide one of the best solutions efficiently. Given a graph, starting from a source node, the algorithm selects the node that minimizes the predefined cost until reaching the destination node. This principle is known as

the *"Best First Search."* A* utilizes a cost estimation function for selecting the next node to visit, denoted as $d(i)$, such that:

$$d(i) = g(i) + h(i) \tag{II.3}$$

Where:

- $d(i)$ represents the estimated total cost of the path.
- $g(i)$ denotes the estimated length of the shortest path from the origin s to node i.
- $h(i)$ indicates the estimated length of the shortest path from node i to the destination t.
- If $h(i) = 0$ for all nodes, then A* is identical to Dijkstra's algorithm.

II.3.2.2.2 Meta-heuristics

Meta-heuristics represent a class of optimization algorithms designed to tackle complex combinatorial optimization problems that are otherwise computationally intractable with exact methods. Unlike exact algorithms that aim for optimality, metaheuristics provide efficient approximate solutions by navigating through the solution space using heuristic rules. They serve as powerful problem-solving tools, offering flexibility and scalability across various domains, from engineering to operations research and beyond.

At its core, a metaheuristic is a higher-level strategy guiding the exploration of the solution space. Instead of exhaustively examining all possible solutions, metaheuristics iteratively refine a population of candidate solutions, leveraging strategies inspired by natural phenomena, human behavior, or mathematical principles.

Two prominent categories of metaheuristics are trajectory-based and nature-inspired metaheuristics. Trajectory-based metaheuristics navigate the solution space by iteratively refining a single solution, gradually moving towards the optimal or near-optimal solution. Examples include *simulated annealing*, and *tabu search*. These algorithms adjust the current solution based on a predefined neighborhood structure, seeking improvements through local search operations.

On the other hand, nature-inspired metaheuristics draw inspiration from natural phenomena or biological processes to guide the search for optimal solutions. These algorithms mimic the behavior of complex adaptive systems observed in nature, such as evolutionary processes, swarm behavior, and physical phenomena. Nature-inspired metaheuristics include *genetic algorithms*, *ant colony optimization* and *particle swarm optimization*. By harnessing the principles of self-organization,

cooperation, and adaptation observed in natural systems, these algorithms offer robust and efficient optimization techniques capable of addressing a wide range of complex problems.

In this section, we delve into the intricacies of trajectory-based and nature-inspired meta-heuristics, exploring their underlying principles, algorithmic frameworks, and applications across diverse domains. Through comprehensive analysis and empirical evaluation, we aim to provide insights into the strengths, weaknesses, and best practices associated with these powerful optimization techniques.

Simulated Annealing

The combinatorial optimization algorithm Simulated Annealing was independently developed by three researchers at IBM, S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi in 1983 [42], and by Cerny in 1985 [15]. The inspiration for this method originates from an analogy with metallurgy, where a process called "annealing" involves heating a solid to a high temperature and then slowly cooling it to achieve low-energy states. The objective is to reach a state of quasi-thermodynamic equilibrium, which corresponds to the solution of the optimization problem.

The method is applied to combinatorial optimization problems. It begins by generating an initial solution s randomly from the search space with the aim of minimizing an objective function f, where $f(s)$ represents the energy. Then, it determines a parameter T that decreases towards 0. At each iteration, the solution is perturbed to induce a change Δ in the system's energy. If this change reduces the energy of the system, it is applied to the current solution. Otherwise, the solution is accepted with a probability of $\exp(\frac{\Delta}{T})$. The theory demonstrates that simulated annealing provides an approximate solution to the optimal solution of the problem more quickly than exhaustive exploration. In practice, it is necessary to use the appropriate internal parameters of the algorithm to accelerate convergence towards a pseudo-optimal solution [6].

Tabu Search

The Tabu Search algorithm was proposed by Glover in 1986 [31] and experienced significant success in the 1990s, demonstrating its performance on numerous combinatorial optimization problems. To this day, it remains one of the most widely used single-solution metaheuristics. The tabu search pseudo code of the algorithm 2 is given below:

Algorithm 2 Tabu Search (TS)

Input: Initial solution s_0, stopping criteria
Output: Best found solution s'
Initialize Tabu lists $(TL_1, TL_2, \ldots, TL_r)$
$s' = s$
repeat
  Find the best admissible solution s_1
  if $f(s_1) > f(s')$ **then**
    $s' = s_1$
  else
    $s = s_1$ and update Tabu list TL
  end if
until stopping criteria

The key feature of this method is the use of memory to record information associated with the search process, making the search somewhat intelligent. The idea is to move from the current solution X to another solution Y such that Y is a neighbor of X. In cases where none of the neighboring solutions improve upon the current solution, the method accepts the least desirable solution to avoid local optima. Typically, all neighbors are explored deterministically as in local search, and the best neighbor replaces the current solution.

When faced with a local optimum, the search continues by selecting the least unfavorable candidate compared to the current solution. The best neighbor is considered as the new solution even if it does not improve upon the current solution, allowing for revisiting previously examined solutions, leading to cycles. To prevent this phenomenon, the method maintains a memory of recently visited solutions called the Tabu List. This list contains all the movements that have already been made and are therefore prohibited (taboo). This prevents falling into a cycle of repetitive movements and escaping from local minima [36].

Genetic Algorithms

Genetic Algorithm (GA) is a class of Evolutionary Algorithm (EA) which generates solutions to optimization problems using techniques inspired by natural evolution, such as mutation, selection, and crossover. It is a search algorithm based on the mechanics of natural selection and natural genetics to solve usually mathematical optimization of search algorithm in computational algorithm. John Holland firstly put forward GA in 1975 [34] when he worked on the studies of cellular automata with his colleagues and his students at the University of Michigan. GA became popular through his book Adaptation in Natural and Artificial Systems (Holland,

1975).

GAs are useful when attempting to solve problems that a) are computationally complex to solve optimally and b) have no trivial heuristic model to approximate a good solution [66].

The process of Genetic Algorithms There are typically six basic operations that a GA will utilize during a run. These are: population initialization, fitness evaluation, parent selection, recombination, mutation and survival selection. A GA initializes the population, evaluates the fitness of each individual in this population and then continually loops until some termination condition is met. Termination conditions are typically that a good enough solution is found, that too many generations have been iterated over, or that the fitness has not increased in some given amount of generations. Each loop consists of selecting the parents that will breed in the population, recombining these parents and producing offspring, sometimes mutating the offspring, evaluating the offspring and finally selecting what individuals in the population will be removed. Figure II.7 describes the work flow of a GA.

Population At the start of a GA run the population needs to be initialized. A population is the set of all individuals currently evaluated in the GA, every individual is represented by chromosomes coded in a determined length of sequence of digits, letters or other showing ways. A population has a certain size, a larger size has the drawback of more evaluation time, a smaller size has the drawback of a lacking diversity. Genetic Algorithm works on two types of spaces alternatively, coding space (genotype) and solution space (phenotype) [44]. The phenotype describes the outward appearance of an individual. A transformation exists between genotype and phenotype, also called mapping, which uses the genotypic information to construct the phenotype. A chromosome refers to a string of certain length where all the genetic information of an individual is stored. Each chromosome consists of many alleles. Alleles are the smallest information units in a chromosome as illustrated in Figure II.9. In genetic algorithm, an encoding function is use to represent mapping of the object variables to a string code and mapping of string code to its object variable is achieve through decoding function [44] as shown in Figure II.8

Fitness In order to represent the quality of the solution another term is borrowed from biological evolution, fitness, which describes how well an individual solves the underlying problem. Fitness is calculated according to some objective function for the problem, solutions that solve this problem better typically get a better fitness. Solutions that are valid and conform to the constraints of the problem will typically have a better fitness than those who do not.

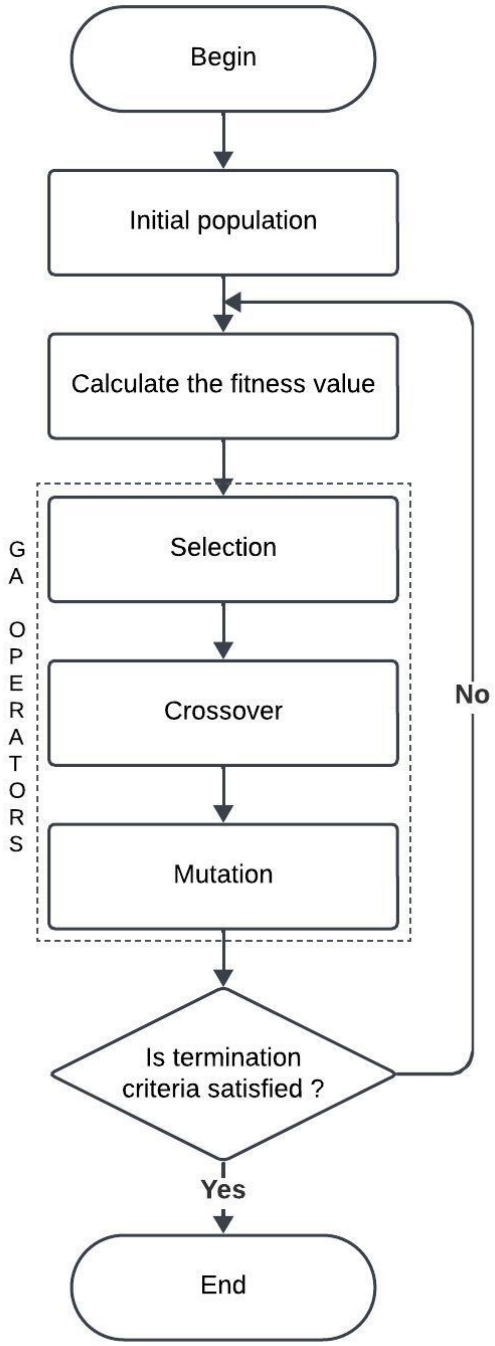


Figure II.7: Genetic Algorithm Flowchart

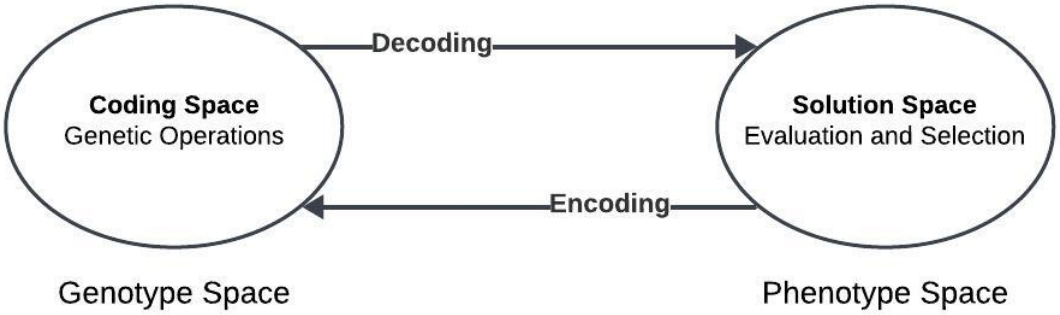


Figure II.8: Encoding-Decoding method

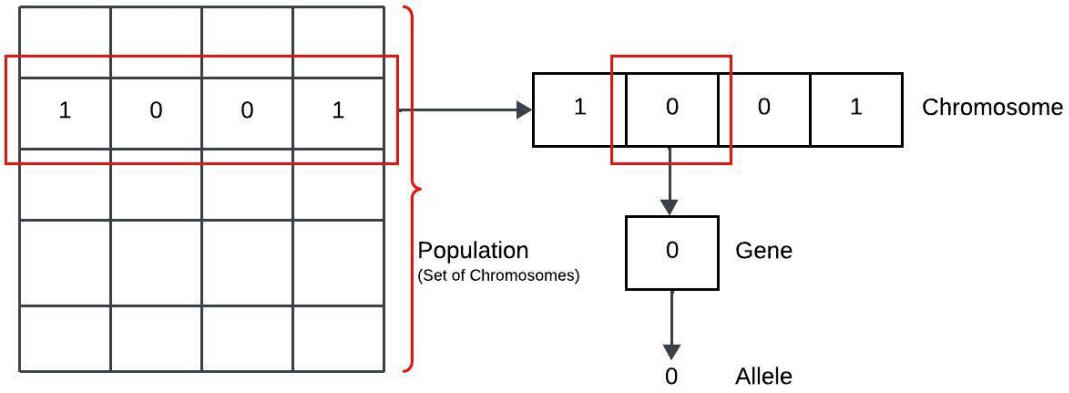


Figure II.9: Genetic algorithm population

Selection After calculating the evaluation of every chromosome, we should choose the same number of chromosomes as in the previous population to create the next generation. This selection takes place in accordance with the natural selection rule that the better parents will have better offspring. Parents are chosen according to their fitness. Thus the chromosomes which have highest evaluation should have the most of the chances to be selected to create new offspring. The selection can be done by many methods, such as roulette wheel selection, Boltzmann selection, tournament selection, rank selection and some others as shown in Figure II.10 [16, 37].

1. **Roulette Wheel Selection**: Fitness proportionate selection, introduced with the development of genetic algorithms, is described in detail by Khalid Jebari et al [37]. In genetic algorithms, roulette wheel selection is a mecha-

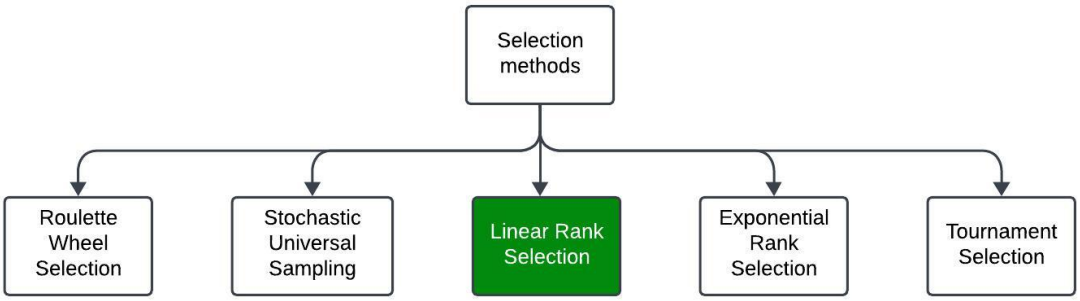


Figure II.10: Selection operator methods

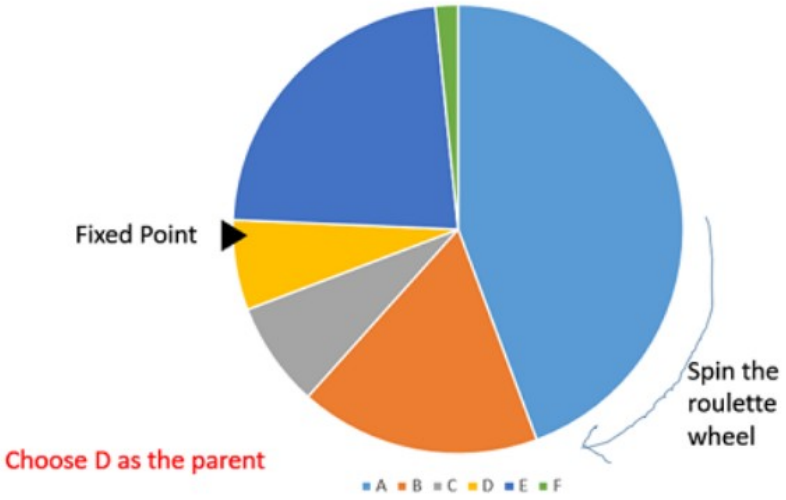
nism used to select individuals from a population for reproduction based on their fitness. It mimics the process of spinning a roulette wheel, where each individual's likelihood of being selected is proportional to its fitness score relative to the total fitness of the population. How it works is that for a set of N elements, each with a fitness F_0F_n, it finds the sum of the fitness for each element in the set and gives each element a chance to be selected with the individual fitness over the sum of the fitness. In mathematical notation, the chance, C, that any element X with fitness F_x would have to be chosen is:

$$C = \frac{F_x}{\sum_{i=1}^{n} F_i} \tag{II.4}$$

To select n parents from the population, the roulette wheel is spun n times. Each spin corresponds to selecting one parent. As mentioned before, the probability of selecting each individual as a parent is determined by their fitness relative to the total fitness of the population, as illustrated in Figure II.11.

2. **Stochastic Universal Sampling**: The SUS selection method [37], a variant of the Roulette wheel selection (RWS), aims to mitigate the risk of premature convergence. Unlike traditional Roulette wheel selection, which typically selects only one fixed point, SUS incorporates multiple fixed points (usually as many as desired number of parents) as illustrated in Figure II.12. Consequently, all parents are selected in a single spin of the wheel. Moreover, this approach promotes the selection of highly fit individuals at least once.

3. **Linear Rank Selection**: The LRS [37] is also a variant of RWS that tries to overcome the drawback of premature convergence of the GA to a local optimum. It is based on the rank of individuals rather than on their fitness. The rank n is accorded to the best individual whilst the worst individual gets the rank 1. Thus, based on its rank, each individual i has the probability of



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Figure II.11: Roulette Wheel Selection [70].

being selected given by the expression

$$p(i) = \frac{\text{rank}(i)}{n \times (n-1)} \tag{II.5}$$

4. **Exponential Rank Selection**: The ERS [37] is based on the same principle as LRS, but it differs from LRS by the probability of selecting each individual. For ERS, this probability is given by the expression:

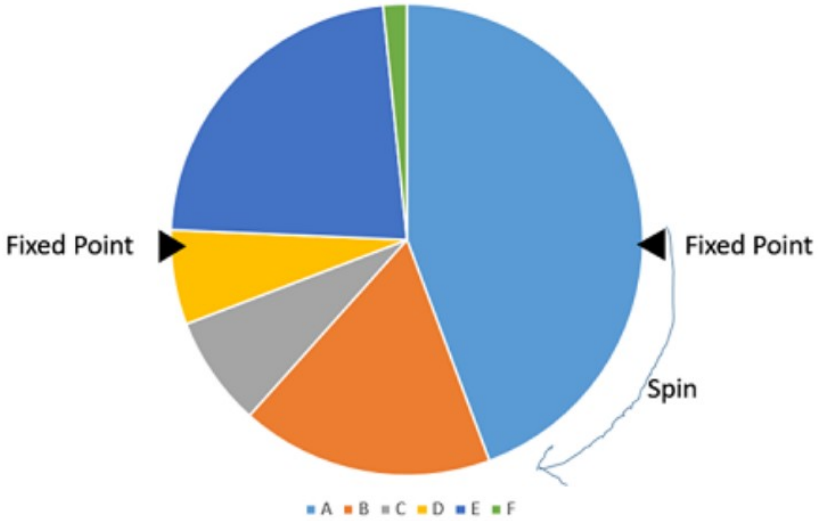
$$p(i) = 1.0 * \exp\left(\frac{-\text{range}(i)}{C}\right) \tag{II.6}$$

with

$$C = \frac{n * 2 * (n-1)}{6 * (n-1) + n} \tag{II.7}$$

5. **Tournament Selection**: Its principle consists in randomly selecting a set of k individuals [70]. These individuals are then ranked according to their relative fitness and the fittest individual is selected for reproduction. The whole process is repeated n times (number of parents) for the entire population as illustrated in Figure II.13. Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.

Crossover Crossover plays a crucial role in genetic algorithms by facilitating the exchange of genetic information between parent individuals to create offspring.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Figure II.12: Stochastic Universal Sampling [70].

After the selection phase in GA, which identifies promising individuals as parents, crossover allows the recombination of their genetic material to generate new solutions. This process aims to diversify the genetic pool and potentially produce offspring with improved characteristics inherited from the parents. The crossover methods are of many types as shown in Figure II.14

1. **One Point Crossover**: One-point crossover [48] operator randomly selects one crossover point and then copies everything before this point from the first parent and then everything after the crossover point from the second parent as shown in Figure II.15.

2. **Two Point Crossover**: The two-point crossover [28] operator operates similarly to the one-point crossover but involves two crossover points instead of one. In this method, two crossover points are randomly selected along the chromosome. Then, the genetic material between the two crossover points is swapped between the parent chromosomes as shown in Figure II.16. This process results in offspring that inherit genetic material from both parents, promoting genetic diversity and potentially creating novel solutions. The two-point crossover operator, while effective, can be scaled to accommodate more crossover points, known as n-point crossover. This allows for more extensive genetic exchange between parent chromosomes, potentially leading to further diversification and exploration of the solution space.

3. **Uniform Crossover**: The operator decides which parent will contribute how the gene value in the offspring chromosomes. The crossover operator allows

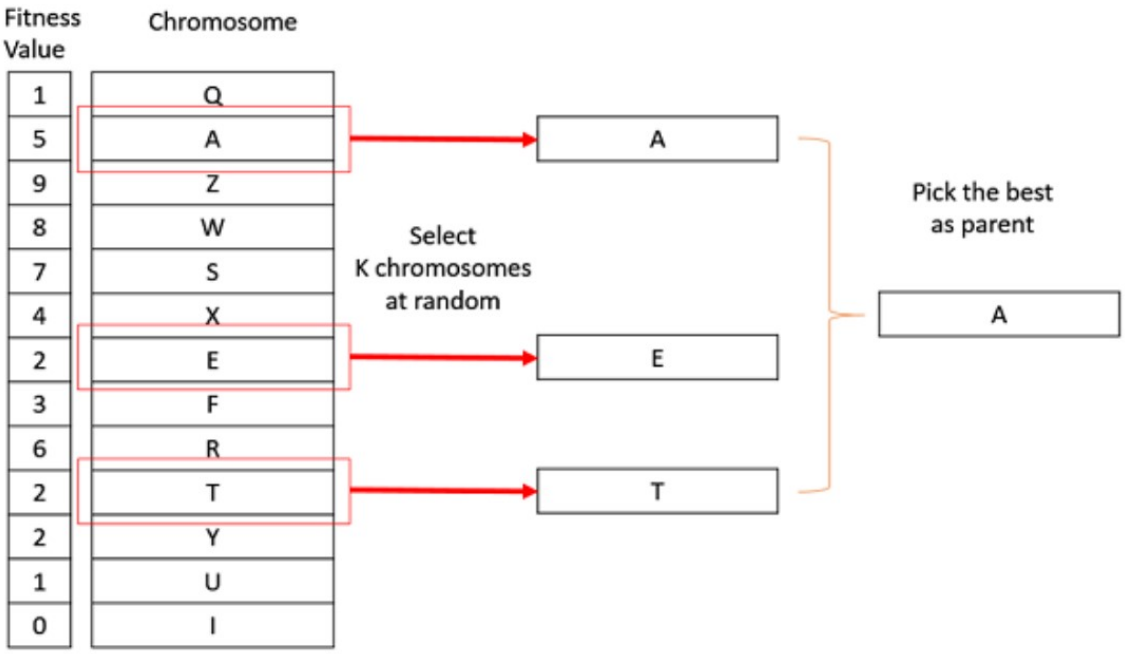


Figure II.13: Tournament Selection [70].

the parent chromosomes to be mixed at the gene level rather than the segment level [76, 48].

Consider the two following parents in the following figure, if the crossover rate is 0.5%, then half of the genes in the offspring will come from parent1 and half from parent2 as shown in Figure II.17.

4. **Arithmetic Crossover**: The arithmetic crossover [48] operator linearly combines two parent chromosome vectors to produce two new offspring according to the equations:

$$\begin{aligned} offspring1 &= w \times parent1 + (1 - w) \times parent2 \\ offspring2 &= (1 - w) \times parent1 + w \times parent2 \end{aligned} \tag{II.8}$$

Where w is a random weighting factor chosen before each crossover operation.

5. **Heuristic Crossover**: The heuristic normal distribution crossover $HNDX$ [76, 48] emphasizes the fitness values of the two parent chromosomes to determine the direction of the search. This operator generates two potential offspring, leading to a better search direction. The key idea behind $HNDX$ lies in ensuring that the offspring produced by the crossover are in the vicinity of the superior parent or in the direction of the search. By leveraging the principles

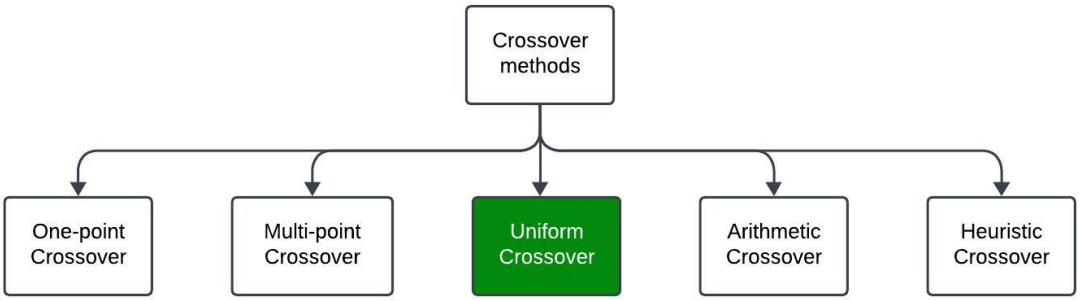


Figure II.14: Crossover methods

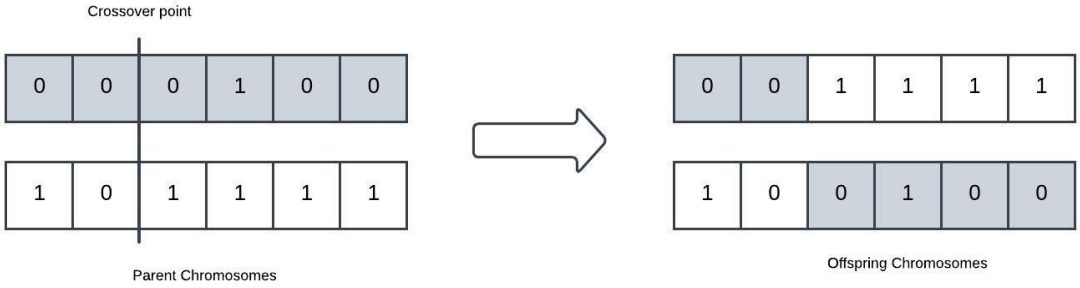


Figure II.15: One Point Crossover

of normal distribution, where a generated random number has a higher probability when closer to its mean, μ, the $HNDX$ operator strategically places the offspring. In the crossover cycle, if the better parent among the two is considered as the mean μ of the normal distribution, the resulting offspring are guaranteed to be in the vicinity of the superior parent according to the normal distribution. The offspring are created according to the equations:

$$\begin{aligned} offspring1 = bestparent + r \times (bestparent - worstparent) \\ offspring2 = bestparent \end{aligned} \tag{II.9}$$

Here, r represents a randomly generated factor between 0 and 1.

Mutation A crucial step in genetic algorithms, occurring after the crossover phase. It introduces random changes to individual chromosomes, typically guided by a user-defined mutation probability, often set at a low value such as 0.01. By altering one or more gene values within a chromosome, mutation injects diversity into the population, potentially leading to the discovery of novel solutions and the escape of

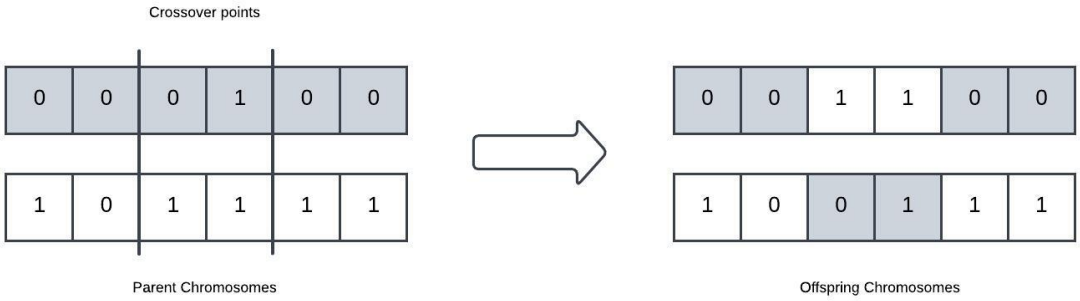


Figure II.16: Two Point Crossover

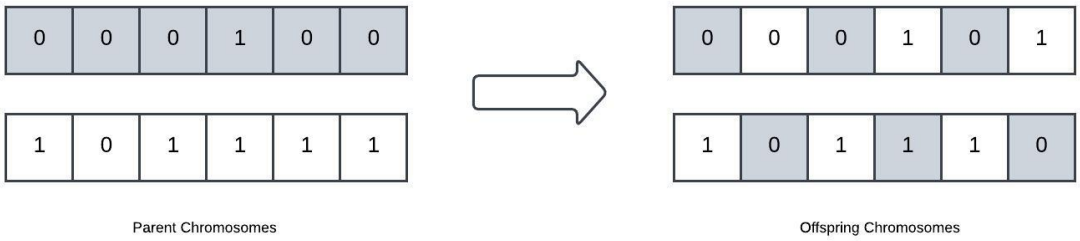


Figure II.17: Uniform Crossover

local optimum. These random changes can introduce entirely new genetic material into the gene pool, expanding the search space and enabling the genetic algorithm to explore previously uncharted territories, potentially improving the overall quality of solutions. The mutation operators are of many types as shown in figure II.18:

1. **Bit Flip Mutation**: The Bit flip Mutation operator simply inverts the value of the chosen gene, i.e., 0 flips to 1 and 1 flips to 0.
2. **Binary Mutation**: Also known as Boundary Mutation, this method replaces the value of the chosen gene with either the upper or lower bound specific to that gene. This mutation method is applicable only to integer and float genes.
3. **Non-uniform Mutation**: This method adjusts the mutation probability as the generation progresses, ensuring that the mutation magnitude decreases over time. This mutation technique is suitable for integer and float genes.
4. **Uniform Mutation**: This method involves replacing the chosen gene's value with a uniformly random value within the user-defined upper and lower bounds for that gene. This mutation approach is applicable only to integer and float genes.

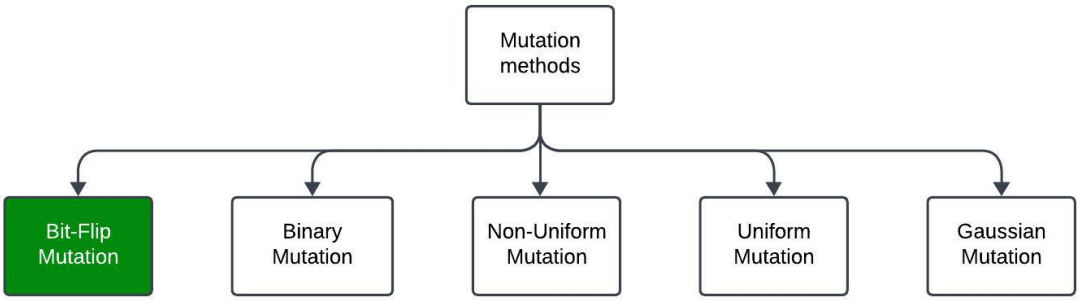


Figure II.18: Mutation methods

5. **Gaussian Mutation**: This method adds a unit Gaussian distributed random value to the selected gene. The resulting gene value is constrained within the specified upper and lower bounds for that gene. Gaussian Mutation is utilized to introduce variability in the population and explore the solution space effectively.

Termination The termination condition defines, when the main evolutionary loop terminates. Often, the Genetic Algorithm runs for a predefined number of generations. This can be reasonable in various experimental settings. Time and cost of fitness function evaluations may restrict the length of the optimization process. A further useful termination condition is convergence of the optimization process. When approximating the optimum, the progress of fitness function improvements may decrease significantly. If no significant process is observed, the evolutionary process stops. For example, when approximating the optima of continuous optimization problems, the definition of stagnation as repeated fitness difference lower than 10^{-8} in multiple successive generations is reasonable.

Summary Genetic Algorithms (GAs) are successful optimization approaches that allows optimization in difficult solutions spaces. They excel in scenarios where derivatives and the fitness landscape exhibits regions of poor conditioning, meaning areas where relationship between the input variable and the objective funciton is complex and unpredictable. In this section, we present an overview of the fundamentals of Genetic algorithms. These Evolutionary algorithms operate on population of candidate solutions, iteratively refining them towards a optimal solutions. Genetic operators, such as crossover and mutation, play pivotal roles in altering the solutions within the population. Crossover operators combine the genetic information from multiple solutions while mutation introduces randomness to diversify the search process. It is essential for mutation to possess characteristics such as scala-

bility, absence of drift, and comprehensive coverage of the solution space. Prior to evaluation against the fitness function, the genotype or chromosome representation of a solution must be translated into its corresponding phenotype, i.e., the actual solution. Selection mechanisms identify the most promising solutions within a population, which serve as parents for the subsequent generation. Equipped with these foundational concepts, we are poised to implement basic Genetic Algorithms. In the next section, we present another type of meta-heuristic, the Variable Neighborhood Search meta-heuristic, which offers an alternative approach to solve optimization problems.

Variable Neighborhood Search (VNS)

VNS [32] is a meta-heuristic that exhibits systematic change in the neighborhood during the search process. The initial solution is changed each time during the local search until a local optimum is reached. VNS is based on three major principles:

- A local optimal solution of one neighborhood structure is not necessary for that of another neighborhood structure;
- A global optimal solution is a local minimum with respect to all neighborhood structures;
- Local optimal solutions with respect to different neighborhoods are relatively close to each other.

Let us assume there are k neighbors structures N_k, $k = 1, ..., k_{max}$. The process starts with the initial solution, we obtain the next solution, from the neighborhood $N(s)$. Performing local changes in the neighborhood, we can obtain a best solution from $N(s)$. Researchers in [32], described VNS that performs several local searches with different neighborhoods until a local optimum is obtained. Below is the general working pseudo code for VNS:

Algorithm 3 Variable Neighborhood Search (VNS)

Input: Set of neighborhood structures N_k for $k = 1, ..., k_{\max}$, initial solution s_0
Output: Best found solution s
Initialization: Select the set of neighborhood structures N_k for $k = 1, ..., k_{\max}$ that will be used in the search.
Generate a random initial solution s.
Set $k = 1$.
while $k \leq k_{\max}$ **do**
  Shaking: Generate a point $\dot{s}$ randomly from $N_k(s)$.
  Local Search: Implement local search method to obtain local optimum $\ddot{s}$ from $\dot{s}$.
  if $f(\ddot{s}) < f(\dot{s})$ **then**
    Set $s = \ddot{s}$ and $k = 1$.
  else
    $k = k + 1$.
  end if
end while
Stop.

The VNS algorithm, depicted in the provided pseudo-code 3, orchestrates a dynamic exploration of solution space, aiming to find optimal or near-optimal solutions to combinatorial optimization problems. The algorithm is structured as follows:

1. The algorithm initiates by selecting a set of neighborhood structures, denoted as N_k, reflecting different levels of solution perturbation.

2. Subsequently, a random initial solution, s, is generated to kick start the search process.

3. The algorithm sets k to 1, initializing the exploration index.

4. The execution progresses with the shaking phase, where a new initial solution, $\dot{s}$, is generated by applying a chosen neighborhood structure $N_k(s)$. This phase aims to inject diversity into the search process.

5. Following the shaking phase, the algorithm undertakes a local search, leveraging various neighborhood structures to refine the solution $\dot{s}$ into a potentially better solution, $\ddot{s}$.

6. If the new solution $\ddot{s}$ proves superior to the current solution s, it replaces s as the new solution, and k is reset to 1. Otherwise, k is incremented and the old solution will remain as s.

7. The algorithm iterates through these steps until a predefined stopping criterion is met, signifying convergence or exhaustion of computational resources.

VNS is a simple and effective metaheuristic approach to solve difficult optimisation problems. The idea of using more than one neighborhood in the search process has gained interest among various researchers and has been used in a variety of applications. Depending on the complexity of the problem and adaptability nature, VNS has led to several variants of VNS. In the following sections we will discuss some of the most often used VNS variants that have distinctive characteristics, some of these variants are:

- Variable Neighborhood Descent
- Reduced Variable Neighborhood Search
- Basic Variable Neighborhood Search
- General Variable Neighborhood Search
- Skewed Variable Neighborhood Search

A. Variable Neighborhood Descent (VND)

VND is a variant of VNS that explores the complete neighborhood and makes changes in a deterministic manner. Due to this process, VND results in large computation time. A frequent implementation consists of ranking moves by order of complexity of their application. This is often the same as by size of their neighborhood Ni(s), and returning to the first one each time a direction of descent is found, and a step made in that direction. Alternatively, a process of experiments is done in order to find the optimal neighborhoods sequence to implement. The pseudo-code of the VND is given below:

The VND algorithm, depicted in the provided pseudo-code 4 and Figure II.20. At its core, VND operates by iteratively exploring various neighborhoods around a given solution to identify improvements. The process begins with selecting an initial solution, denoted as s, and an initial neighborhood structure, represented as N_1. The algorithm then iterates through different neighborhoods, indexed by l, seeking an improved solution. Within each iteration, a candidate solution s' is generated within the current neighborhood $N_l(s)$. If this candidate solution s' offers a better objective function value than the current solution s, it replaces s as the new solution. Otherwise, the algorithm moves to the next neighborhood structure, incrementing l by one. This process continues until a stopping criterion is met, such as reaching a predefined number of iterations or achieving a satisfactory solution quality. It's

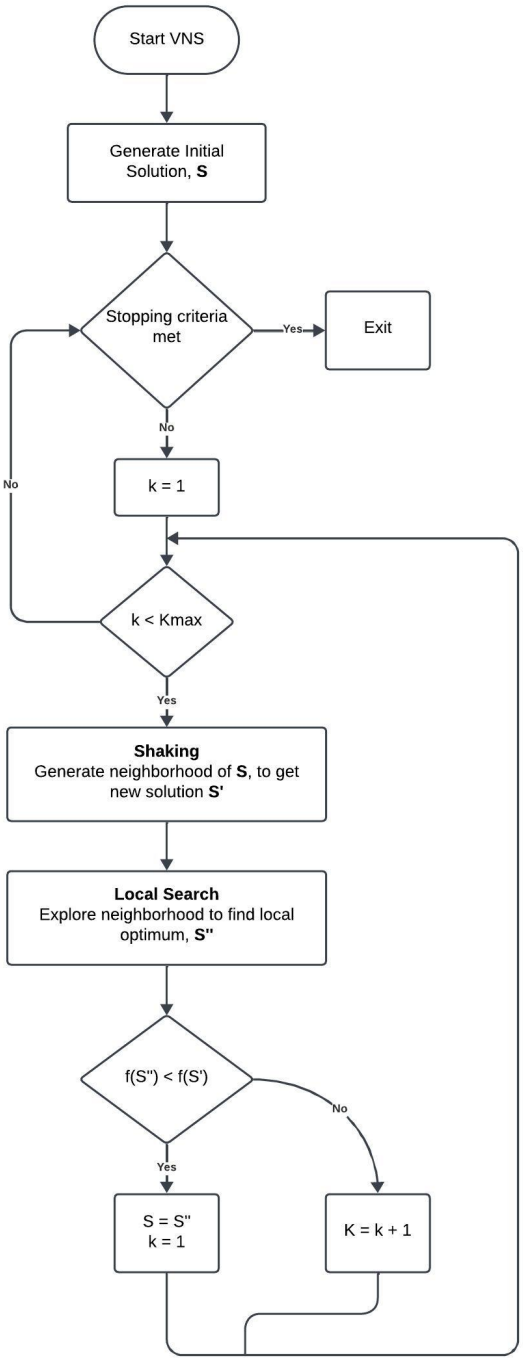


Figure II.19: VNS Flowchart

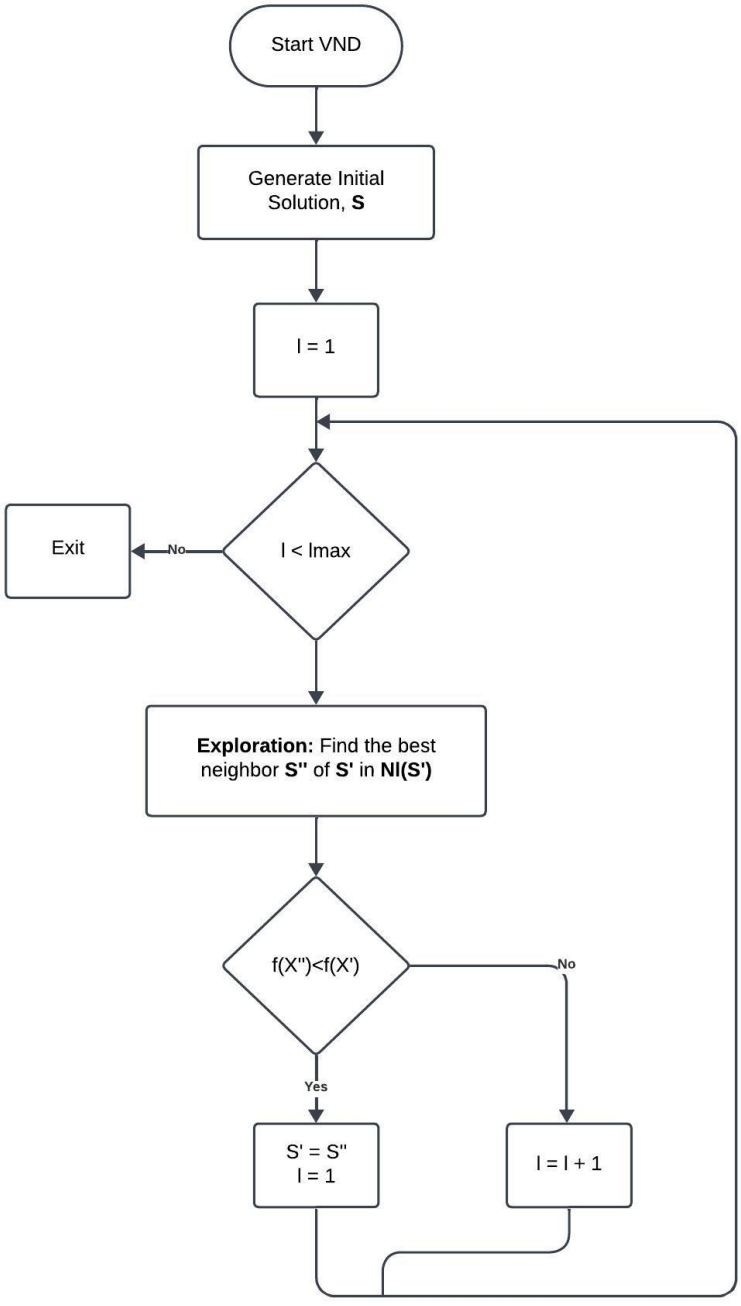


Figure II.20: VND Flowchart

Algorithm 4 Variable Neighborhood Descent (VND)

Input: Set of neighborhood structures N_k for $k = 1, ..., k_{\max}$, initial solution s_0
Output: Best found solution s
Select initial neighborhood N_1
$l \leftarrow 1$
repeat
  $s' \leftarrow s$
  for each neighborhood N_l **do**
    Find best solution s' in $N_l(s)$
    if $f(s') < f(s)$ **then**
      $s \leftarrow s'$
      $l \leftarrow 1$
      Break
    end if
  end for
  $l \leftarrow l + 1$
until Stopping criteria met

noteworthy that the shaking operation, as depicted in the original pseudocode is deleted. The VND algorithm's effectiveness lies in its ability to systematically explore diverse neighborhoods, enabling it to escape local optima and converge toward better solutions in optimization problems. The computational time in VND is very high, and for that reason, it is used in larger size combinatorial problems where the application uses more computational time. Rong and Kendall [57], investigated VND for the delay-constrained least cost (DCLC) multicast routing problem, and showed that VND outperforms other existing algorithms. Some the variants of the VND algorithm are [50]:

1. **Basic Sequential VND**
   - In Sequential VND, the algorithm explores neighborhoods in a sequential order, moving from one neighborhood to the next systematically.
   - It follows a predetermined sequence of neighborhood structures during the optimization process.
   - The search continues within the same neighborhood until improvements are no longer achieved before moving to the next neighborhood.
2. **Cyclic VND**

- Cyclic VND alternates between different neighborhood structures regardless of the achieved improvements, creating a cyclical pattern of neighborhood exploration.
- It does not strictly adhere to a predefined order of neighborhoods, allowing for a more dynamic exploration of the solution space.
- The algorithm changes neighborhoods in a cyclical fashion, providing a different approach to neighborhood exploration compared to Sequential VND.

3. **Pipe VND**

- Pipe VND continues the search within the same neighborhood as improvements occur, focusing on refining the current solution within a specific neighborhood.
- It differs from Sequential and Cyclic VND by maintaining the search within the current neighborhood until further improvements are no longer possible.
- Pipe VND offers a more focused approach to optimization by concentrating on enhancing the solution within a single neighborhood before transitioning to a new one.

4. **Union-VND**

- Also called Multiple Neighborhood Search
- at each iteration, U-VND explores the single neighborhood, obtained as the union of all predefined l_{max} neighborhoods, trying to improve the current incumbent solution [50].

B. Reduced Variable Neighborhood Search (RVNS)

Reduced Variable Neighborhood Search (RVNS) is another variant of VNS. It is mostly based on the third principal of VNS, a global optimum is the best solution across all neighborhoods. Hence, in a specific neighborhood a solution is randomly selected. This random selection constitutes a stochastic search, and it does not use a local search to improve the solution. Below is the pseudo code for RVNS:

Algorithm 5 RVNS

Input: Set of neighborhood structures N_k for $k = 1, ..., k_{\max}$, initial solution s_0

Output: Best found solution s
Initialization: Select the set of neighborhood structures N_k for $k = 1, ..., k_{\max}$ that will be used in the search.
Set $k = 1$.
while $k \leq k_{\max}$ **do**
  Shaking: Generate a point $\dot{s}$ randomly from $N_k(s)$.
  if $f(\ddot{s}) < f(\dot{s})$ **then**
    Set $s = \ddot{s}$ and $k = 1$.
  else
    $k = k + 1$.
  end if
end while
Stop.

In the RVNS algorithm 5, a set of neighborhoods $N_1(s), N_2(s), \ldots, N_{k_{\max}}(s)$ is defined around the current point s. These neighborhoods are typically arranged in a nested fashion, with each one containing the previous. The algorithm randomly selects a point within the first neighborhood. If this point yields a better solution than the current one (i.e., $f(\dot{s}) < f(s)$), the search continues within that neighborhood ($s = \dot{s}$). Otherwise, the algorithm progresses to the next neighborhood. This process repeats for all neighborhoods until a stopping condition is met, often determined by a maximum computing time or a maximum number of iterations. The nested structure of neighborhoods ensures that the algorithm explores smaller, closer neighborhoods more thoroughly before moving on to larger ones, thus facilitating an efficient search process [32].

C. Basic Variable Neighborhood Search (BVNS)

Th BVNS in another variant of VND proposed in [32], it is a hybrid of VND and RVNS. Thus, the BVNS uses a process to find the next optimal solution from the most fitting neighborhood structure, then the solution is further refined and improved by using a local search technique. This improved solution is the current solution from the neighborhood in the iteration. The BVNS provides a good solution and saves computational time. Algorithm 6 shows how the BVNS works.

We begin by selecting a series of neighborhood structures that define neighborhoods around any given point s in the solution space. Next, we employ local search techniques to find a local optimum s^*. Then, within the first neighborhood $N_1(s)$

Algorithm 6 BVNS

Input: Select the set of neighborhood structures N_k for $k = 1, ..., k_{\max}$ that will be used in the search, Initial Solution s_0
Output: Best found solution s
Find an initial solution s;
Choose a stopping condition;
repeat
  Shake: Generate a point $\dot{s}$ at random from the kth neighborhood of s ($\dot{s} \in N_k(s)$);
  Local search: Improve $\dot{s}$ using a local search method;
  if $\dot{s}$ is better than the incumbent **then**
    Set $s = \dot{s}$;
  else
    Increment k ($k = k + 1$);
  end if
until stopping condition is met

of s, we randomly select a point $\dot{s}$ and perform a local search descent from $\dot{s}$. This process is repeated iteratively. This leads to a new local minimum $\ddot{s}$. At this point, three outcomes are possible [32]:

- If $\ddot{s} = s$, indicating that the search has returned to the same local minimum, the procedure continues with the next neighborhood $N_k(s)$, where $k > 2$.
- If $\ddot{s} \neq s$ but $f(\dot{s}) > f(s)$, suggesting that another local optimum has been discovered but is not superior to the previous best solution (or incumbent), the procedure advances to the next neighborhood.
- If $\ddot{s} \neq s$ and $f(\dot{s}) < f(s)$, indicating that a superior local optimum has been found, the search restarts around $\ddot{s}$, initiating once again with the first neighborhood.

Should the last neighborhood is reached without a solution better than the incumbent being found, the search begins again at the first neighborhood N1(s) until a stopping condition, e.g., a maximum time or maximum number of iterations, or maximum number of iterations since the last improvement, is satisfied.

D. General Variable Neighborhood Search

The subsequent variant of VNS discussed in this section is the General VNS, which merges aspects of VND and RVNS. It begins by employing RVNS to obtain

a solution and subsequently applies local search to enhance it. Below is the general pseudocode 7 for GVNS:

Algorithm 7 General Variable Neighborhood Search (GVNS)

Input: Set of neighborhood structures N_k for $k = 1, ..., k_{\max}$, initial solution s_0
Output: Best found solution s
Initialization: Select the sets of neighborhood structures N_k for $k = 1, \ldots, k_{\max}$ for the shaking phase and N_{local} for local $= 1, \ldots, \text{local}_{\max}$ for the local search; Choose a stopping condition.
repeat
  Set $k \leftarrow 1$.
  repeat
    Shaking:
    Generate a point x' at random from the kth neighborhood $N_k(x)$ of x.
    Local search by VND:
    Set $l \leftarrow 1$.
    repeat
      Exploration of neighborhood: Find the best neighbor x'' of x' in $N_l(x')$.
      Move or not:
      if $f(x'') < f(x')$ **then**
        Set $x' \leftarrow x''$ and local $\leftarrow 1$.
      else
        Set local $\leftarrow$ local $+ 1$.
      end if
    until $l = l_{\max}$
    Move or not:
    if this local optimum is better than the incumbent **then**
      Move there: $x \leftarrow x''$, and continue the search with N_1 ($k \leftarrow 1$).
    else
      Set $k \leftarrow k + 1$.
    end if
  until $k = k_{\max}$
until stopping condition is met

The algorithm begins by initializing sets of neighborhood structures for both the shaking phase and the local search. It then finds an initial solution and improves it using RVNS. A stopping condition is also defined to determine when the algorithm should terminate.

Within each iteration, the algorithm sets up structured sequence of operations.

It initiates by setting the neighborhood index k to 1 and iterates through the subsequent steps until the maximum neighborhood index k_{max} is attained.

Initially, the algorithm executes the shaking phase by generating a random solution x' from the kth neighborhood $N_k(x)$ of the current solution x. It then engages in local search utilizing VND within $N_l(x')$, where N_l denotes the neighborhood with index l in the VND.

During the local search phase, the algorithm explores neighboring solutions within $N_l(x')$ and evaluates the potential transition to a superior solution x''. Upon identifying an improved solution, the algorithm updates the current solution x' and recommences the local search process.

Upon completion of the local search, the algorithm assesses whether the attained local optimum surpasses the incumbent solution. If affirmative, the solution is updated, and the search recommences with the initial neighborhood N_1. Otherwise, the algorithm progresses to the subsequent neighborhood k and repeats the iterative process until k_{max} is reached.

The algorithm iterates until the specified termination condition is satisfied, presenting an effective methodology for exploring diverse neighborhoods and deriving high-quality solutions. Mladenovic and hansen in [32] have implemented GVNS for the travelling salesman's problem, and provided the upper bounds in more than half of the existing benchmark instances. General VNS has been selected as the primary technique utilized in our thesis project, providing a robust framework for addressing the deployment of Relay nodes in fenced areas wireless sensor networks.

E. Skewed Variable Neighborhood Search (SVNS)

The SVNS is motivated by the topology of the search space and is a modified version of BVNS. The basic idea of SVNS is that in the neighborhood change step accept as new incumbent solutions not only improving solutions but in some cases those which are worse than the current incumbent solution. Therefore, in the so called *skewed neighborhood change step*, a trial solution is evaluated taking into account not only the objective values of the trial and the incumbent solution but also the distance between them as shown in algorithm 8 below:

Algorithm 8 Skewed VNS

Input: Set of neighborhood structures N_k for $k = 1, ..., k_{\max}$, initial solution s_0 and its objective function value $f(s_0)$.
Output: Best found solution s
Initialization: Select the set of neighborhood structures N_k for $k = 1, ..., k_{\max}$.
Set $s_{\text{opt}} = s$, $f_{\text{opt}} = f(s)$.
Choose a stopping condition and a parameter value α.
repeat
  Set $k = 1$.
  repeat
    Shaking: Generate a point s' at random from the k-th neighborhood of s.
    Local search: Apply a local search method with s' as the initial solution. Denote the obtained local optimum as s''.
    if $f(s'') < f_{\text{opt}}$ **then**
      Update the best solution: $f_{\text{opt}} = f(s'')$, $s_{\text{opt}} = s''$.
      Restart the search with the first neighborhood: $k = 1$.
    else
      if $f(s'') - \alpha \cdot \rho(s, s'') < f(s)$ **then**
        Set $s = s''$ and $k = 1$.
      else
        Set $k = k + 1$.
      end if
    end if
  until $k = k_{\max}$
until stopping condition is met

The authors in [32], have demonstrated SVNS for the weighted maximum satisfiability of logic problem. They have shown SVNS has performed better than tabu search and VNS for large and medium size problems.

II.3.3 Multi-Objective Optimization Methods

Multi-objective optimization (MOO) tackles the challenge of finding optimal solutions in scenarios where multiple objectives compete and often conflict with each other. Unlike traditional single-objective optimization, where the goal is to optimize a single criterion, MOO addresses situations where several objectives need to be optimized simultaneously. This paradigm shift reflects the complexity of real-world problems, which often involve multiple criteria that must be considered holistically.

In the realm of multi-objective optimization, the decision-making process becomes more intricate as it shifts from seeking a single optimal solution to identify-

ing a diverse set of solutions that represent trade-offs between competing objectives. These trade-off solutions offer decision-makers a spectrum of choices, each reflecting a different compromise between conflicting objectives.

For instance, in our case, we aim to optimize two primary objectives that inherently conflict with each other: **the number of relay nodes (RNs)** deployed in the network site and **the diameter of the network**. The number of relay nodes directly impacts the network's cost and complexity, while the diameter represents the longest path among the shortest paths from all sentinel nodes (SNs) to the sink node. Balancing these objectives requires finding an optimal or near-optimal trade-off that minimizes the number of relay nodes while simultaneously minimizing the network's diameter. Achieving this balance ensures an efficient network topology that minimizes both deployment costs and communication latency.

The pursuit of a set of trade-off optimal solutions reflects the essence of multi-objective optimization. By considering all objectives as equally important, MOO endeavors to provide decision-makers with a comprehensive understanding of the problem landscape. Armed with this knowledge, stakeholders can then apply higher-level qualitative considerations to select the most suitable solution based on their preferences, constraints, and overarching goals.

In essence, multi-objective optimization offers a powerful framework for navigating complex decision spaces, empowering decision-makers to explore diverse solution possibilities and make informed choices that balance competing objectives effectively.

II.3.3.1 Lexicographic method

The **lexicographic method** is a non-aggregated and non-Pareto approach used to handle multi-objective optimization problems. This method prioritizes and processes objectives based on a pre-defined order set by the decision maker. The order reflects the importance of the objectives, and the problem is solved sequentially according to this hierarchy.

In the lexicographic method, each objective function f_i is optimized one at a time, starting with the highest-priority objective. The solution to the first objective becomes the constraint for the next objective in line. This process continues sequentially until all objectives have been addressed, culminating in a solution that represents the global optimum according to the specified priorities.

The key steps of the lexicographic method are as follows:

1. **Order Definition**: The decision maker establishes a hierarchy of objectives based on their relative importance.

2. **Sequential Optimization**: Starting with the highest-priority objective, the method optimizes each objective function in sequence. The solution obtained

from optimizing one objective is used as a constraint for the next objective.

3. **Final Solution**: The process continues until all objectives have been optimized, resulting in a final solution that respects the predefined hierarchy of objectives.

In the work by BENTABET and BENGHELIMA (2019), the lexicographic method was applied to a multi-objective optimization problem involving the deployment of Relay Nodes (RNs) in a wireless sensor network. The objectives included minimizing the deployment cost, maximizing the network coverage, and ensuring energy efficiency. The objectives were prioritized based on their importance to the decision makers. The lexicographic method was used to solve the problem, starting with the most important objective (minimizing deployment cost), followed by maximizing network coverage, and finally ensuring energy efficiency. This sequential approach facilitated the identification of an optimal solution that met all the prioritized objectives effectively [6].

By using the lexicographic method, the authors were able to handle the multi-objective optimization problem efficiently, demonstrating the practical applicability of this method in real-world scenarios where clear priorities can be established among the objectives.

II.3.3.2 Aggregated Methods

Aggregated methods transform the multi-objective problem (MOP) into a single-objective problem. Examples of aggregated methods include:

II.3.3.2.1 Weighted Sum Method The weighted sum method combines multiple objectives into a single objective by assigning weights to each objective. It is a straightforward approach and commonly used in classical optimization methods. When dealing with multiple objectives, the weighted sum method is often the first choice due to its simplicity and widespread application [69]. It is based on adding the weighted objectives to form a single cost function as follows,

$$\min_{k \in Q} \sum_{i=1}^{n} w_i f_i(k) \tag{II.10}$$

where $w_i > 0$ for all $i = 1, \ldots, n$ and $\sum_{i=1}^{n} w_i = 1$. The optimization may also be subjected to various constraints. The objective functions are often normalized in the weighted sum when they have different ranges of values.

A clear advantage of this method is that the single-objective optimization (SOP) methods can be used to produce a unique solution, which also means that the solution can be found with less computational burden. However, the weighting

factors have a huge impact on the solution and they are hard to be selected [61]. It is also worth mentioning that "weights indicate the relative importance of the corresponding objective function but they do not imply priorities" [61].

II.3.3.2.2 Epsilon-Constraint Method The epsilon-constraint method selects one main objective to maximize and treats the other objectives as restrictions. A procedure that overcomes some of the convexity problems of the weighted sum technique is the ϵ-constraint method. This involves minimizing one objective and expressing the other objectives in the form of inequality constraints as shown in the next equation,

$$\begin{aligned} \text{minimize} \quad & f_1(x), \\ \text{subject to} \quad & f_i(x) \leq \epsilon_i, \quad i = 2, \ldots, m, \\ & x \in X, \end{aligned} \tag{II.11}$$

A notable challenge in the ϵ-constraint method lies in selecting appropriate constraints to ensure feasibility. Additionally, relying solely on hard constraints often falls short of accurately expressing the true design objectives. Alternative methods, such as prioritizing objectives to guide the optimization process within specified bounds of acceptance. Nevertheless, articulating these priorities and constraints effectively in the initial stages of optimization remains a significant challenge [12].

II.3.3.2.3 Goal Attainment Method The goal attainment method described by Gembicki [29] includes a set of prior chosen design targets. Each target is associated with an objective. The formulation of the MOP in this case allows the objectives to be under- or over-achieved so that the initial design goals can be chosen to be imprecise by the decision-maker. The relative degree of under- or over-achievement of these targets is determined by a set of weighting factors. A standard optimization problem using this method can be given as follows,

$$\min_{\gamma \in \mathbb{R}^1, k \in Q} \gamma \tag{II.12}$$

such that $f_i(k) - w_i\gamma \leq f_i^*(k)$ subject to other constraints.

In this method, we introduce constraints to ensure that each objective function $f_i(k)$ remains below or equal to its respective goal $f_i^*(k)$, scaled by a factor γ, subject to other constraints. Here, $f_i(k)$ represents the value of the i-th objective function, while w_i denotes the weight assigned to each objective, indicating its relative importance. The decision maker must select this weight vector before applying the multi-objective optimization method. The goals $f_i^*(k)$ represent the desired values that the objective functions should achieve, and γ is an unrestricted scalar [61].

It is important to note that while this method resembles the weighted sum approach, the primary focus remains on determining the appropriate weight vector.

Additionally, each element of the weight vector does not directly correspond to the priority of the respective objective function [61].

II.3.3.3 Pareto Methods

Pareto methods leverage the concept of dominance to identify the Pareto front, representing the set of efficient (or non-dominated) solutions where no objective function can be improved without degrading others.

Theorem II.3.1 (Pareto Dominance) *A vector $x \in D$ is said to dominate another vector $y \in D$ if and only if:*

$$f_i(x) \leq f_i(y), \forall i \in \{1, 2, ..., n\}, \exists j \in \{1, 2, ..., n\} \,/\, f_j(x) < f_j(y)$$

This is denoted as $x \prec y$ and read as: x Pareto dominates y.

Identifying the Pareto front involves evaluating the dominance relationship among solutions to determine a set of non-dominated solutions. These solutions represent the best trade-offs among objectives, offering decision-makers a range of optimal choices to consider based on their preferences and constraints. The Pareto front provides a visual representation of these trade-offs, aiding in the understanding and selection of the most suitable solutions for the given multi-objective optimization problem [6].

II.4 Introduction to Machine Learning

Machine Learning (ML) stands at the forefront of artificial intelligence, revolutionizing the way machines process information and make decisions. Unlike traditional programming paradigms, ML empowers machines to learn from past data or experiences, enabling them to make predictions or classifications about future events without explicit programming. But what exactly constitutes learning in the realm of ML? According to Tom Mitchell, a renowned expert in Computer Science and Machine Learning, a computer program is said to learn from experience E with respect to some task T and performance measure P, if its performance on T improves with experience E [52].

ML tasks are typically defined by how the ML system should process examples, which are collections of features such as pixels in an image, within a dataset. Common tasks include classifications, regressions, transcriptions, machine translation, synthesis, and sampling. Evaluating the performance of an ML algorithm involves designing quantitative measures such as accuracy or error rate to assess its effectiveness.

In this section, we delve into the intricacies of ML, focusing on three main categories of ML algorithms: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. Supervised learning involves learning from labeled data, where the algorithm learns to predict the output from input examples. Unsupervised learning, on the other hand, deals with uncovering hidden patterns or structures in unlabeled data. Lastly, Reinforcement Learning involves learning to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties.

By exploring these distinct paradigms of machine learning, we aim to provide a comprehensive understanding of their principles, applications, and challenges in modern AI systems.

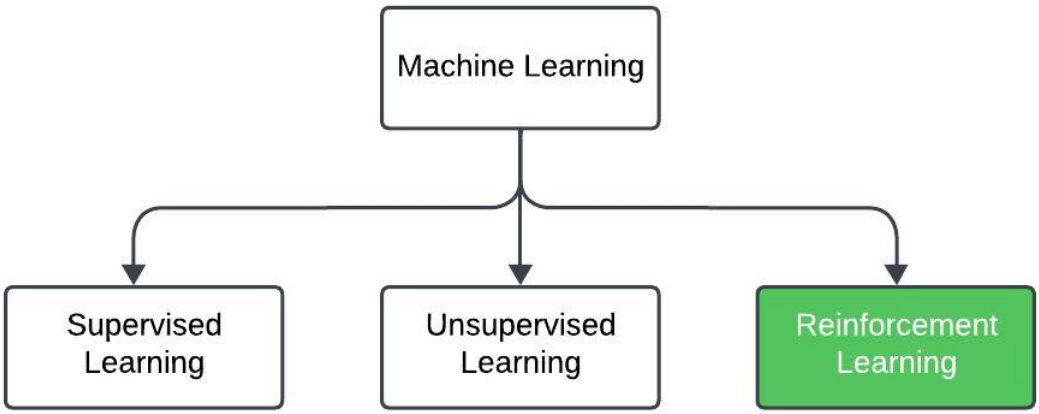


Figure II.21: Machine Learning techniques categories

II.4.1 Supervised Learning

Supervised Learning is the most popular paradigm for performing ML operations. Two kinds of problems that supervised machine learning aims to solve are classification problems and regression problems.

II.4.1.1 Classification

In 2012, Kevin P. Murphy [51] defined the objective of classification as learning a mapping from the input set x to the outputs y, where $y \in \{1, ..., C\}$ and C represents the number of classes as shown in Figure. For binary classification problems ($C = 2$), $y \in \{0, 1\}$, while for multi class classification ($C > 2$), y can take on multiple values. In cases where class labels are not mutually exclusive, it is referred to as multilabel

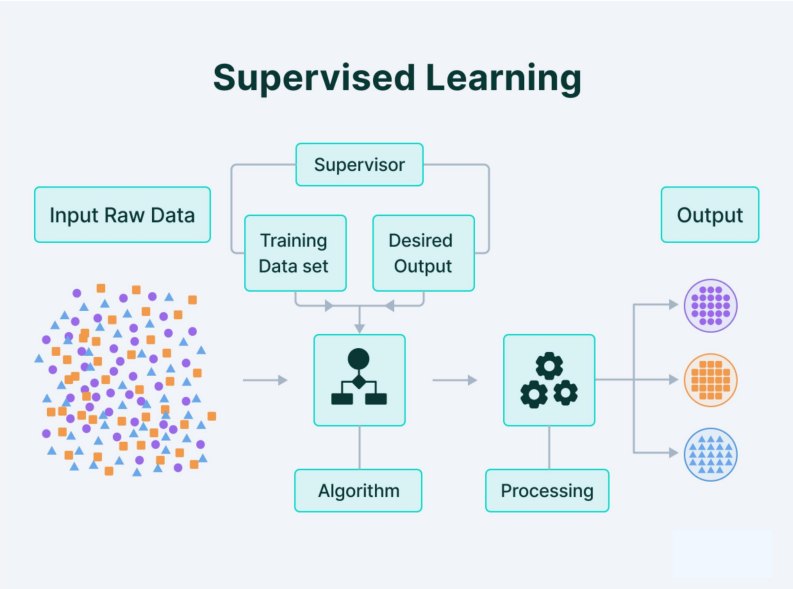


Figure II.22: Supervised Learning process [71].

classification. Let's assume $y = f(x)$ for some unknown function f. The objective of learning is to estimate this function f based on predictions $\hat{y} = \hat{f}(x)$, enabling accurate predictions on new input data. Ambiguous cases may arise, where it's challenging to determine the correct label class for input features. In such scenarios, introducing a probability distribution over possible labels, denoted as $p(y|x, D)$, can be beneficial. This distribution typically represents a vector of length C. Given a probabilistic output, the "true labels" can be computed by:

$$\hat{y} = \arg\max_{y} p(y|x, D) \tag{II.13}$$

This expression identifies the most probable label based on the given input vector x and training set D.

II.4.1.2 Regression

Regression is akin to classification, with the key distinction being the continuous nature of response variables. In regression, the objective is to predict a continuous outcome variable y based on one or more input variables x. Linear regression stands out as the simplest and most widely used technique, assuming a linear relationship between the output and input variables [35].

Several real-world supervised machine learning problems illustrate the utility of regression [35]:

- Predicting the GDP of a country for the next year, based on current economic conditions.
- Forecasting the revenue of a company for the upcoming month, leveraging historical revenue data and other relevant information.
- Estimating the number of views for a particular YouTube video, considering the viewing history of a specific YouTube channel.

II.4.2 Unsupervised Learning

Unsupervised learning forms a crucial component of machine learning, focusing on extracting meaningful patterns and structures from unlabeled data. Unlike supervised learning, where the algorithm learns from labeled examples, unsupervised learning algorithms operate on raw, unlabeled data, making it a versatile and powerful tool in various domains.

The primary objective of unsupervised learning is to uncover inherent structures within the data, such as clusters, relationships, or underlying distributions, without explicit guidance or supervision. This exploration enables the algorithm to identify hidden patterns, anomalies, or similarities that may not be apparent to human observers. Consequently, unsupervised learning plays a pivotal role in exploratory data analysis, feature engineering, and data preprocessing tasks.

Clustering and dimensionality reduction are two fundamental tasks within unsupervised learning. Clustering algorithms group similar data points together based on their intrinsic characteristics, facilitating segmentation and classification of data into meaningful clusters. On the other hand, dimensionality reduction techniques aim to compress high-dimensional data into a lower-dimensional space while preserving essential information, thereby enhancing computational efficiency and interpretability.

Furthermore, unsupervised learning techniques extend beyond traditional data analysis domains and find applications in various fields such as image and speech processing, anomaly detection, recommendation systems, and natural language processing. By autonomously identifying patterns and structures in data, unsupervised learning algorithms pave the way for valuable insights, knowledge discovery, and decision-making support in complex and unstructured datasets.

II.4.2.1 Clustering

Cluster analysis is an exploratory technique whose aim is to identify groups, or clusters, of high density in which observations are more similar to each other than

observations assigned to different clusters. This process requires to quantify the degree of similarity, or dissimilarity, between observations. The results of the analysis is strongly dependent on the kind of the used similarity metric.

For instance, let's consider a scenario where we have extensive data on visitors to our blog. We can employ a clustering algorithm to identify groups of similar visitors. At no point do we explicitly tell the clustering algorithm which group a visitor belongs to; it autonomously discovers these connections without our intervention. For example, the algorithm might observe that 40% of our visitors are male comic book enthusiasts who typically browse our blog in the evenings, while 20% are young science fiction fans who visit the site on weekends, and so forth. If we utilize a hierarchical clustering algorithm, it can further divide each group into smaller subgroups, aiding in targeted messaging for each group.

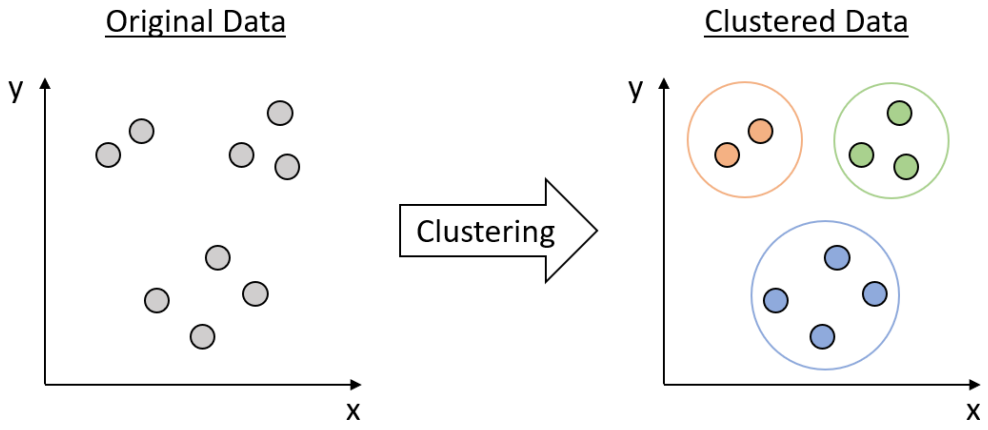


Figure II.23: Graphical representation of how clustering can be used to identify patterns in a data set [5].

II.4.2.2 Dimensionality reduction

Dimensionality reduction plays a pivotal role in unsupervised learning by transforming high-dimensional data into lower-dimensional spaces without compromising essential information. This technique is crucial for simplifying the modeling process, enhancing visualization, and mitigating the curse of dimensionality commonly encountered in machine learning tasks.

The primary goal of dimensionality reduction in unsupervised learning is to reduce the complexity of datasets while retaining critical information. By transforming high-dimensional data into lower-dimensional spaces, dimensionality reduction

facilitates data visualization, model simplification, and improved computational efficiency.

The process of dimensionality reduction involves projecting data onto lower dimensional spaces while preserving essential features. This is achieved by identifying the most relevant dimensions or features that capture the underlying structure of the data. The choice of dimensions or features is critical, as it directly impacts the quality of the reduced data.

One of the key challenges in dimensionality reduction is the sensitivity to outliers. Outliers can significantly impact the results of dimensionality reduction, leading to inaccurate or misleading representations of the data. To address this issue, techniques such as robust statistics and robust dimensionality reduction methods have been developed.

Another challenge is overfitting, which occurs when the reduced data is too complex and fails to generalize well to new data. Overfitting can be mitigated by using regularization techniques, such as L1 and L2 regularization, which help to reduce the complexity of the reduced data.

Dimensionality reduction is a fundamental concept in unsupervised learning, and its applications span various domains. It is used to identify patterns, reduce noise, and improve the interpretability of data. In natural language processing, dimensionality reduction is used to reduce the dimensionality of text data, enabling more efficient processing and analysis. In image analysis, dimensionality reduction is used to reduce the dimensionality of image data, enabling more efficient processing and recognition.

In conclusion, dimensionality reduction is a crucial concept in unsupervised learning that enables the transformation of high dimensional data into lower dimensional spaces while preserving essential information. Its applications are diverse and span various domains, offering insights, simplifying modeling tasks, and paving the way for enhanced data analysis and interpretation.

II.4.3 Reinforcement Learning

Reinforcement learning (RL) is currently experiencing a surge of interest in the research community, largely driven by the recent advancements in deep learning (DL). The emergence of deep reinforcement learning, facilitated by DL techniques, has significantly broadened the scope and applicability of RL methods. Positioned as the third major paradigm of machine learning alongside supervised and unsupervised learning, RL focuses on learning to make decisions through a process of trial and error.

In RL, an agent interacts with its environment, seeking to learn optimal strategies to maximize cumulative rewards. This process mimics the way humans and animals learn from their experiences to shape their behavior. By receiving feedback in the

form of reward signals, the agent learns to discern favorable actions from unfavorable ones [65].

Before delving into the findings of this thesis, it is essential to establish a clear understanding of the current state-of-the-art in RL. This section aims to explore the intricacies of this field, providing insights into its underlying principles and methodologies.

Examples A good way to understand reinforcement learning is to consider some of the examples and possible applications that have guided its development [65]:

- A master chess player makes a move. The choice is informed both by planning—anticipating possible replies and counterreplies—and by immediate, intuitive judgments of the desirability of particular positions.
- A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station. It makes its decision based on the current charge level of its battery and how quickly and easily it has been able to find the recharger in the past. and moves.
- A gazelle calf struggles to its feet minutes after being born. Half an hour later it is running at 20 miles per hour.

II.4.3.1 Elements of Reinforcement Learning

Reinforcement learning offers a computational framework for tackling sequential decision-making problems. Unlike problems that can be solved with a single action, RL addresses scenarios where a sequence of actions is required to achieve optimal outcomes. This section endeavors to elucidate the core principles and concepts of RL, providing readers with a foundational understanding of this research domain.

Beyond the *agent* and the *environment*, one can identify four main sub-elements of a reinforcement learning system: a *policy*, a *reward signal*, a *value function*, and, optionally, a *model* of the environment.

Agent The entity that interacts with the environment, making decisions based on observations of the current state. These decisions, termed actions (denoted as a_t), are the sole means by which the agent can influence the environment. While the agent lacks direct control over the environment, it can modify and shape it through its actions. Typically, the agent operates within a predefined set of actions, known as the action space. Environments may feature discrete action spaces, where only a finite number of moves are available (e.g., $A = \{North, South, East, West\}$ in a two-dimensional maze), or continuous action spaces, where actions are represented

as vectors of real values. This distinction is crucial when selecting an appropriate algorithm, as not all algorithms are compatible with both types of action spaces. Depending on the specific requirements of the problem, it may be necessary to adapt or customize the algorithm accordingly. The sequence of states and actions, collectively referred to as a trajectory (τ), serves to represent an episode within the RL framework [47].

Environment represents all the things that are outside the agent. whenever the agent takes an action, it emits a reward and an observation of the environment [47].

Policy defines the learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. It corresponds to what in psychology would be called a set of stimulus–response rules or associations (provided that stimuli include those that can come from within the animal). In some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior. In general, policies may be stochastic [65].

Reward Signal [65, 47] The rewards in reinforcement learning (RL) play a crucial role in defining the objectives of the learning process. Acting as a feedback signal, rewards guide the agent's behavior by distinguishing between positive and negative actions, driving the agent to reinforce and refine its decision-making strategies. At each time step, the environment provides the agent with a single number, the reward, representing the immediate outcome of its action. Over time, the agent's goal is to maximize the total reward accumulated, shaping its behavior to favor actions that lead to favorable outcomes.

In a biological context, rewards can be likened to the experiences of pleasure or pain, serving as immediate feedback for the agent's actions. The reward received by the agent depends on its current action and the state of the environment, with the agent having no control over the reward process itself. However, the agent can influence the reward signal through its actions, either directly or indirectly by altering the environment's state.

The reward signal serves as the basis for adjusting the agent's policy, guiding its decision-making process. Actions that result in low rewards may prompt the agent to revise its policy to favor alternative actions in similar situations in the future.

Value function While rewards indicate immediate desirability, value functions provide a long-term perspective by estimating the cumulative rewards expected from a given state over the future. Values offer a refined assessment of state desirability, considering the rewards available in subsequent states [65].

Unlike rewards, which are directly provided by the environment, values must be estimated from the agent's observations over its lifetime. Value estimation is a central component of RL algorithms, reflecting the importance of long-term planning and decision-making. Despite rewards being primary in driving behavior, values take precedence in decision-making, guiding the agent towards actions that maximize long-term rewards.

In addition to the temporal aspect of rewards, RL differs from supervised and unsupervised learning in its lack of a supervisor and its reliance on direct experience rather than labeled data. While supervised learning involves learning from labeled data provided by a supervisor, RL relies on observations and rewards for learning. Furthermore, the sequential nature of RL introduces dependencies between actions, making resulting data non-independent and challenging traditional learning paradigms.

Model The fourth and final element of some reinforcement learning systems is a *model* of the environment which is represented formally as[38]:

- a discrete set of environment states, S ;
- a discrete set of agent actions, A ;
- a set of scalar reinforcement signals; typically $0, 1$, or the real numbers.

This concept entails replicating the dynamics of the environment or, more broadly, facilitating predictions about the environment's behavior. For instance, when provided with a specific state and action, the *model* can anticipate the subsequent state and associated reward. These models serve a crucial role in planning, referring to the process of determining a course of action by contemplating potential future scenarios prior to their occurrence. Approaches for addressing reinforcement learning challenges that leverage models and planning are termed *model-based* methods. In contrast, *model-free* methods are characterized as direct trial-and-error learners, essentially representing the antithesis of planning [65].

II.4.3.2 Approaches to Reinforcement Learning

In the realm of reinforcement learning (RL), agents are equipped with algorithms tailored to maximize the rewards obtained from their interactions with the environment. Each RL algorithm possesses its own unique characteristics and is often designed to excel in specific application domains. Recognizing the distinctions among

these algorithmic approaches is essential for determining which one best suits the requirements of a given problem. However, navigating the landscape of RL algorithms can be daunting due to their sheer diversity and complexity.

This section endeavors to shed light on the fundamental differences among RL algorithms, offering insights that are pertinent to the context of this thesis. While the distinctions outlined here are pivotal, it is important to acknowledge that they may not encompass the entirety of the RL algorithm spectrum. Nonetheless, they serve as a valuable framework for understanding the varied approaches employed in reinforcement learning.

II.4.3.2.1 Reinforcement Learning Framework

In Reinforcement Learning (RL), the agent interacts with a Markov Decision Process (MDP) (S, A, γ, P, r), where S denotes the fully-observed state space of the environment and the agent, A denotes the action space, γ denotes the discount factor, $P = \{p(s_0), p(s_{t+1}|s_t, a_t)\}$ denotes the initial state distribution and transition dynamics of the environment, and $r(s, a)$ denotes the reward function. The reinforcement signal, or reward feedback, plays a crucial role in the agent's learning process. As depicted in Figure II.24, at each time step t, the agent observes the environment's state $S(t)$, selects an action $A(t)$ according to its policy $\pi(a_t|s_t)$, and receives a reward $r_t = r(s_t, a_t)$. Leveraging the accumulated expected reward and current environmental state, the agent endeavors to refine its actions to learn the optimal policy π^* and to maximize the γ-discounted cumulative future return.

RL finds applications across various domains including robotics, aircraft control, self-driving cars, and business strategy planning. Initially conceived for single-agent scenarios, RL aims to devise an optimal policy tailored to maximize the agent's expected reward, with the policy's efficacy contingent on the characteristics of the environment.

Learning Settings

Online and Offline The learning setting in Reinforcement Learning could be *online* or *offline*. In the first case, the learning process is done in parallel or concurrently while the agent continues to gather new information to use, while the second one progresses toward learning using limited data. Generalisation becomes a critical problem in the latter approach because the agent is not able to interact anymore with the environment [47].

In the context of this thesis, what matters is *offline learning*: the learning phase occurs concurrently with the agent's interaction with the environment, allowing for continuous adaptation as new information becomes available with each subsequent

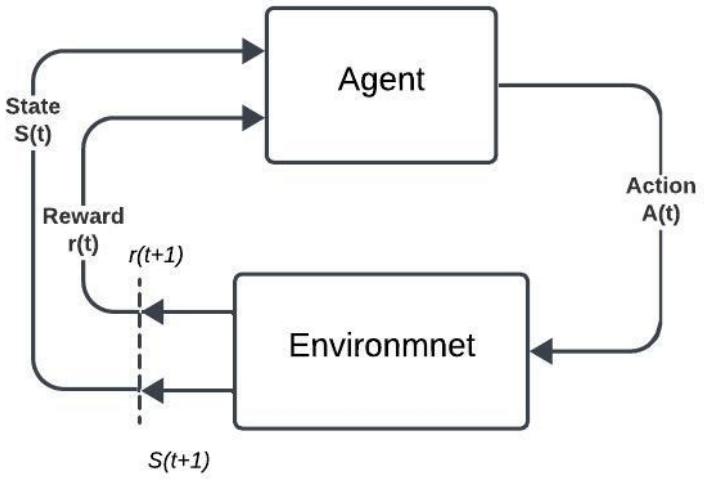


Figure II.24: Reinforcement Learning Framework

learning episode.

On-policy and Off-policy Another important difference in Reinforcement Learning algorithms consists of the distinctive usage of the policy to learn. *On-policy* algorithms heavily rely on training data collected according to the current policy. They are specifically designed to utilize data obtained from the most recent policy iteration. Conversely, off-policy methods have the flexibility to utilize alternative sources of valuable data for learning, beyond direct experience. This capability enables the agent to leverage large experience buffers containing data from past episodes [47].

In the context of this thesis, Our RL framework is an *on-policy* method. Thus, it relies on information and experience sampled according to the current policy iteration for learning.

II.4.3.3 Model-free and Model-Based Reinforcement Learning

This part describes three approaches in RL based on being model-based or model-free, where the model-free includes value-based and policy-based algorithms as shown in Figure II.25. For the clarify of presentation, we explicitly separate model-based from model-free in the discussion here;

II.4.3.3.1 Model-Based approach

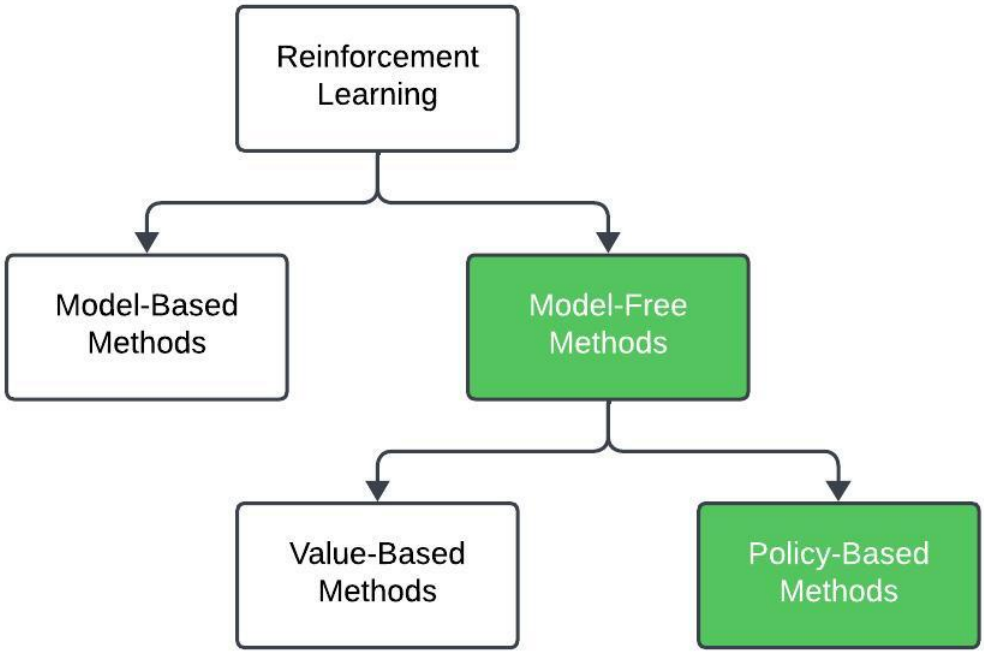


Figure II.25: Reinforcement Learning approaches

Model-based algorithms, often referred to as optimal control within the domain of control theory, comprise a collection of techniques that leverage a learned or existing approximate model of a Markov Decision Process (MDP) to determine the optimal policy. Unlike model-free approaches, which rely on direct interaction with the environment to gather samples, model-based algorithms utilize the MDP model to simulate possible scenarios and derive the best course of action without the need for real-world data or interactions with the real environment as depicted in Figure II.26.

There are two primary principles to model-based learning. The first one implies to assemble a model starting from prior knowledge and to exploit it to calculate the policy and the value-function, while the second one is to infer the model from the environment by sampling experience. The central drawback of the first technique is that prior knowledge could be not as accurate as expected, leading to sub-optimal results. Consequently, the preferred way to learn is the second one.

II.4.3.3.2 Model-free approach

In practical scenarios, having a high level of understanding of the model is often unattainable, leaving agents to navigate with limited insight into the workings of

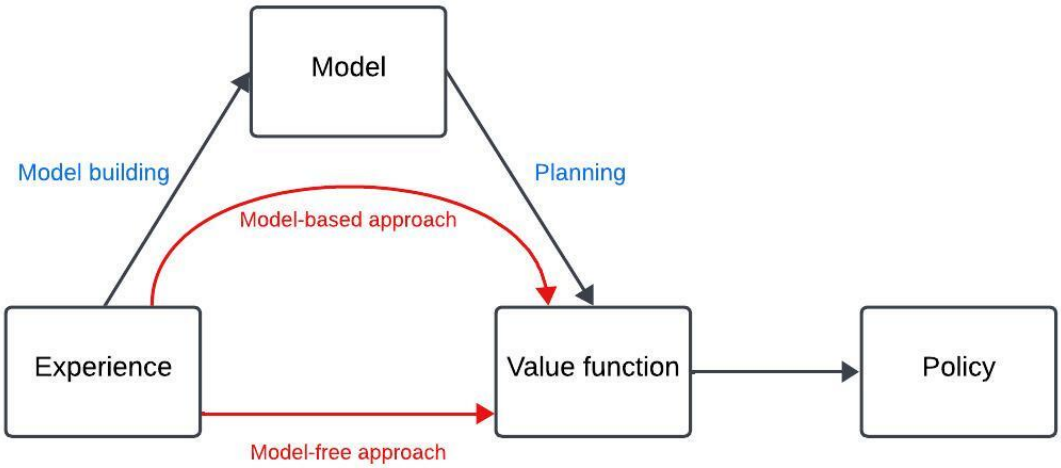


Figure II.26: Difference between Model-free and Model-based RL algorithms

the world. In such instances, the reliance shifts to model-free methods, built on the premise of lacking prior information regarding state transitions and rewards. This section aims to offer a concise overview of two model-free strategies for prediction and control: *Monte Carlo* (MC) methods and *Exploration/Exploitation* methods.

Monte Carlo learning

Monte Carlo methods can learn from episodes of experience using the simple idea that averaging sample returns provide the value. This lead to main caveat of these methods: they work only with episodic MDPs because the episode has to terminate before it is possible to calculate any returns. The total reward accumulated in an episode and the distribution of the visited states is used to calculate the value function while the improvement step is carried out by making the policy greedy concerning the value function. This approach brings to light the exploration dilemma about how it is possible to guarantee that the algorithm will explore all the states without prior knowledge of the whole environment. ϵ-greedy policies are exploited instead of full greedy policy to solve this problem. An ϵ-greedy policy is a policy that acts randomly with probability ϵ and follows the policy learned with probability $(1-\epsilon)$. Unfortunately, even though Monte Carlo methods are simple to implement and they are unbiased because they do not bootstrap, they require a high number of iteration to converge. Furthermore, they have a wide variance in their value function estimation due to lots of random decisions within an episode [47].

Exploration Exploitation learning

One important aspect in RL is that it must explore the environment to gather information in order to build a policy. The agent do not want to leave unexplored areas but also to use the accumulated knowledge to make better decisions. In this sense, it is well known that a suitable trade-off between exploration and exploitation in imperative for global optimization performance. To gain ore rewards an agent can follow certain actions that are known to produce high immediate rewards, however, in order to know which is the best action it has to explore the environment. In many cases the exploration strategy depends on the time that the agent has interacted with the environment.

Example of the Exploration/Exploitation trade off Imagine you're a tourist exploring a new city, and you're on the hunt for the best restaurant among the plethora of restaurants available. You could play it safe and stick to what you know, revisiting a familiar chain or opting for a reliable cuisine. However, this strategy might mean missing out on the better options. On the other hand, you could explore more, trying out new places and better restaurants.
Much like the movie industry, where sequels of proven successes are churned out alongside risky new ventures, your decision as a tourist mirrors the exploration-exploitation dilemma.

Policy-based and Value-based

The use of policy or value function as the central part of the method represents another essential distinction between Model-free RL algorithms. *Policy-based* methods rely on approximating the agent's policy, typically represented as a probability distribution over available actions. These methods directly optimize the agent's behavior and may require multiple observations from the environment, making them less sample-efficient. On the other hand, *value-based* methods focus on determining the value of available actions to indirectly find the optimal behavior. Unlike policy-based methods, they prioritize identifying the best action rather than the probability distribution of actions. Additionally, *value-based* methods can leverage other sources such as old policy data or replay buffers.

II.5 Multi-Armed Bandit Problem: Techniques for Optimal Action Selection

II.5.1 Learning with Multi-Armed Bandits

In this section we present a general formulation of the classic Multi Armed Bandit (MAB) problem. The Multi-Armed Bandit Problem (MAB) is a fundamental problem in the field of reinforcement learning and decision-making under uncertainty. The problem involves a gambler (controller) who has to choose among several slot machines (also called "one-armed bandits") with unknown payout probabilities as shown in Figure II.27. The gambler's objective is to maximize his or her total payout by choosing the best slot machine to play.The MAB problem is commonly used in various fields, such as clinical trials, online advertising, and recommendation systems. The MAB problem can be seen as a trade-off between exploration and exploitation. Exploration refers to the gambler trying out different slot machines to learn about their payout probabilities, while exploitation refers to the gambler sticking to the slot machine that has shown the highest payout probability so far.

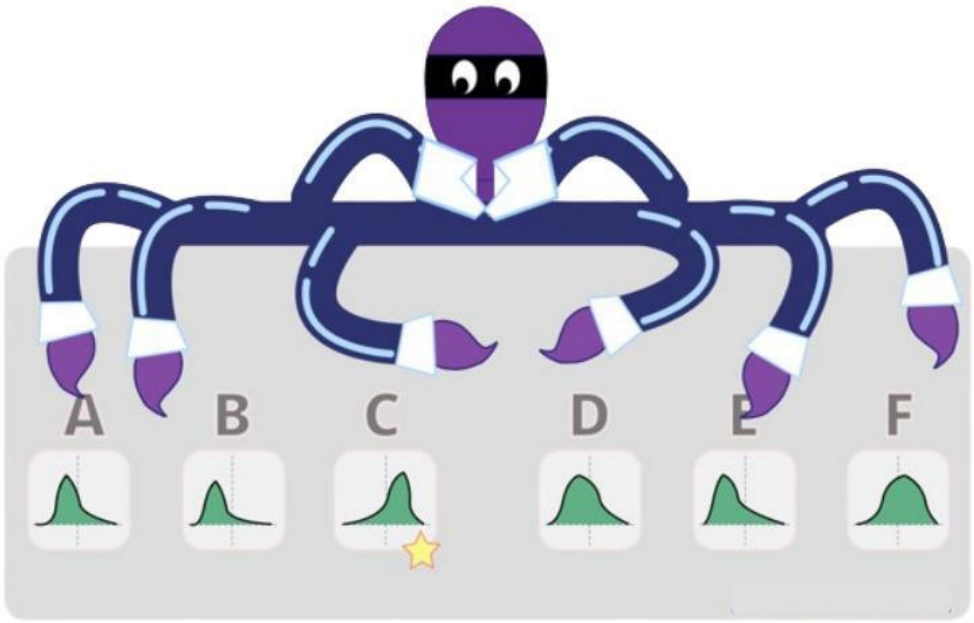


Figure II.27: Multi Armed Bandit illustration [4].

Here are some techniques employed in addressing the Multi-Armed Bandit Problem (MABP) through reinforcement learning:

- Thompson Sampling
- ϵ-Greedy Approach
- Softmax Exploration
- **Upper Confidence Bound**

II.5.1.1 Thompson Sampling

This description of Thompson sampling follows closely that of *Oliver Chappelle* and *Lihong Li* in [17]. Thompson Sampling tackles the Multi-Armed Bandit Problem by applying a Bayesian approach. It maintains a probability distribution for each machine's expected reward, representing our uncertainty about the true reward distribution. At each round, the algorithm samples from these distributions and selects the machine with the highest sample as the action to take. The Thompson sampling pseudo code is as follows [1]:

Algorithm 9 Thompson Sampling for Bernoulli Bandits

Input: Number of arms N
Output: Estimated success rates for each arm
for $i = 1, \ldots, N$ **do**
    $S_i \leftarrow 0$, $F_i \leftarrow 0$
end for
for $t = 1, 2, \ldots$ **do**
    for $i = 1, \ldots, N$ **do**
        Sample $\theta_i(t)$ from the Beta$(S_i + 1, F_i + 1)$ distribution
    end for
    Play arm $i(t) := \arg\max_i \theta_i(t)$ and observe reward r_t
    if $r_t = 1$ **then**
        $S_{i(t)} \leftarrow S_{i(t)} + 1$
    else
        $F_{i(t)} \leftarrow F_{i(t)} + 1$
    end if
end for

1. **Initialize Priors**: The algorithm initializes a prior distribution for each slot machine's reward. Typically, a non-informative prior like the Beta distribution is used, assuming equal probabilities for all possible rewards.

2. **Action Selection**: At time t, the algorithm samples a reward value from each machine's distribution. The gambler plays an arm according to the probability of its mean being the largest.

3. **Observe Reward**: After the selected action is executed, the algorithm observes the actual reward obtained from that machine, $S_i(t)$ successes (reward = 1) and $F_i(t)$ failures (reward = 0).

4. **Update Probability Distribution**: Using the observed reward, Thompson Sampling updates the probability distribution (posterior) for the selected machine using Bayesian inference. The updated distribution becomes the prior distribution for the next round.

5. **Repeat the process**

II.5.1.2 Epsilon-Greedy Approach

The epsilon-greedy algorithm is a straightforward approach that balances exploration (randomly choosing an arm) and exploitation (choosing the arm with the highest estimated reward). The epsilon-greedy algorithm begins by setting ϵ to a fixed value. After that a random probability value is generated between 0 and 1 for each trial. A random arm is chosen if the produced probability is less than (epsilon). Otherwise the arm with the highest reward at the time is chosen [64].

Epsilon Decreasing: Building upon the epsilon-greedy algorithm, epsilon-decreasing gradually reduces the exploration rate over time. This strategy allows for more exploration in the initial stages and transitions toward greater exploitation as the algorithm gathers more information.

Algorithm 10 Epsilon-Greedy Algorithm

Input: Number of arms N, initial ϵ, number of trials T
Output: Estimated rewards for each arm
Initialize ϵ to a fixed value
Initialize rewards $R_i \leftarrow 0$ and counts $C_i \leftarrow 0$ for each arm i
for each trial $t = 1$ to T **do**
    Generate a random probability value p between 0 and 1
    if $p < \epsilon$ **then**
        Choose a random arm a_t
    else
        Choose the arm a_t with the highest estimated reward R_i/C_i (if $C_i > 0$) or random if not played yet
    end if
    Play arm a_t and observe reward r_t
    Update rewards $R_{a_t} \leftarrow R_{a_t} + r_t$
    Update counts $C_{a_t} \leftarrow C_{a_t} + 1$
    Optionally decrease the value of ϵ
end for
return $\left\{\frac{R_i}{C_i} \mid i = 1, \ldots, N\right\}$

II.5.1.3 Softmax Exploration

Although ϵ-greedy action selection is an effective and popular algorithm of balancing exploration and exploitation in reinforcement learning. ϵ-Greedy's drawback is that when it explores it chooses equally among all actions. Consequently, it's just as probable to select the least promising action as it is to choose one that appears to be the second best. Softmax methods are based on Luce's axiom of choice (1959) [43] and pick each arm with a probability that is proportional to its average reward. Arms with greater empirical means are therefore picked with higher probability. The most common $Softmax$ method uses a *Gibbs*, or *Boltzmann*, distribution and in our thesis, we will present the *Boltzmann* exploration, a $Softmax$ method that selects an ram using a *Boltzmann* distribution [65]. Given initial empirical means $\hat{\mu}_1(0), \ldots, \hat{\mu}_k(0)$,

$$p_i(t+1) = \frac{e^{\hat{\mu}_i(t)/\tau}}{\sum_{j=1}^{k} e^{\hat{\mu}_k(t)/\tau}}, \quad i = 1, \ldots, n \tag{II.14}$$

where τ is a positive parameter called the temperature, controlling the randomness of the choice. High temperatures cause the actions to be all (nearly) equiprobable. Low temperatures cause a greater difference in selection probability for actions

that differ in their value estimates. In other words, When $\tau = 0$, Boltzmann Exploration acts like pure greedy. As τ tends to infinity, the algorithms picks arms uniformly at random [65, 43].

II.5.1.4 Upper Confidence Bound

The UCB family of algorithms has been proposed by *Auer*, *Cesa − Bianchi* & Fisher (2002) as a simpler, more elegant implementation of the idea of optimism in the face of uncertainty, proposed by *Lai* & *Robbins* (1985). The name of this family of algorithms stems from the fact that the upper bound plays a leading role in action selection, since we are searching the action with the highest reward rate. The following algorithms were used to identify a suitable strategy.

II.5.1.4.1 UCB1

The UCB1 algorithm relies on a straight-forward application of $Hoeffding$'s inequality; therefore, it is free from any prior knowledge on how the distribution resembles making it a $model - free$ algorithm.

Theorem (Hoeffding's Inequality). Let $Z_1, \ldots, Z_n$ be independent and identically distributed random variables such that $0 \leq Z_i \leq 1$. Then,

$$Pr\left[\left|\frac{1}{n}\sum_{i=1}^{n} Z_i - E[Z]\right| > \epsilon\right] \leq \delta = 2\exp\left(-2n\epsilon^2\right) \tag{II.15}$$

The intuition for this result is straightforward. When averaging a set of variables Z_i, it's expected that the average will be close to the expected value $\mathbb{E}[Z]$. Hoeffding's Inequality precisely quantifies the likelihood and degree of deviation from this expectation.

In $Hoeffing$'s inequality, $Q(a)$ denotes the true mean of the action, $\hat{Q(a)}$ as sample mean, $N_t(a)$ as the number of times an action was taken, and $U_t(a)$ as the upper confidence bound.

$$P[Q(a) > \hat{Q}_t(a) + U_t(a)] \leq e^{-2tU_t(a)^2} \tag{II.16}$$

Applied to the Multi-Armed Bandit problem, an agent's policy is to choose a slot machine in every iteration t that maximizes UCB1 using the following equation:

$$UCB1(a) = \arg\max_{j=1\ldots l}\left[\hat{p}_t(a_j) + C\sqrt{\frac{2\log\sum_1^l n_l}{n_t(a_j)}}\right] \tag{II.17}$$

The first part of the equation is related to the exploitation. $\hat{p}_t(a_j)$ denotes the average empirical reward received by the slot machine with a with index j at iteration t. In other words, it represents the expected reward in response to the selected slot machine.

The second part is linked with the exploration of possible slot machines since the gambler explores machines based on the uncertainty of an action's expected reward. where the square-root term measure of the uncertainty or variance in the estimate of a's value [65]. $n_t(a_j)$ denotes the number of times the machine with index j has been selected up until iteration t. the $\sum_1^l n_l$ is the total number of iterations up until iteration t. C is the exploration factor which is responsible on balancing the trade-off between exploitation and exploration, C parameter determines the confidence level. Usually C ranges between 1 and 2; with $C = 2$ meaning that we prioritize exploration rather then exploitation.

Each time a is selected the uncertainty is presumably reduced; $N_t(a)$ is incremented and, as it appears in the denominator of the uncertainty term, the term is decreased. If a machine other than a is selected; as it appears in the numerator the uncertainty estimate is increased.

Sutton and Barto (2018) [65] demonstrated the effectiveness of the UCB1 algorithm in tackling the 10-armed bandit problem, showcasing its robust performance. However, they highlighted the challenges associated with UCB1 when applied to non-stationary problems and environments with large state spaces.

II.5.1.4.2 Bayesian UCB

The Bayes-UCB Algorithm [41] is an extension of UCB1 for solving the MAB problem. Bayesian-UCB is inspired by the Bayesian modeling of the bandit problem which helps making a reasonable bound estimation if we know the distribution upfront.

Going from UCB1 to a Bayesian UCB [46] can be fairly simple. assuming the rewards of each arm are normally distributed, we replace the second part of the UCB1 term with:

$$C\frac{\sigma(x_a)}{\sqrt{n_t(a_j)}} \tag{II.18}$$

$\sigma(x_a)$ is the standard deviation of arm a's rewards at time t, c is an adjustable hyper-parameter for defining the size of the confidence interval we want to set to an arm's mean observed reward. n_a is the number of times arm a has been pulled. The resulting equation of arm selection is as follows:

$$Bayesian - UCB(a) = \arg\max_{j=1\ldots l}\left[\hat{p}_t(a_j) + C\frac{\sigma(x_a)}{\sqrt{n_t(a_j)}}\right] \tag{II.19}$$

II.5.1.4.3 UCB-Tuned

Auer et al.[9] propose UCB-Tuned as an improvement to UCB, besides the arithmetic mean of an arm, it also considers the variance in the bias sequence. This approach's logic is that variance-aware algorithms can quickly locate sub-optimal actions, thereby diminishing the total regret. Establishing that the slot machine to activate is the one that has the maximum value of the following equation:

$$UCB - Tuned(a) = \arg\max_{j=1...l} \hat{x_j} + \sqrt{\frac{\log n_t}{n_t(a_j)} min\{\frac{1}{4}, V_j(n_t(a_j))\}} \tag{II.20}$$

Where:

$$V_j(s) = \hat{x_{j,s}}^2 + \sqrt{\frac{2\log t}{n_t(a_j)}} \tag{II.21}$$

The $\frac{1}{4}$ signifies an upper bound on the variance of a Bernoulli random variable [9].

II.6 Guiding Local Search with Reinforcement Learning

In this thesis, guiding local search using Reinforcement Learning (RL) involves leveraging the Upper Confidence Bound (UCB) policy within a General Variable Neighborhood Search (GVNS) framework. The approach, termed *UCB_GVNS*, aims to balance exploration and exploitation by dynamically selecting the most promising neighborhood structures to apply during the search process. This methodology enhances the search for optimal solutions by adaptively focusing on the most beneficial neighborhoods, based on feedback received during the search.

The RL agent is tasked with navigating the combinatorial search space by making decisions that maximize cumulative rewards over time. These decisions are guided by the UCB policy, which systematically explores less-visited neighborhoods while exploiting those that have historically yielded high rewards. This balance is crucial for avoiding local optima and ensuring a thorough exploration of the search space.

An essential aspect of this approach is the credit assignment problem, which involves determining the contribution of specific actions (neighborhood choices) to the overall success. This problem is challenging due to the sparse, delayed, and potentially deceptive nature of rewards in RL environments. Addressing these challenges is vital for the effective functioning of the *UCB_GVNS* algorithm, as it ensures that the RL agent can accurately learn from its experiences and improve its decision-making process over time

II.6.1 Credit Assignment problem

In reinforcement learning (RL), an agent learns to make decisions by receiving and maximizing a reward signal from its environment, this is referred to as the *credit assignment problem* in the domains of machine learning and neurobiology. This reward signal guides the agent in understanding which actions are beneficial and which are not, ultimately shaping its behavior. The process of assigning rewards, however, is fraught with challenges, primarily because rewards can be sparse, delayed, or deceptive [22].

II.6.1.1 Challenges in Reward Assignment

One of the primary issues in RL is the problem of sparse rewards. In many environments, an agent might not receive any feedback (reward) for a prolonged period, making it difficult to learn effective policies. For instance, in a maze-solving task, the agent might only receive a reward upon reaching the exit, providing no intermediate feedback to guide its progress.

Another significant challenge is delayed rewards, where the consequences of an action are not immediately apparent. This delay can obscure the connection between actions and their outcomes, complicating the learning process. For example, in strategic games like chess, the impact of an early move might only become clear many turns later.

Deceptive rewards present yet another hurdle. In some environments, certain actions might yield short-term rewards but lead to long-term penalties. This can mislead the agent into developing suboptimal policies that prioritize immediate gains over better long-term strategies.

II.6.1.2 Potential-Based Reward Shaping

To address these challenges, authors in [10] have explored various methods of reward shaping, which involves modifying the reward signal to provide additional guidance to the agent. One effective technique is potential-based reward shaping (PBRS). PBRS modifies the reward function by incorporating extra knowledge about the task, which helps in accelerating the learning process without altering the optimal policy. This method leverages the cumulative rewards from past episodes to shape the reward signal dynamically. Specifically, the reward function is adjusted based on the agent's performance relative to its best and worst episodes so far. This adaptive method ensures that the agent receives continuous feedback on its improvement, even in environments with sparse or delayed rewards. The proposed reward function can be described by the following equation:

$$\phi(s,a,t) = \begin{cases} 0 & \text{if } R(s,a) = 0 \\ 1 + \frac{Rep - Rep_u(t)}{Rep_u(t) - Rep_l(t)} & \text{otherwise} \end{cases} \tag{II.22}$$

where $R(s,a)$ is the immediate reward, Rep is the cumulative reward for the current episode, $Rep_u(t)$ is the maximum episode reward observed so far, and $Rep_l(t)$ is the minimum episode reward observed so far [10]. The reward assignment problem in reinforcement learning is a significant challenge that impacts the efficiency and effectiveness of learning agents. Techniques such as potential-based reward shaping provide promising solutions by enhancing the reward signal with additional knowledge. By dynamically adjusting rewards based on the agent's performance, these methods offer a robust framework for improving learning outcomes in complex and sparse environments. Our reward assignment approach is inspired by PBRS but differs in several key aspects. These differences will be explored in detail in Chapter III. Future research can further refine these approaches, integrating more sophisticated knowledge extraction techniques to enhance the adaptability and performance of RL agents.

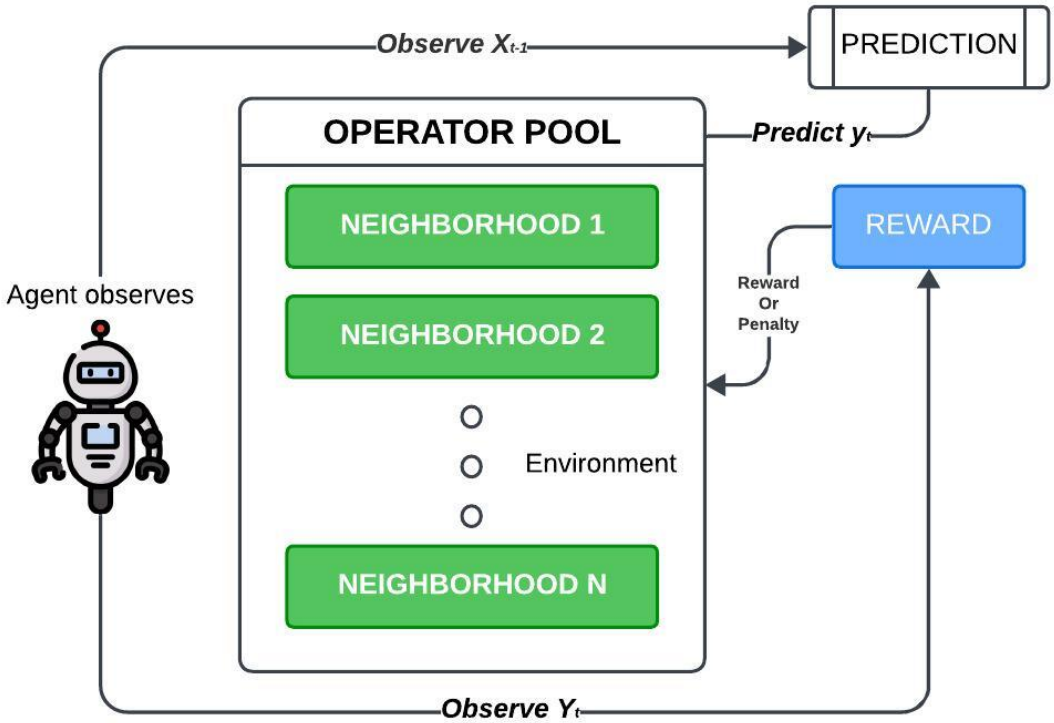


Figure II.28: UCB_GVNS contextual diagram

Figure II.28 presents the schematic diagram of the *UCB_GVNS*. at iteration t, The agent observes the state of the environment then makes a prediction of the best neighborhood to be used, the agent then observes the reward given to the neighborhood chosen, the process repeats for a number of episodes exploring possible neighborhoods in the operator pool and their rewards.

II.7 Conclusion

In summary, Chapter II has delved into the powerful combination of meta-heuristic algorithms and machine learning techniques to tackle combinatorial optimization problems. By harnessing the capabilities of GAs and VNS, enhanced by the reinforcement learning strategy of the UCB1, we have developed a robust hyper-heuristic framework. This framework will be thoroughly evaluated in Chapter 3, where we will address the node placement problem, detail the proposed model and framework, and present the results obtained from our approach. Additionally, we will compare our findings with other existing methods to highlight the effectiveness of our solution.

Chapter III

Implementation and Experimental Results

III.1 Introduction

In this chapter, we detail the implementation of our proposed hyper-heuristic approach to solve the deterministic deployment problem of Wireless Sensor Networks (WSN) dedicated to the monitoring of sensitive sites. The proposed approach integrates meta-heuristic algorithms, such as genetic algorithms and variable neighborhood search, with reinforcement learning techniques, specifically the Upper Confidence Bound (UCB1) algorithm. Initially, a genetic algorithm provide an initial solution. This solution is then passed to the UCB1-guided General Variable Neighborhood Search (GVNS) algorithm, which dynamically selects the optimal neighborhood structure at each iteration to enhance the solution's quality.

The following sections provides a comprehensive guide to the implementation and optimization of our surveillance network model. The insights gained from this chapter are instrumental in developing robust security solutions, ensuring rapid and reliable alert transmission in various operational environments.

III.2 Optimal Node Placement Problem

Considering an area of interest (or site) to be monitored using a *WSN*. The sentinel sensor nodes (SSNs) and RNs are deployed in a deterministic manner so that every point on the border is covered by at least one SSN, and there exists a path, composed of a limited number of hops (minimum relays), from each SSN to the sink.

The deployment space of the site is discretized. The site is modeled by a two-dimensional square grid due to the challenge of exploring all possible positions within a well-defined area. The centers of the grid cells are indexed by i and represent candidate locations where nodes can be deployed.

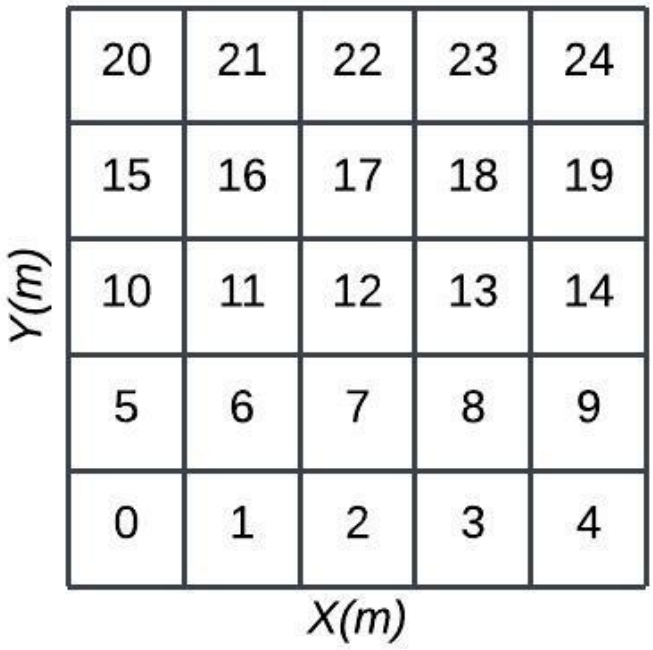


Figure III.1: Spatial Discretization: Square Grid Node Deployment

The set of cells representing the boundary of the considered space is denoted by P. For a point $p \in P$ (center of the cell) to be covered, the entire cell must be covered (the four points constituting a cell). The cells are numbered from 0 to $(n-1)$ as shown in Figure III.1. "n" represents the number of cells in the grid. In the case of Figure III.1, $n = 25$. The Cartesian coordinates of a position i will be calculated as follows:

$$\begin{cases} x_i & = (i \mod Cols) \cdot step + \frac{step}{2} \\ y_i & = (i \div Cols) \cdot step + \frac{step}{2} \end{cases} \tag{III.1}$$

Where:

- i: cell index
- Cols = number of columns in the grid
- step = distance between the centers of two neighboring cells

This equation applies to both squared and non-squared grids. In a squared grid, the number of rows and columns are equal (e.g., Cols = $\sqrt{n}$ for n cells), ensuring uniform cell spacing. For non-squared grids, where the number of rows differs from the number of columns, Cols defines the number columns, maintaining correct positioning of cell centers. This method, derived from the equation in [6], provides a systematic way to map cell indices to Cartesian coordinates, ensuring accurate spatial representation of the discretized space.

SSNs and RNs form a multi-hop network that must be fully connected to the sink. The sink is randomly placed within the grid. Two nodes are directly connected if the distance between them is less than or equal to the communication range R_c.

III.3 Model Description

The SSNs are positioned at the borders of the network as shown in Figure III.2. These sentinels serve as the first line of defense and are responsible for generating an alert upon detecting intrusion.

Element	**Description**
`grid`	Size of the grid representing the area
`sink`	Coordinates of the sink
`sinked sentinels`	Sentinels that have been connected to the sink
`sinked relays`	Relays that have been connected to the sink
`free slots`	Available slots for relays
`communication range`	Connectivity span for Relays, Sensors and sink nodes
`Sensing range`	Sensing range for the sensor nodes
`mesh size`	Size of the mesh
l_{max}	Maximum number of neighborhoods
α, β	Parameters for epsilon constraints

Table III.1: Important elements of UCB_GVNS Hyper-heuristic

The Table III.1 lists essential parameters and elements, such as the grid size, sink coordinates, and communication ranges, which are crucial for understanding the deployment and operational efficiency of the surveillance network. These elements are part of the UCB_GVNS Hyper-heuristic, which plays a pivotal role in our model's optimization process. This hyper-heuristic utilizes a combination of General Variable Neighborhood Search (GVNS) and Upper Confidence Bound (UCB1) algorithms to effectively deploy relay nodes (RNs) within the network, ensuring optimal alert transmission from sentinels (SSNs) to the sink.

Meanwhile, RNs are distributed throughout the network area. These relays act as intermediate nodes, facilitating the transmission of alerts between the sentinels and the sink located centrally within the network. RNs play a crucial role in ensuring that alerts can travel efficiently across the network, minimizing transmission delays and optimizing network performance.

Finally, the sink is positioned centrally within the network, represented as a triangle in Figure III.2. The receives alerts from both SSNs (in the case where the SSN is directly connected to it) and from RNs, consolidating the information for further analysis or decision-making purposes. Figure III.2 clearly demonstrates that all border cells are covered. Therefore, we achieve complete border coverage along with a path to the sink from each SSN in the bordering.

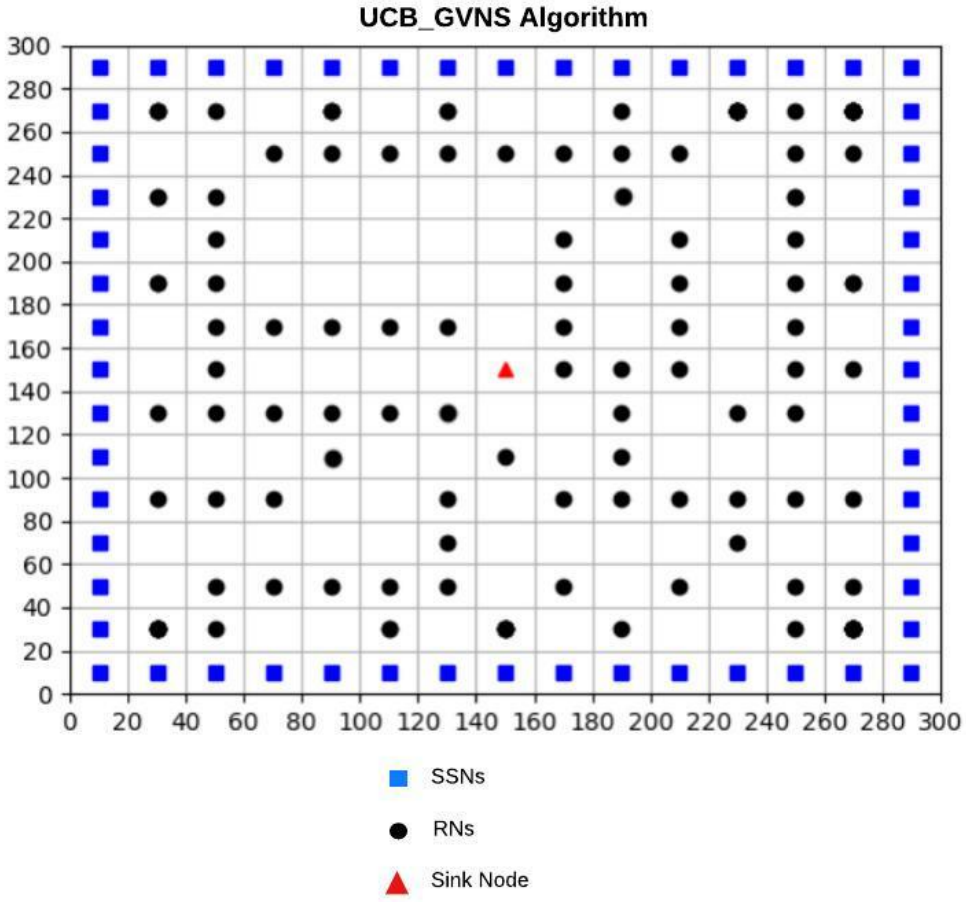


Figure III.2: Surveillance Model

III.4 Description of the Proposed framework

In this master's thesis, we delve into the unresolved question of determining the most effective sequence of local search operators to optimize the performance of Variable Neighborhood Search. Our approach introduces a RL-based meta-heuristic algorithm termed *UCB_GVNS*, which draws inspiration from the Multi-armed Bandit problem, a specific instance of a single-state reinforcement learning problem.

III.4.1 Initial Solution

As depicted in Figure III.3, the *UCB_GVNS* algorithm receives its initial solution from a Genetic Algorithm (GA). As noted [69], employing a genetic-based initial solution is advantageous because GAs offer control and consistently produce solutions with high fitness values, presenting a challenging starting point for our algorithm.

The GA operates by iteratively evolving a population of candidate solutions through generations. It employs mechanisms such as crossover and mutation to generate new solutions from existing ones. The crossover operation involves combining genetic information from two parent solutions to produce offspring, while mutation introduces random modifications to diversify the solution space.

Furthermore, the evaluation function assesses the fitness of each solution based on predefined objectives, allowing for the selection of the most promising individuals for reproduction. The GA's initialization phase involves generating an initial population of solutions through randomization.

Throughout the algorithm's execution, fitness scores are computed for each solution, guiding the selection of parent solutions for reproduction. This iterative process continues until a termination condition is met, which is reaching a maximum number of generations. Ultimately, the GA aims to produce an initial solution that serves as a starting point for subsequent optimization using the UCB_GVNS hyper-heuristic.

Genetic Algorithm parameters

To provide a comprehensive understanding of the solution process, this subsection focuses on the GA employed to generate the initial solution. We will detail the specific GA operators chosen and the parameter settings utilized in our study.

Fitness The evaluation methodology employed in this work leverages a multi-objective fitness function termed ϵ-constraints. This function aims to achieve a balance between two key performance indicators ($KPIs$): minimizing the number of RNs deployed to ensure the connectivity between every SSN and the sink node,

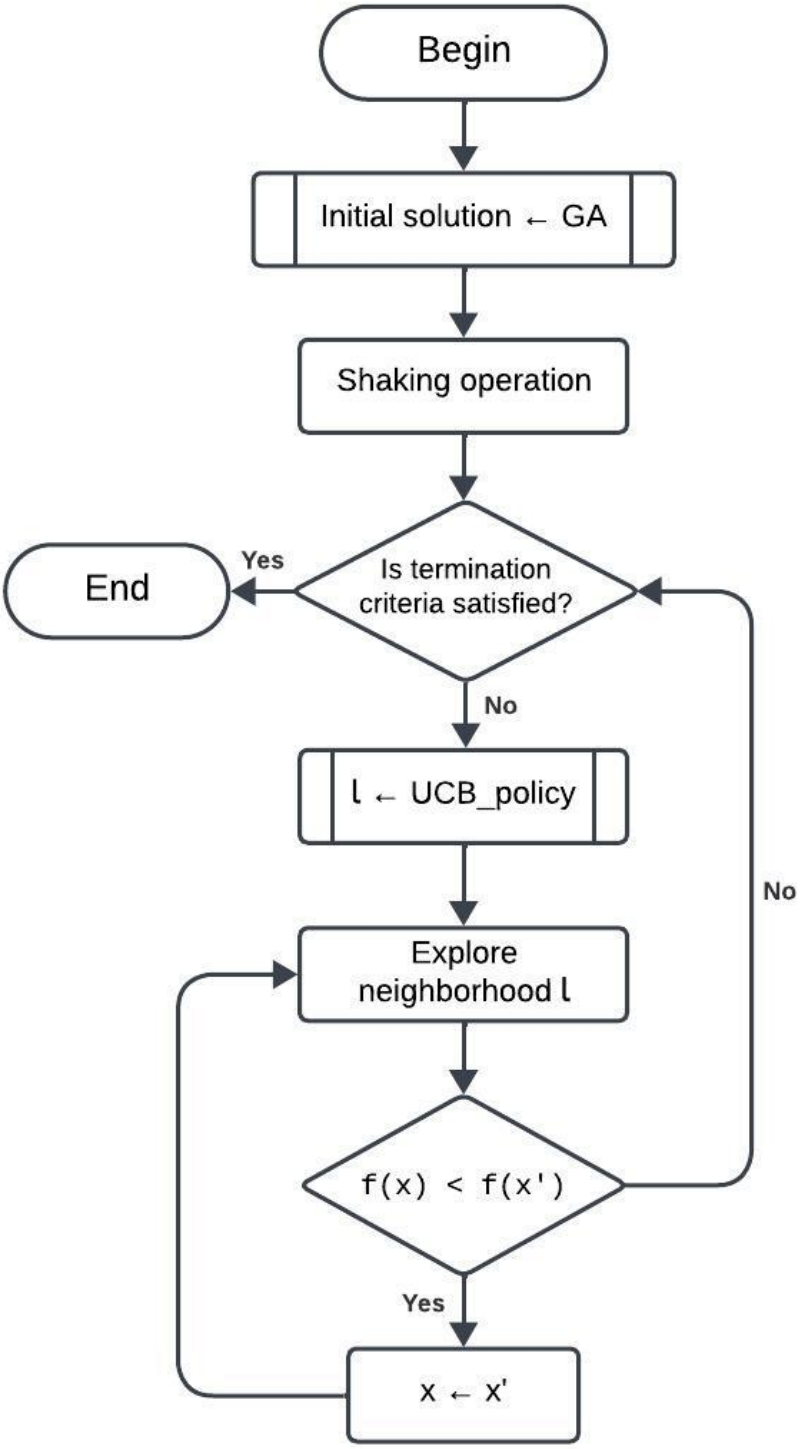


Figure III.3: UCB_GVNS Framework Flowchart

and the Network Diameter [1] (ND). The function calculates a weighted sum, combining the number of deployed RNs and the ND (representing the number of hops). Weighting factors allow for prioritization based on application needs. Additionally, the function incorporates a penalty for infeasible deployments characterized by unreachable sentinels or excessive communication hops exceeding a predefined threshold (ϵ). This approach ensures efficient exploration of the deployment search space by penalizing infeasible solutions.

Selection The selection process within this genetic algorithm employs a straightforward elitism strategy. Here, the evaluate function calculates a fitness score for each solution in the population, considering both the number of connected relays and the sentinel network diameter. These solutions are then ranked based on their fitness scores, with higher scores indicating better performance. To generate offspring for the next generation, the algorithm leverages the top two ranked solutions (parents) identified through this ranking process. This elitism approach ensures that solutions with superior performance have a higher chance of influencing the next generation, potentially leading to an overall improvement in the population's quality over time.

Crossover The genetic algorithm incorporates a Uniform Crossover technique to create new offspring solutions (children) by combining genetic material from two high-performing parent solutions. Following parent selection based on their fitness scores, the algorithm prioritizes the parent with a more compact structure (fewer nodes) as the primary source. It then iterates through corresponding routes (genes) from both parents. For each gene pair, a coin flip determines whether the offspring inherits the route from the primary or secondary parent. This approach offers a balance between simplicity and exploration, allowing the creation of diverse offspring with a chance of inheriting successful routes from both parents. Below you will find the uniform crossover pseudocode in algorithm 11.

Mutation To prevent the population from stagnating on sub-optimal solutions, the genetic algorithm incorporates a mutation process that injects controlled variability. This is achieved by randomly selecting a solution and a sentinel within that solution. The algorithm then randomly selects a slot from the available free slots. If the chosen slot is already occupied by a relay node in the selected sentinel's route, the relay node is removed, allowing the slot to become free again. Conversely, if the chosen slot is free, a new relay node is added to the sentinel's route, effectively extending its reach. And below the pseudo-code for the bit flip mutation in algorithm 12:

[1] the longest shortest path

Algorithm 11 Uniform Crossover

Input: Parents P_1 and P_2
Output: Offspring O_1 and O_2
$O_1 \leftarrow$ empty individual
$O_2 \leftarrow$ empty individual
for each gene i in P_1 and P_2 **do**
  if random number r between 0 and $1 < 0.5$ **then**
    $O_1[i] \leftarrow P_1[i]$
    $O_2[i] \leftarrow P_2[i]$
  else
    $O_1[i] \leftarrow P_2[i]$
    $O_2[i] \leftarrow P_1[i]$
  end if
end for
return O_1, O_2

Algorithm 12 Single Bit Flip Mutation

Input: Individual I
Output: Mutated individual M
$M \leftarrow$ copy of I
Choose a random gene i from M
Flip the value of gene i (0 to 1 or 1 to 0)
return M

Hyper-parameters Our genetic algorithm relies on two crucial hyper parameters: population size and number of generations. Balancing between high-quality solutions and efficient execution time is paramount. After thorough experimentation, we settled on a population size of 5 individuals and 15 generations. These values were meticulously selected through a trial-and-error approach to optimize the trade-off between solution quality and execution time. Notably, we deliberately kept the mutation rate and crossover rate parameters constant to maintain simplicity and focus on the primary parameters influencing the algorithm's performance.

III.4.2 Main components of the proposed framework

In hyper-heuristic frameworks, the focus lies on developing a robust strategy for solving combinatorial problems by intelligently selecting and applying various heuristic methods. At its core, a hyper-heuristic framework involves the utilization of meta-heuristics that operate at a higher level of abstraction, guiding the application of lower-level heuristics or neighborhood operators in our case. These neighborhood

operators define the space of potential solutions for a given problem, each capable of transforming a solution into a nearby solution within the problem space. For instance, in WSN optimization scenario, neighborhood operators might involve actions like adding, deleting, or relocating relay nodes. By employing a range of such operators, a hyper-heuristic framework aims to efficiently explore and exploit the solution space, leveraging the strengths of different heuristics to achieve optimal or near-optimal solutions. The combination of different heuristic methods within this framework allows for adaptive and dynamic problem-solving strategies, enhancing the scalability and effectiveness of the overall optimization process [67, 49].

III.4.2.1 Neighborhood operators

Given a set S that contains all the possible solutions of a combinatorial problem, a neighborhood operator N is a function that maps a given solution $s \in S$ to a neighborhood of solutions$N(s) \subseteq S$. The neighborhood structure our model uses, consists of five neighborhood operators, namely: *Add random relay node, Delete random RN, Relocate a RN, Add a relay node next to a SSN, Delete a RN next to SSN*

III.4.2.2 Combining UCB1 and GVNS

This section delves into the optimization method employed for sensor deployment within a WSN. This innovative approach leverages a powerful combination of two techniques: Upper Confidence Bound applied to a General Variable Neighborhood Search *(UCB_GVNS)*.

The *UCB_GVNS* algorithm thrives on the exploration-exploitation trade-off inherent to UCB1, strategically guiding a GVNS search process. Let's dissect the algorithm (Algorithm 14) to understand its inner workings, with the UCB1 operator selection pseudo-code itself depicted in algorithm 13.

First, the algorithm starts with the shaking operation, which will be discussed in detail later. As shown in the Flowchart III.3, until the termination criteria are met, a set of steps is performed by the algorithm. The UCB1 policy within the *UCB_GVNS* algorithm is crucial for balancing exploration and exploitation during this process. It systematically selects which neighborhood to explore at each iteration by calculating an upper confidence bound for each neighborhood, reflecting both potential reward (solution quality) and uncertainty (the need to explore less-visited neighborhoods). This approach ensures that the algorithm does not get stuck in local optima by continually assessing and exploring various neighborhoods. By prioritizing neighborhoods with higher upper confidence bounds, the UCB1 policy effectively navigates the solution space, finding the optimal action to take at every iteration. If the neighborhood chosen by the UCB policy is applied and an im-

provement is found in term of fitness value, the RL agent keeps exploiting the same neighborhoods until the local optimum is found. These steps keep repeating until the termination criteria are met, resulting in an optimized network configuration.

The ND is calculated using *Dijkstra* algorithm, calculating the shortest path from the sink to every SSN in the network and then picking the longest path among the shortest ones as the diameter.

Algorithm 13 UCB1 Adaptive operator selection

Input: $C, s, n, \hat{p}$
Output: Selected neighborhood index {s: Time step, k: neighborhood count}
if operators that have not been selected exist **then**
    $x \leftarrow$ uniform selection from operator pool;
else
    $x \leftarrow \arg\max_{j=1\ldots k} \left(\hat{p}_j + C\sqrt{\frac{2\log(s)}{n_j}}\right)$
end if
$n_x \leftarrow n_x + 1$;

III.4.2.2.1 Shaking

The shaking operation is designed to diversify the search process by generating new candidate initial solution. The shaking operation involves sequentially applying all neighborhood operations to generate diverse solutions. The sequence of neighborhoods for the shaking operation are: *adding a relay next to an existing relay, deleting a random relay, deleting a relay next to a sentinel with multiple relay neighbors, relocating a relay, and adding a relay next to a sentinel without relay neighbors.* The systematic exploration of various neighborhoods helps escaping local optimum and exploring a wide solution space.

The shaking operation in our hyper-heuristic can be likened to the initial action in the multi-armed bandit problem, where every arm (i.e. neighborhood) is pulled once. In the context of multi-armed bandit problem, pulling each arm initially ensures that every arm is explored, providing baseline performance data for each arm. This initial exploration phase is crucial for establishing a foundation for the subsequent balance between exploration and exploitation.

Similarly to the *UCB_GVNS* algorithm, the shaking operation ensures that every neighborhood (akin to an arm in the multi-armed bandit) is explored initially. By sequentially applying each neighborhood operation during the shaking phase, the algorithm gathers initial performance metrics for the neighborhoods. This comprehensive initial exploration provides valuable experience and insights into the

Algorithm 14 UCB_GVNS pseudocode

```
1: Output: Optimal sinked relays, free slots
2: l ← 1
3: while termination criteria not met do
4:    i ← 0
5:    improvement ← True
6:    previous ← Calculate fitness
7:    while improvement and i < len(sinked relays) + 1 do
8:       improvement ← False
9:       i ← i + 1
10:      chosen_neighborhood ← Apply UCB1 policy to select neighborhood
11:      Apply neighborhood action based on chosen neighborhood
12:      after ← Calculate fitness
13:      if previous > after then
14:         improvement ← True
15:         Reward ← Credit Assignment
16:      else
17:         Penalty ← Credit Assignment
18:      end if
19:   end while
20: end while
```

potential effectiveness of different neighborhoods, which is essential for the UCB1 strategy to make informed decisions in the later stages of the optimization process.

III.4.2.2.2 Fitness calculation

The fitness calculation in our approach involves a multi-objective evaluation aimed at balancing two critical performance metrics for relay deployment: the number of relays successfully connected to the sink node and diameter of the network. Specifically, the fitness function is defined as a weighted sum of the number of sinked relays and the network diameter, which represents the communication hops. The weights α and β allow for prioritization based on specific application requirements. If a solution breaches connectivity constraints or exceeds a predefined diameter threshold ϵ, it is assigned an infinite fitness value, effectively penalizing infeasible solutions. This method ensures efficient exploration of the solution space by focusing on viable deployment configurations. Below the ϵ-constraints pseudocode:

Algorithm 15 epsilon_constraints

```
1: Input: Solution s
2: ϵ ← grid
3: fitness ← (alpha × sinked_relays) + (beta × diameter)
4: if Connectivity breach or diameter > epsilon then
5:     return ∞
6: else
7:     return fitness
8: end if
```

III.4.2.3 Termination criteria

The *UCB_GVNS* algorithm employs a multi-pronged approach to determine when to stop searching for the best relay deployment configuration. This termination strategy balances achieving a good solution within a reasonable amount of time.

One approach is a simple fixed budget method. You define a maximum number of iterations the algorithm will run for, regardless of the improvement observed. This ensures the program terminates after a set amount of computational effort is expended, which is in our case the number of meshes.This approach is termed *STAT_UCB_GVNS*.

However, a fixed number of iterations might not always be ideal. To address this, the algorithm incorporates early stopping criteria based on the concept of stagnation. Here, the concept of velocity is introduced. Velocity captures the change in solution quality (measured by a fitness function) between consecutive iterations. It's calculated as the absolute difference in fitness values. Additionally, an average velocity is computed over a window of past velocities to smooth out short-term fluctuations and assess the overall trend.

The early stopping criteria leverage these velocity measures. As the algorithm 16, if the average velocity over recent iterations falls below a certain threshold (indicating minimal improvement) and the number of consecutive iterations without improvement exceeds a predefined threshold (patience), the algorithm terminates early. This logic prevents the program from getting stuck in situations where it keeps exploring without significant progress, the approach is termed *DYN_UCB_GVNS*.

Choosing the most suitable termination criteria depends on your specific goals and constraints. If computational resources are limited, a fixed maximum iteration approach might be preferred. On the other hand, if achieving the absolute optimal solution is crucial, a higher number of iterations or a more stringent early stopping criterion based on velocity might be necessary. Additionally, for problems that exhibit slow convergence or a high risk of getting stuck in suboptimal solutions, a velocity-based early stopping approach can be particularly beneficial.

Algorithm 16 Dynamic Termination Criteria for UCB_GVNS

Input: Parameters: *max_iterations*, *patience*, *velocity_threshold*, *max_consecutive_errors*
Output: Termination condition met based on specified criteria
Variables: $iteration = 0$, $consecutive\_errors = 0$, $last\_n\_velocities = []$
while $consecutive\_errors < max\_consecutive\_errors$ **do**
    Perform UCB_GVNS logic
    Calculate improvement velocity for current iteration
    Update *last_n_velocities* with current velocity
    Calculate average velocity from *last_n_velocities*
    Check termination criteria ($iteration < max\_iterations$ and $average\_velocity < velocity\_threshold$)
    if termination criteria met **then**
        break
    end if
    Update counters
    $iteration \leftarrow iteration + 1$
end while
return Termination condition met

III.4.2.4 Rewards and Penalties

The *UCB_GVNS* framework adopts a multi step learning procedure where the outcome of an action referred to as a reinforcement signal is infrequent and delayed in time. *The credit assignment* problem is the problem of identifying which actions (neighborhoods) or sequence of actions lead to better rewards.

Our hyper-heuristic framework utilizes a novel, non-binary reward scheme, adopting a different strategy for *rewards* and *penalties* signals.

In case of a positive feedback (reward), the following equation determines the reward. The new solution, $after$ fitness is always smaller than *previous* fitness for minimization problems, Rs represents the velocity of improvement after applying the neighborhood action, in other words it represents the distance between the new solution and the prior one. Rf represents the reward factor as every action (neighborhood) has a fixed factor to differentiate between neighborhoods and there rewards.

$$reward = Rs * Rf \tag{III.2}$$

Where:

$$Rs = |previous - after|$$

In contrast, when the neighborhood yields a solution with a worse fitness value, the following equation determines the negative rewards. Although similar, the dif-

ference between the reward and penalty equation is that the penalty has a negative value and the Pf has fixed value for every operator, conversely to the reward equation where every neighborhood has a reward factor associated with it.

$$penalty = -(Rs * Pf) \tag{III.3}$$

Where:

$$Rs = |previous - after|$$

The resulting reward signals from all operators are stored in a *quality* array associated with every neighborhood, the quality associated with the neighborhood changes according to the signal resulted by applying it.

III.5 Simulation and results discussion

III.5.1 Simulation environment

Simulation settings refers to the predefined parameters and conditions under which the simulation is conducted. These settings determine the environment and constraints for the network model, ensuring consistency and reproducibility of the simulation results. The specific simulation settings used in our study are detailed in Table III.2.

Parameter	Values
Grid size	17×17, 20×20, 25×25, 30×30, 35×35
Single mesh size	20 m
Surface areas	$115600\,\mathrm{m}^2$, $160000\,\mathrm{m}^2$, $250000\,\mathrm{m}^2$, $360000\,\mathrm{m}^2$, $490000\,\mathrm{m}^2$
Communication range (SSNs, RNs and the sink)	30 m
Sensing range (SSNs)	15 m
Number of scenarios	10
α, β	0.5, 0.5

Table III.2: Simulation environment

III.5.2 Performance Metrics

Table III.3 outlines the performance measures utilized in this study.

Measure	Description
Average fitness	$\frac{\sum \text{fitnesses}}{\text{number of scenarios}}$. The fitness is calculated using ϵ-constraints at the end of each scenario.
Average total number of RNs	$\frac{\sum \text{total number of RNs}}{\text{number of scenarios}}$. The total number of RNs represents the number of RNs obtained after each scenario.
Average of the avg. hops	$\frac{\sum \text{total average hops}}{\text{number of scenarios}}$. The average hops represent the average number of hops after each scenario.
Average execution time	$\frac{\sum \text{execution time}}{\text{number of scenarios}}$. Execution time represents the time elapsed for each scenario.

Table III.3: Performance Metrics

III.5.3 Experimental Setup

To conduct the experiments outlined in this thesis, we leveraged a High-Performance Computer (HPC) equipped with Ubuntu Linux, accessed via SSH protocol. The HPC boasted an 8-core CPU and 32GB of RAM. Python 3.5 served as the primary programming language for implementing the hyper-heuristic framework.

III.5.4 Results discussion

In this subsection, we compare the performance results of our proposed approach, namely UCB_GVNS, with its two variants UCB_GVNS with static stopping criteria, labeled *STAT_UCB_GVNS* and UCB_GVNS with dynamic stopping criteria, labeled *DYN_UCB_GVNS*, with Basic Variable Neighborhood Search (BVNS) proposed in [69]. It is worth noting that for the sake of fairness in the comparison, we replaced the greedy algorithm used by the BVNS in [69] to generate the initial solution, with the GA used by *DYN_UCB_GVNS*, *STAT_UCB_GVNS*.

Tables III.4, III.5, and III.6 present the simulation results comparing RL-based approaches and the BVNS approach, starting from the initial solution provided by the GA, across various metrics (number of RNs, ND, average hops, and fitness value). "**Static**" indicates the *STAT_UCB_GVNS*, and "**Dynamic**" indicates *DYN_UCB_GVNS*. It is apparent that *UCB_GVNS* outperforms the *BVNS* in terms of number of relays and fitness value, with the *DYN_UCB_GVNS* variant performing the best. In terms of Network diameter and average Hops, the values

provided by *BVNS* remain the same as the values obtained by the GA, since BVNS do not explore the solution space compared to *UCB_GVNS* based methods.

Metric	17x17			20x20		
	Static	Dynamic	BVNS	Static	Dynamic	BVNS
GA Number of RNs	213	213	213	303	303	303
GA ND	9	9	9	$9.2 \approx 10$	$9.2 \approx 10$	$9.2 \approx 10$
GA Avg. hops	9	9	9	$9.2 \approx 10$	$9.2 \approx 10$	$9.2 \approx 10$
GA Fitness	111.25	111.25	111.25	156.1	156.1	156.1
Number of RNs	109	98	204	149	130	292
ND	15	15	9	15	16	10
Avg. hops	12	12	9	13	13	10
Exec. time	13m 47s	23m 47s	5m 7s	48m 2s	1h 14m 43s	15m 11s
Fitness	62	56	106.5	83	73	151

Table III.4: Comparison of GA and UCB_GVNS performance with dynamic and static stopping criteria, and BVNS for 17x17 and 20x20 grids

Metric	25x25			30x30		
	Static	Dynamic	BVNS	Static	Dynamic	BVNS
GA Number of RNs	497	497	497	738	738	738
GA ND	12	12	12	16	16	16
GA Avg. hops	12	12	12	15	15	15
GA Fitness	254.5	254.5	254.5	377	377	377
Number of RNs	247	245	485	325	318	729
ND	22	22	12	27	27	15
Avg. hops	19	19	12	22	22	15
Exec. time	3h 55m	4h 54m	28m 32s	7h 55m	8h 32m	35m 4s
Fitness	134.5	133.9	248.5	175.5	171.95	372

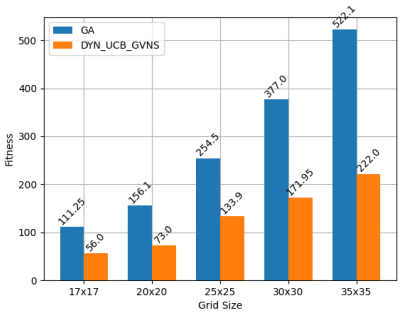
Table III.5: Comparison of GA and UCB_GVNS performance with dynamic and static stopping criteria, and BVNS for 25x25 and 30x30 grids

Metric	35x35		
	Static	Dynamic	BVNS
GA Number of RNs	1027	1027	1027
GA ND	17	17	17
GA Avg. hops	17	17	17
GA Fitness	522.1	522.1	522.1
Number of RNs	429	420	1016
ND	24	24	17
Avg. hops	22	22	17
Exec. time	11h 15m	12h 40m	1h 2m 11s
Fitness	226.05	222	516.5

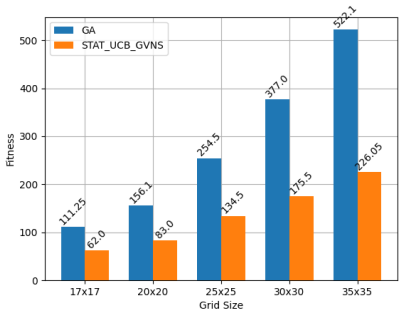
Table III.6: Comparison of GA and UCB_GVNS performance with dynamic and static stopping criteria, and BVNS for 35x35 grids

III.5.4.1 Evaluating the fitness improvement

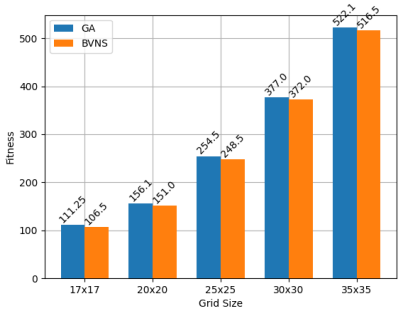
We compare the fitness improvements of the three approaches: *DYN_UCB_GVNS*, and *STAT_UCB_GVNS*, and *BVNS*. Figure III.4 illustrate the comparative performance of these approaches. The results indicate that the *UCB_GVNS* approaches outperformed the *BVNS*, with the *DYN_UCB_GVNS* achieving the highest fitness improvement. This suggests that the adaptive nature of the dynamic stopping criteria allows for a more efficient exploration the solution space, leading to superior optimization results compared to both the static stopping criteria and the *BVNS* approach.



(a) *DYN_UCB_GVNS*



(b) STAT_UCB_GVNS



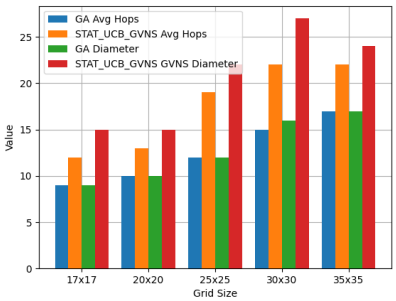
(c) BVNS

Figure III.4: Fitness Comparison between GA and the three approaches

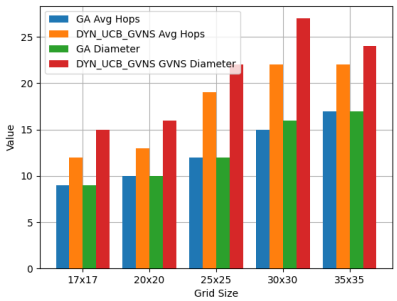
III.5.4.2 Evaluating the average hops and Network Diameter

Figure III.5 presents the Average number of hops from all sentinels to the sink through relays:

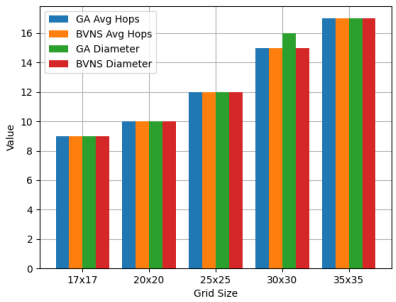
As shown in Figure III.5 the average hops count and network diameter show an increase for the RL-based methods. Conversely to the BVNS approach, which



(a) *STAT_UCB_GVNS*



(b) DYN_UCB_GVNS



(c) BVNS

Figure III.5: Network Diameter and AVG hops: Comparison between GA and the three approaches

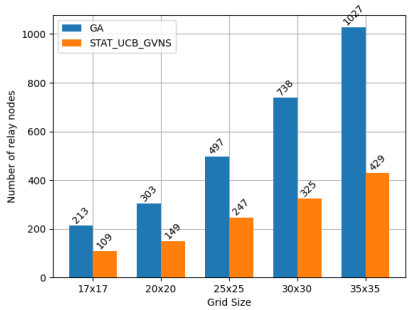
can be interpreted as the balance reached between number of relays and network diameter in order to minimize the fitness value.

III.5.4.3 Evaluating the number of relays deployed

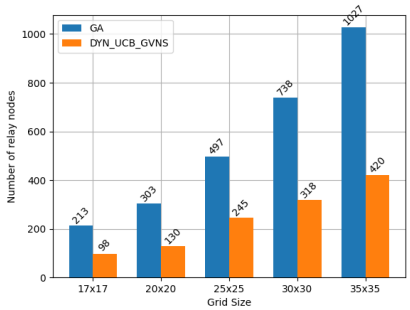
Figure III.6 presents the Average number relays deployed for different grid sizes. The figure shows that the RL-based methods achieve a better number of relay nodes deployed in every grid comparing to the *BVNS* algorithm, with the *DYN_UCB_GVNS* achieving the best number of relays deployed.

III.5.4.4 Comparing fitness convergence

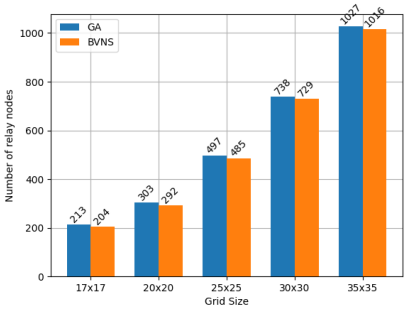
Both RL-based approaches show significant fitness improvement. However, while the *DYN_UCB_GVNS* takes longer in terms of execution time, it guarantees fitness convergence more effectively than the *STAT_UCB_GVNS*, which stops when the maximum number of iterations is reached as depicted in Figure III.7



(a) *STAT_UCB_GVNS*



(b) DYN_UCB_GVNS



(c) BVNS

Figure III.6: Number of relays Comparison between GA and the three approaches

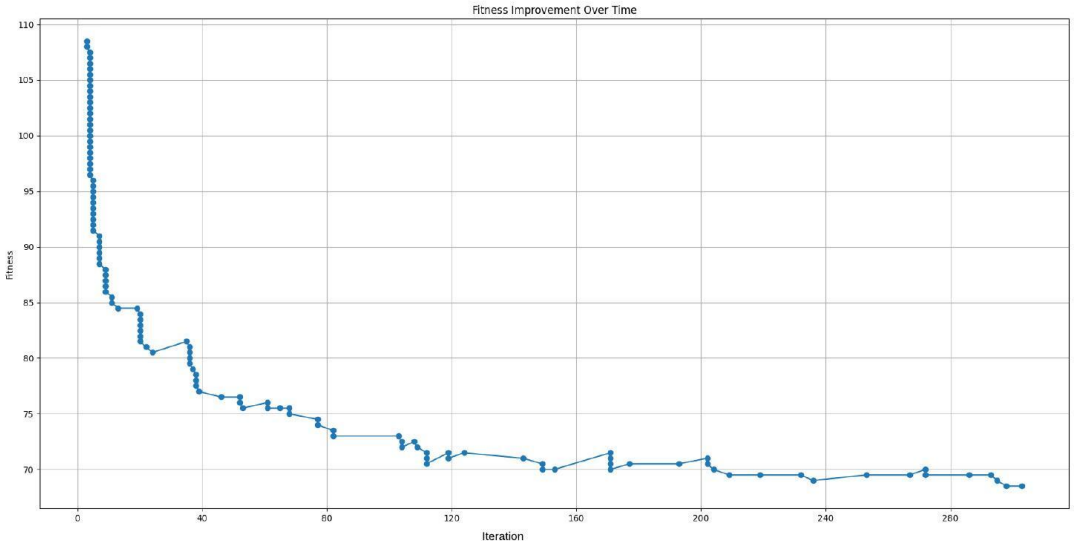
III.5.4.5 Evaluating the Execution time

Figure III.8 represents the execution time for the proposed approaches, the *STAT_UCB_GVNS*, the *DYN_UCB_GVNS*, and *BVNS*. The RL-based methods takes longer time comparing to the BVNS, which is necessary to explore the neighborhoods efficiently. The *DYN_UCB_GVNS* takes the longest to execute. This is attributed to its dynamic stopping criteria, which aim to achieve the best possible fitness convergence.

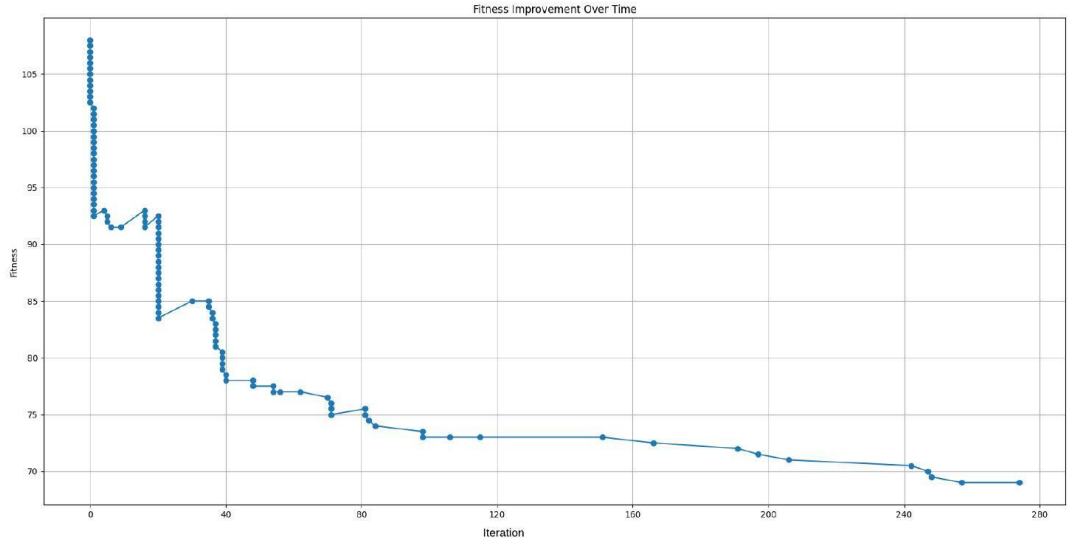
III.5.4.6 Evaluation of computational complexity

The evaluation of the computational complexity of the proposed algorithms focuses on the memory consumption and CPU cores utilized during their execution.

Memory Consumption The memory consumption of the algorithm is influenced by several factors, including the size of the grid, the number of nodes, and the specific algorithmic approach used. In general, the memory usage grows with the number of nodes and the complexity of the operations performed on each node. For example,



(a) *DYN_UCB_GVNS* Fitness Convergence



(b) *STAT_UCB_GVNS* Fitness Convergence

Figure III.7: Fitness Convergence Comparison between *DYN_UCB_GVNS* and *STAT_UCB_GVNS*

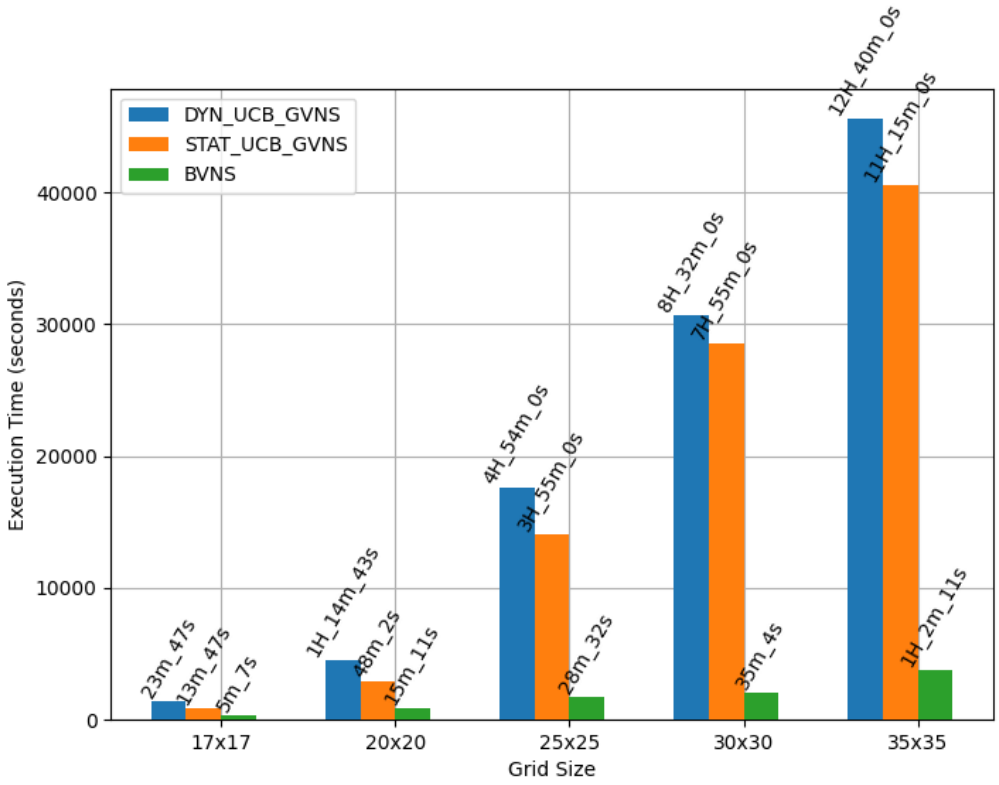


Figure III.8: Comparison of Execution Time for Different Methods

the dynamic approach (*DYN_UCB_GVNS*) requires additional memory to store the dynamically updated parameters and intermediate results, leading to higher memory consumption compared to the static approach (*STAT_UCB_GVNS*) and the basic approach *(BVNS*). The *UCB_GVNS* algorithm requires an average of *1.20 MB* for a 17x17 grid.

CPU Cores Utilized The CPU usage is a crucial aspect, especially when considering the execution time and efficiency of the algorithms. The *DYN_UCB_GVNS* algorithm, with its dynamic stopping criteria, requires more computational resources to explore the solution space thoroughly. This results in higher CPU usage and longer execution times compared to the *STAT_UCB_GVNS* and BVNS algorithms. The percentage of CPU cores utilized can vary based on the experimental setup, in our case for a 17x17 grid the CPU cores utilized averaged 25% of the 8 CPU cores available.

III.6 Conclusion

In this chapter, we have explored the implementation and experimental results of our proposed methods for optimal node placement in WSNs. Our primary focus was on evaluating the effectiveness of different approaches, namely *DYN_UCB_GVNS*, *STAT_UCB_GVNS*.

The implementation of our framework demonstrated its capability to handle the complexities associated with node placement problem in fenced areas WSNs. We began by detailing the optimal node placement problem and the model used for simulations. Our framework incorporated genetic algorithms for initial solution generation and utilized advanced techniques like the Reinforcement Learning method Upper Confidence Bound (UCB1) and General Variable Neighborhood Search (GVNS) for optimization.

The experimental results provided significant insights into the performance of each approach. Our simulations, conducted under various grid sizes and configurations, highlighted the strengths and weaknesses of each method. The dynamic stopping criteria of *DYN_UCB_GVNS* showed superior performance in terms of fitness improvement and efficient exploration of the solution space compared to the static stopping criteria and BVNS approach. This adaptive nature enabled better optimization, as evidenced by the higher fitness scores and improved performance metrics.

The comparative analysis also included evaluations based on metrics such as the number of relays, network diameter, average hops, and execution time. The results indicated that while BVNS performed well in terms of execution time, the RL-based approaches, particularly the *DYN_UCB_GVNS*, achieved better overall optimization results.

In summary, this chapter has validated the effectiveness of our proposed methods through comprehensive implementation and rigorous experimental evaluation. The findings underscore the potential of dynamic RL-based optimization techniques in addressing complex WSN deployment challenges, paving the way for more robust and efficient network designs. Future work can extend these methodologies to broader applications and further refine the adaptive mechanisms to enhance performance under varied conditions.

General Conclusion

This thesis presented a comprehensive investigation into the optimization of node placement in Wireless Sensor Networks (WSNs) using advanced techniques combining Reinforcement Learning (RL) and Meta-Heuristics. Our primary goal was to develop and evaluate novel methods to enhance network performance, considering key metrics such as the number of relays, network diameter, average hops, and execution time.

Our research began with a detailed exploration of existing methodologies for WSN deployment, highlighting the complexities and challenges involved. We reviewed various approaches and identified a lack of intelligence based approaches, establishing the need for more adaptive and efficient optimization techniques.

The core of our work introduced a hybrid framework incorporating Genetic Algorithms (GA) for initial solution generation, followed by optimization using Upper Confidence Bound (UCB1) and General Variable Neighborhood Search (GVNS). We proposed and implemented two primary approaches within this framework: *DYN_UCB_GVNS*, utilizing dynamic stopping criteria, and *STAT_UCB_GVNS*, based on static stopping criteria. Additionally, we evaluated the Basic Variable Neighborhood Search (BVNS) approach for comparative analysis.

The advancements made in enhancing Variable Neighborhood Search (VNS) performance through the integration of machine learning techniques present several promising avenues for future research:

1. **Refinement of Reinforcement Learning (RL) Algorithms**: Explore more sophisticated RL models and fine-tune hyperparameters to achieve greater improvements in solution quality and computational efficiency.

2. **Simplified Fitness Value and Network Diameter Calculations**: Develop methods that are computationally less expensive and easier to implement for precise calculations of fitness value and network diameter, significantly reducing the overall computational burden.

3. **Exploration of Different Neighborhood Structures**: Experiment with various neighborhood configurations to identify structures that provide a better balance between exploration and exploitation, leading to enhanced optimization results.

4. **Utilization of Model-Based Reinforcement Learning**: Implement model-based RL, which involves creating a model of the environment to simulate different scenarios, improving the efficiency and effectiveness of the search process.

5. **Application of Deep Reinforcement Learning (DRL) Methods**: Leverage deep neural networks in DRL to handle high-dimensional state spaces more effectively, potentially leading to breakthroughs in optimization performance.

6. **Real-World Implementation and Testing**: Validate theoretical results through real-world implementation and testing in various environments, adapting the algorithms to practical constraints and dynamic conditions.

These future efforts will not only enhance the robustness and versatility of the proposed solutions but also contribute to the evolving field of combinatorial optimization and its applications in diverse domains.

Abstract

Wireless Sensor Networks (WSNs) have become over the years a very attractive field of research. In fact, they had the attention of many researchers who have been interested in issues raised by these networks, such as energy, deployment, coverage, connectivity, latency, routing, etc. WSNs are particularly characterized by their miniaturized aspect, which makes them stealthy, and have rapid deployment in accessible or inaccessible zones. WSNs, which are considered an emerging technology, have a wide variety of applications in various fields such as military, health, transportation, agriculture, etc. In this thesis, we address the hybridization of Machine Learning (ML) tools and meta-heuristics to optimize the deployment of Relay Nodes (RNs) and Network Diameter (ND) in WSNs dedicated to the surveillance of sensitive fenced areas (e.g., oil/nuclear sites, airport, etc.). Consequently, we propose a novel hyper-heuristics, labeled *STAT_UCB_GVNS* and *DYN_UCB_GVNS*, that use Reinforcement Learning (RL) to guide the local search process, enhancing the effectiveness of the VNS algorithm in terms of solution quality. Experimental results demonstrate that our RL-based approach, achieves significant improvements in fitness value, specifically in the number of deployed RNs and ND, compared to Basic VNS (BVNS). Indeed, the proposed RL-based approach achieves an average fitness improvement of 49.97% while deploying an average of 53.144% fewer relays than BVNS.

Keywords: Wireless Sensor Networks, Deterministic Deployment, Coverage, Connectivity, Network Diameter, Multi-objective Combinatorial Optimization, Meta-Heuristics, Machine Learning, Reinforcement Learning,

الملخص

لقد أصبح مجال شبكات المستشعرات اللاسلكية (WSNs) على مر السنين مجالاً جذاباً للبحث. في الواقع، استحوذ على اهتمام العديد من الباحثين الذين اهتموا بالقضايا التي تثيرها هذه الشبكات، مثل الطاقة، النشر، التغطية، الاتصال، الكمون، التوجيه، وغيرها. تتميز شبكات المستشعرات اللاسلكية بشكل خاص بصغر حجمها، مما يجعلها خفية، وسريعة الانتشار في المناطق التي يمكن الوصول إليها أو التي يصعب الوصول إليها. تعتبر شبكات المستشعرات اللاسلكية تقنية ناشئة، ولها تطبيقات واسعة في مجالات مختلفة مثل العسكرية، الصحة، النقل، الزراعة، وغيرها. في هذه الأطروحة، نتناول تهجين أدوات التعلم الآلي (ML) والميتا-إرشادات لتحسين نشر عقد الترحيل (RNs) وقطر الشبكة (ND) في شبكات المستشعرات اللاسلكية المخصصة لمراقبة المناطق الحساسة المسيجة (مثل مواقع النفط/النووية، المطارات، وغيرها). بالتالي، نقترح نوعاً جديداً من الهايبر-إرشادات، تم تسميته STAT_UCB_GVNS وDYN_UCB_GVNS، الذي يستخدم التعلم المعزز (RL) لتوجيه عملية البحث المحلي، مما يعزز فعالية خوارزمية VNS من حيث جودة الحل. تُظهر النتائج التجريبية أن نهجنا القائم على التعلم المعزز يحقق تحسينات كبيرة في قيمة اللياقة، وخاصة في عدد عقد الترحيل المنتشرة وقطر الشبكة، مقارنة بخوارزمية VNS الأساسية (BVNS). في الواقع، يحقق النهج القائم على التعلم المعزز تحسناً متوسطاً في اللياقة بنسبة 49.97٪ بينما يقوم بنشر عدد أقل من العقد الترحيل بمتوسط 53.144٪ مقارنةً بـ BVNS.

الكلمات المفتاحية: شبكات المستشعرات اللاسلكية ، النشر الحتمي ، التغطية ، الاتصال ، قطر الشبكة ، تحسين المجموعات المتعددة متعددة الأهداف ، Meta-Heuristics ، التعلم الآلي ، التعلم المعزز ،

Bibliography

[1] Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on learning theory*, pages 39–1. JMLR Workshop and Conference Proceedings, 2012.

[2] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[3] Mohammad Abu Alsheikh, Shaowei Lin, Dusit Niyato, and Hwee-Pink Tan. Rate-distortion balanced data compression for wireless sensor networks. *IEEE Sensors Journal*, 16(12):5072–5083, 2016.

[4] BRIAN AMADIO. Multi-armed bandits and the stitch fix experimentation platform. https://multithreaded.stitchfix.com/blog/2020/08/05/bandits/, August 05, 2020. Accessed: 2024-05-20.

[5] TREVOR AMESTOY. Clustering basics and a demonstration in clustering infrastructure pathways. https://waterprogramming.wordpress.com/2022/03/16/clustering/basics/and/a/demonstration/in/clustering/infrastructure/pathways/, March 16, 2022.

[6] Slimane Charafeddine BENGHELIMA Amira BENTABET. L'optimisation du déploiement d'un rcsf pour une application de surveillance de sites. Master's thesis, Université Belhadj Bouchaib D'Ain Témouchent, Ain Témouchent, ALGERIA, 2019.

[7] Kasyful Amron, Wuryansari M Kusumawinahyu, Syaiful Anam, and Wayan F Mahmudy. Reliable wireless sensor network planning with multipath topology through relay placement optimization. *Journal of Robotics and Control (JRC)*, 5(2):471–481, 2024.

[8] Ibrahim Atoui, Abdallah Makhoul, Samar Tawbe, Raphaël Couturier, and Abbas Hijazi. Tree-based data aggregation approach in periodic sensor networks using correlation matrix and polynomial regression. In *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pages 716–723. IEEE, 2016.

[9] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

[10] Babak Badnava, Mona Esmaeili, Nasser Mozayani, and Payman Zarkesh-Ha. A new potential-based reward shaping for reinforcement learning agent. In *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 01–06. IEEE, 2023.

[11] Stefano Basagni, M Yousof Naderi, Chiara Petrioli, and Dora Spenza. Wireless sensor networks with energy harvesting. *Mobile Ad Hoc Networking: Cutting Edge Directions*, pages 701–736, 2013.

[12] batserve. Optimization toolbox 3 user's guide. PDF document, 2010. Accessed on 04/05/2024.

[13] Ghulam Bhatti. Machine learning based localization in large-scale wireless sensor networks. *Sensors*, 18(12):4179, 2018.

[14] Stephen Boyd and Jacob Mattingley. Branch and bound methods. *Notes for EE364b, Stanford University*, 2006:07, 2007.

[15] Vladimír Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45:41–51, 1985.

[16] Ryan Champlin. Selection methods of genetic algorithms. Student Scholarship - Computer Science, 2018.

[17] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. *Advances in neural information processing systems*, 24, 2011.

[18] Li Cui, Fei Wang, Haiyong Luo, Hailing Ju, Li, and Tianpu. A pervasive sensor node architecture. In *Network and Parallel Computing: IFIP International Conference, NPC 2004, Wuhan, China, October 18-20, 2004. Proceedings*, pages 565–567. Springer, 2004.

[19] Subham Datta. Branch and bound algorithm. https://www.baeldung.com/cs/branch-and-bound/, 2020. Accessed: 02/05/2024.

[20] Issa Dembele and Abdel Razak Hamadou Adamou. L'optimisation du déploiement d'un rcsf utilisant un modèle radio réaliste, dédié à la surveillance d'un site sensible clos. Master's thesis, Université Belhadj Bouchaib d'Ain Témouchent, Ain Témouchent, Algeria, 2020.

[21] Dingzhu Du, Panos M Pardalos, Xiaodong Hu, and Weili Wu. *Introduction to Combinatorial Optimization*. Springer, 2022.

[22] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[23] Soumaya Fellah. *Optimisation Multi-objectif appliquée au déploiement et à la performance des réseaux de capteurs sans fil*. PhD thesis, Université d'Oran1-Ahmed Ben Bella, Avril 2018.

[24] Samuel Gabrielsson. A parallel tabu search alglorithm for the quadratic assignment problem. Master's thesis, Luleå University of Technology, 2007.

[25] Vera Galishnikova and Peter Pahl. Constrained construction of planar delaunay triangulations without flipping. *Structural Mechanics of Engineering Constructions and Buildings*, 14:154–174, 12 2018.

[26] Rakesh Chandra Gangwar and Roohi Singh. Machine learning algorithms from wireless sensor network's perspective. In Jaydip Sen, Mingqiang Yi, Fenglei Niu, and Hao Wu, editors, *Wireless Sensor Networks*, chapter 3. IntechOpen, Rijeka, 2023.

[27] GeeksforGeeks. 0-1 knapsack using branch and bound. https://www.geeksforgeeks.org/0-1-knapsack-using-branch-and-bound/, 24/05/2024.

[28] GeeksforGeeks. Crossover in genetic algorithm. https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/, Accessed: 07/05/2024.

[29] F Gembicki and Y Haimes. Approach to performance and sensitivity multiobjective optimization: The goal attainment method. *IEEE Transactions on Automatic control*, 20(6):769–771, 1975.

[30] James M Gilbert and Farooq Balouchi. Comparison of energy harvesting systems for wireless sensor networks. *International Journal of automation and computing*, 5:334–347, 2008.

[31] Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[32] Pierre Hansen and Nenad Mladenovic. A tutorial on variable neighborhood search. *Les Cahiers du GERAD ISSN*, 711:2440, 2003.

[33] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[34] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[35] Olga Isakova. Application of machine learning algorithms for classification and regression problems for mobile game monetization. Master's thesis, 2019.

[36] Mehdi Jaoua. *Algorithme de recherche tabou pour la planification optimale d'une campagne marketing sur les moteurs de recherche*. PhD thesis, École Polytechnique de Montréal, 2014.

[37] Khalid Jebari, Mohammed Madiafi, et al. Selection methods for genetic algorithms. *International Journal of Emerging Sciences*, 3(4):333–344, 2013.

[38] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. An introduction to reinforcement learning. *The Biology and Technology of Intelligent Autonomous Agents*, pages 90–127, 1995.

[39] Jaewoong Kang, Jongmo Kim, and Mye M Sohn. Supervised learning-based lifetime extension of wireless sensor network nodes. *J. Internet Serv. Inf. Secur.*, 9(4):59–67, 2019.

[40] Holger Karl and Andreas Willig. *Protocols and architectures for wireless sensor networks*. John Wiley & Sons, 2005.

[41] Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On bayesian upper confidence bounds for bandit problems. In *Artificial intelligence and statistics*, pages 592–600. PMLR, 2012.

[42] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[43] Volodymyr Kuleshov and Doina Precup. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*, 2014.

[44] Anit Kumar. Encoding schemes in genetic algorithm. *International Journal of Advanced Research in IT and Engineering*, 2(3):1–7, 2013.

[45] Rihem LARBI. Déploiement optimal des nœuds de capteurs employant le clustering k-means et un algorithme génétique. Master's thesis, ÉCOLE DE TECHNOLOGIE SUPÉRIEURE UNIVERSITÉ DU QUÉBEC, 2021.

[46] James LeDoux. Bandit algorithms: Epsilon-greedy, ucb, and exp3 in python. https://jamesrledoux.com/algorithms/bandit-algorithms-epsilon-ucb-exp-python/, March 24, 2020.

[47] Piero Macaluso. *Deep Reinforcement Learning for Autonomous Systems*. PhD thesis, Politecnico di Torino, 2020.

[48] Ashima Malik. A study of genetic algorithm and crossover techniques. *International Journal of Computer Science and Mobile Computing*, 8(3):335–344, 2019.

[49] Mustafa Mısır, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. An intelligent hyper-heuristic framework for chesc 2011. In *International Conference on Learning and Intelligent Optimization*, pages 461–466. Springer, 2012.

[50] Anis Mjirda, Raca Todosijević, Saïd Hanafi, Pierre Hansen, and Nenad Mladenović. Sequential variable neighborhood descent variants: an empirical study on the traveling salesman problem. *International Transactions in Operational Research*, 24(3):615–633, 2017.

[51] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[52] Francesca Nuzzo. *Sanity checks for explanations of deep neural networks predictions*. PhD thesis, Politecnico di Torino, 2020.

[53] Mohammad S. Obaidat and Sudip Misra. *Introduction to wireless sensor networks*, page 1–13. Cambridge University Press, 2014.

[54] D.Y. Patil College of Engineering. Wsn challenges, required mechanism, attacks, and applications of wireless sensor networks (wsns) in disaster condition, 2023. Accessed: 2024-05-19.

[55] Erik Persson. Energy harvesting in wireless sensor networks, 2019.

[56] Gregory J Pottie and William J Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[57] Rong Qu, Ying Xu, and Graham Kendall. A variable neighborhood descent search algorithm for delay-constrained least-cost multicast routing. In *Learning and Intelligent Optimization: Third International Conference, LION 3, Trento, Italy, January 14-18, 2009. Selected Papers 3*, pages 15–29. Springer, 2009.

[58] Priyanka Rawat, Kamal Singh, Hakima Chaouchi, and Jean-Marie Bonnin. Wireless sensor networks: A survey on recent developments and potential synergies. *The Journal of Supercomputing*, 68, 04 2013.

[59] Priyanka Rawat, Kamal Deep Singh, Hakima Chaouchi, and Jean Marie Bonnin. Wireless sensor networks: a survey on recent developments and potential synergies. *The Journal of supercomputing*, 68:1–48, 2014.

[60] Maher Rebai, Hichem Snoussi, Faicel Hnaien, Lyes Khoukhi, et al. Sensor deployment optimization methods to achieve both coverage and connectivity in wireless sensor networks. *Computers & Operations Research*, 59:11–21, 2015.

[61] Yousef Sardahi. *Multi-objective optimal design of control systems*. PhD thesis, University of California, Merced, 2016.

[62] Claudio Savaglio, Pasquale Pace, Gianluca Aloi, Antonio Liotta, and Giancarlo Fortino. Lightweight reinforcement learning for energy efficient communications in wireless sensor networks. *IEEE Access*, 7:29355–29364, 2019.

[63] Noman Shabbir and Syed Hassan. *Routing Protocols for Wireless Sensor Networks (WSNs)*. 10 2017.

[64] Anupam Singh. Reinforcement learning based empirical comparison of ucb, epsilon-greedy, and thompson sampling. *Int. J. of Aquatic Science*, 12(2):2961–2969, 2021.

[65] Richard S Sutton, Andrew G Barto, et al. Introduction to reinforcement learning. vol. 135, 1998.

[66] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[67] Haluk Rahmi Topcuoglu, Abdulvahid Ucar, and Lokman Altin. A hyper-heuristic based framework for dynamic optimization problems. *Applied Soft Computing*, 19:236–251, 2014.

[68] Frantz Tossa, Wahabou Abdou, Eugene C Ezin, and Pierre Gouton. Improving coverage area in sensor deployment using genetic algorithm. In *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part V 20*, pages 398–408. Springer, 2020.

[69] Walid TOUIL. Optimization of constrained relay node deployment using a metaheuristic. Mémoire de master, Ain Temouchent – University Belhadj Bouchaib, 2022/2023.

[70] TutorialsPoint. Genetic algorithms - parent selection. https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm, Accessed: 07/05/2024.

[71] V7 Labs. Supervised vs unsupervised learning. https://www.v7labs.com/blog/supervised-vs-unsupervised-learning, Accessed: 2024-05-05.

[72] Yun Wang, Gary Attebury, and Byrav Ramamurthy. A survey of security issues in wireless sensor networks. *IEEE Communications Surveys & Tutorials*, 8(2):2–23, 2006.

[73] Muhammad Rhifky Wayahdi, Subhan Hafiz Nanda Ginting, and Dinur Syahputra. Greedy, a-star, and dijkstra's algorithms in finding shortest path. *International Journal of Advances in Data and Information Systems*, 2(1):45–52, 2021.

[74] Alexander J. Williams, Matheus F. Torquato, Ian M. Cameron, Ashraf A. Fahmy, and Johann Sienz. Survey of energy harvesting technologies for wireless sensor networks. *IEEE Access*, 9:77493–77510, 2021.

[75] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer networks*, 52(12):2292–2330, 2008.

[76] Farah Ayiesya Zainuddin, Md Fahmi Abd Samad, and Durian Tunggal. A review of crossover methods and problem representation of genetic algorithm in recent engineering applications. *International Journal of Advanced Science and Technology*, 29(6s):759–769, 2020.