Thesis submitted to the

UNIVERSITY OF MOHAMED BOUDIAF – MSILA

FACULTY OF MATHMATICS AND COMPUTER SCIENCE
DEPARTEMENT OF COMPUTER SCIENCE

In partial fulfillment of the requirements for the degree of

Master in **Computer science**

Specialty: Information Systems and Software Engineering

By

HADJI Ayat

Title of the thesis

A POPULATION-BASED ITERATED GREEDY ALGORITHM FOR MAXIMIZING WIRELESS SENSOR NETWORK LIFETIME

Under the supervision of

LAKEHAL-AYAT Raouf Ouanis

Composition of the jury

Dr. ATTIR Azedine	University of Msila	President
Dr. LAKEHAL-AYAT Raouf Ouanis	University of Msila	Supervisor
Dr. GUESMIA Salah	University of Msila	Examiner

June, 2023

Acknowledgement

I would like to express my heartfelt gratitude to the Almighty Allah, whose unwavering strength enabled me to successfully complete this year. I firmly believe that my accomplishments are solely attributed to his blessings.

I am deeply fortunate to have been under the supervision of Dr. Lakhal AYAT, whose profound guidance and exceptional proposal on this topic paved the way for my research journey. I am sincerely grateful for his invaluable advice and unwavering support throughout the entire process.

I extend my sincere appreciation to Dr. Attir and Dr. Geusmia for graciously accepting to serve on the panel of the jury for my dissertation defense. Their expertise and contribution added immense value to the evaluation of my work.

Furthermore, I am immensely thankful to Miss Lounas Selma for her unwavering guidance and insightful advice throughout the course of my work. Her expertise was instrumental in shaping the outcome of this work.

Lastly, I owe immeasurable gratitude to my beloved family: My mother, My father, Dr. Ikhlass, Dr. Toka, Dr. Maroua and Sondoss for their unwavering support and encouragement during the past five years. Their constant presence and belief in my abilities have been a tremendous source of strength and motivation.

Table of Contents

List of Figures

List of Abbreviations

DDS: Disjoint Dominating Set

EA: Evolutionary Algorithms

IG: Iterated Greedy

MDDS: Maximum Disjoint Dominating Sets

MWDDS: Maximum Weighted Disjoint Dominating Sets

OP: Optimization Problem

PBIG: Population Based Iterated Greedy

RG: Random Graph

SA: Simulated Annealing

SLS: Stochastic Local Search

WSNs: Wireless Sensor Networks

List of Tables

List of Equations

GENERAL INTRODUCTION

Wireless Sensor Networks (WSNs) are formed of a collection of spatially distributed entities that communicate wirelessly, which are called sensor nodes. The lifespan of a wireless sensor network is heavily reliant on the energy usage of the sensors, considering the constrained availability of energy resources such as: Environmental Monitoring, Entertainment Security, Emergencies, Smart World, Medical and Health [1].

These sensor nodes are composed of three main parts: a sensing unit, a processing unit and a communication unit (radio communication system). Sensors sense data using the first unit, then subsequently, the processed data generated by the device's processing unit is wirelessly transmitted to other nodes, or a designated central destination known as the Base Station. The devices possess restricted storage capacity and operate on a constrained power supply; hence, many power-saving techniques are used. The remaining lifespan for a device before its power is depleted is computed as the battery capacity (Watt hours) divided by the average power depletion (Watts). Nonetheless, the energy consumption of a sensor node is subjected to multiple factors. Thus, the extension of the network’s lifetime in parallel with achieving a reliable communication and effective sensing coverage is a challenging problem in this field.

A network lifetime denotes the time in which the network is fully operational and capable of executing its designated tasks. Hence, the remaining lifetime of a wireless sensor network (WSN) is heavily reliant on the energy usage of its sensors, considering the constrained availability of energy resources. To optimize the network's lifetime, several mechanisms were classified under the following classes according to Cardei [2]:

- Duty cycling: The sensor nodes switch between two modes: sleep mode and awake mode (active).
- Power control: By fine-tuning the transmission range of the communication unit.
- Efficient routing and data gathering techniques.
- Minimize data transmissions and eliminate non-essential activities.

In our study, the used technique for extending and enhancing the WSN lifetime falls under the first class; this technique uses the conception of graph theory in the

following way. The graph represents the network, and its nodes (vertices) are the sensor devices of this network. If two devices are able to communicate with each other using the communication unit, then the corresponding nodes are connected by an edge. Consider that sensor entities have two main functionalities that are:

- Data sensing with less energy consumption.
- Data processing and forwarding to the base station with more energy consumption.

This study employs the notion of dominating sets, in which a set of nodes called dominators are responsible for covering the whole network. In our context, the dominators perform the tasks that consume more energy, including data processing and forwarding to the base station. On the other hand, the rest nodes only perform the task of sensing data which consumes less energy. The concept of sleep and awake mode is observed here. Any node out of the dominating set must be dominated; that is, it has at least one neighbor from the dominating set. In this model the nodes are all switched on, even the dominated nodes (that is, the nodes out of the dominating set that are dominated by at least one node that forms a part of the dominating set) because the task of sensing data is performed by all network nodes. However, only dominators that play the role of cluster heads are responsible for performing the task of forwarding and processing the data.

Our study aims to effectively form the maximum number of Disjoint Dominating Sets (DDS) so that they are used consecutively to play the role of a cluster head. This problem is known as the maximum weighted disjoint dominating sets problem (MWDDS). Moreover, we will implement this technique using one of the most known heuristic optimization algorithms, the iterated greedy algorithm, and make use of the population-based algorithms technique which it a Population Based Iterated Greedy (PBIG) algorithm. It works to find the DDS as follows:

An initial solution, which is in this case a population of solution, and in which the solution is a set of DDS, is generated using a greedy heuristic method. Then iteratively, the algorithm goes through two main phases:

1- The destroying phase in which the initial solution is partially destroyed and some elements of the current solution are removed, and

2- The construction phase which uses of a randomized greedy heuristic to build the partial solution resulting from the previous phase, hopefully, to get a complete solution.

The algorithm keeps iterating over these two main phases until a stopping criterion is met. Then, the best solution in the final generated population is returned, and it represents the best combination of disjoint dominating sets. Furthermore, for the experimental evaluation of this algorithm, we used a set of benchmark instances, which are problem-specific instances generated for the problem of maximizing a sensor network lifetime.

This dissertation is presented in four chapters, Chapter 01 provides a concise overview of optimization problems and their classifications. It covers the formal optimization problem definitions, different classifications of optimization problems, some basic concepts of complexity classes (including P and NP), and a classification of optimization algorithms. Chapter 02 provides an outline of the basics of graph theory and the notions which are used in the problem of MWDDS, in addition to the problem statement description with an example. Chapter 03 explains the concept of the population-based iterated greedy algorithm (PBIG), the detailed description of the algorithm, and the side procedures that we implemented in this study. Finally, Chapter 04 demonstrates the experimental evaluation of the performance of this algorithm on a set of know problem instances.

CHAPTER 01: COMBINATORIAL OPTIMIZATION PROBLEMS

INTRODUCTION

Optimization is a crucial field of applied mathematics because of its diverse range of applications and the existence of effective algorithms. It involves finding the best possible solution to a problem by minimizing or maximizing a specific objective function, which depends on several decision variables. Additionally, the solution must also satisfy functional constraints. Optimization can be used in various fields to help achieve the most desirable outcome given the available resources and limitations. The availability of efficient algorithms to quickly and effectively solve optimization problems further highlights the significance of optimization [3]. An overview on the basics of optimization field will be covered in this chapter, including the optimization problem's definition and elements, different classifications of optimization problems according to different factors, classes of computational complexity (P and NP), and optimization methods and algorithms to solve optimization problems.

1. Optimization Problems

An Optimization Problem (OP) is a problem that can be solved using optimization methods, and which seek to find the best solution among a large set of possible solutions. In other words, an optimization problem involves finding the optimal solution that achieves the best possible outcome from a set of available options. These problems can arise in many different fields and can be addressed using various optimization techniques. Its goal is to find the solution that provides the most desirable results while satisfying any constraints or limitations that may be present [4].

2. Statement of an Optimization Problem

2.1.Design Vector

A system can be characterized by a collection of values, some of which can be modified in the design process, while others are predetermined by external factors or constraints. The adjustable values are referred to as design or decision variables, and they are collated

into a design vector x, while the fixed values are known as parameters or environmental restrictions [5].

2.2.Design Constraints

When designing a system, the selection of design variables must adhere to specific criteria and cannot be chosen randomly. These criteria are referred to as design constraints, which may be limitations on the system's performance or behavior, as well as physical constraints. In other words, there are certain rules and limitations that must be taken into consideration when selecting design variables to ensure that the resulting system functions appropriately and efficiently [5].

2.3.Objective Function

The classical design procedure is aimed at creating a design that meets all necessary requirements. While there may be several design options that are acceptable, optimization is used to identify the best possible design. To compare different designs, a criterion known as the objective function is chosen and expressed as a function of the design variables. This function is typically determined by considering physical or economic factors. However, choosing the right objective function can be challenging due to the inapplicability of a given optimal design with respect to all criteria. For example, there may be a tradeoff between performance and cost, or between performance and reliability. Thus, the selection of the objective function is a critical decision in the design process.

In cases in which multiple criteria need to be satisfied, a multi-objective optimization problem arises, which can be approximately solved using a cost function that weighs several objective functions. Essentially, the cost function is a mathematical equation that assigns different weights to each objective function based on its relative importance. By finding the optimal values of the design variables that minimize the cost function, the designer can achieve a design that satisfies multiple criteria simultaneously. Despite its technical nature, this approach is often necessary to achieve the best possible outcome for a given design problem [5].

3. Formal Definition of an Optimization Problem

An optimization problem maximizes or minimizes a function that is relative to a given set, and represents a range of choices available in a certain situation. The function enables comparing the different choices to determine the best choice. In this work, we used the following notations:

- x is the vector of variables, also called parameters, with $x \in S$ and $S \subseteq R^n$, and S being the feasible set, which refers to the set of possible choices that are considered acceptable or suitable for x.

- f is the objective function, a scalar function (i.e., one which returns a single value from an input value of x that we want to maximize or minimize, with $f : R^n \rightarrow R$.

- c_i are constraint functions, which are scalar functions of x that define certain equations and inequalities that the vector x must satisfy.

We define the optimization problem (maximizing or minimizing problem) more formally as follows. We have a minimization problem formed as:

$$\min_{x \in R^n} f(x) \qquad subject\ to \qquad c_i(x) = 0,\ \ i \in \varepsilon$$
$$c_i(x) \geq 0,\ \ i \in \gamma$$

In which γ and ε are sets of indices for equality and inequality constraints, respectively.

And we have: $\max_{x \in S} f(x) = -\min_{x \in S} f(-x)$. [6]

4. Optimization Problems Classification

The general optimization problems can be classified based on several factors:

4.1. Objective Classification

4.1.1 Single Objective

In situations in which the objective is to identify the optimal solution, optimization techniques are often employed to minimize or maximize an objective function that captures all possible goals. This type of optimization is useful as a decision-making tool since it enables a better understanding of the problem. However, it should be noted that this approach is typically unable to provide a set of alternative solutions that balance multiple

objectives. Therefore, it may be necessary to consider other criteria in addition to the objective function in order to achieve the desired outcomes [7].

4.1.2 Multiple Objective

When there are multiple conflicting objectives, it is often not possible to identify a single solution that can optimally but simultaneously satisfy all objectives. This is due to the fact that the interaction between different objectives can lead to complex trade-offs. Instead, the problem often requires simultaneous optimization of multiple objective functions, which can lead to a set of solutions that represent a compromise among the conflicting objectives [7].

4.2. Problem Classification

4.2.1. Constrained and Unconstrained

The presence or absence of constraints is an important aspect of optimization problems. While some argue that all real-world problems involve constraints in the form of constraint functions or variable bounds, the study of unconstrained optimization problems remains significant. This is because many optimization algorithms tackle constrained problems by transforming them into unconstrained problems, or a sequence of unconstrained problems. Moreover, several techniques used for solving unconstrained problems can be naturally extended to develop solution procedures for constrained problems, and can serve as a source of inspiration for such problems. [8]

4.3. Variable Classification

In optimization, variables can be classified into three main types: continuous, discrete and mixed variables.

4.3.1. Continuous Variables

In optimization problems with continuous variables, the variables are permitted to assume any value from a specific range of values, which is usually a set of real numbers. It is possible for some or all of the variables to be restricted to binary values, meaning that they can only take on the values of 0 or 1. Additionally, some variables may be limited to integer values, or values from sets with a finite number of elements [9].

4.3.2. Discrete

In discrete optimization, some or all of the variables in a given model are necessarily comprised in a discrete set. This differs from continuous optimization, in which variables may take on any value within a given range. Discrete optimization can be further divided into two subfields: integer programming, in which the discrete set consists of a subset of integers, and combinatorial optimization, in which the discrete set is a set of objects, or combinatorial structures. Examples of combinatorial structures include assignments, combinations, routes, schedules, and sequences [9].

The optimization problem with integer or discrete variables is termed as combinatorial problem. In combinatorial problems, we seek an object from a finite or infinite set-typically an integer, set, permutation, or graph.

4.3.3. Mixed Variables

They involve both continuous and discrete variables.

4.4. Function classification

4.4.1. Linear or Nonlinear Functions

A model is called a linear programming model or linear model if all its functions are linear. However, if one or more of its functions involve nonlinearity, it is referred to as a nonlinear model. Solving nonlinear models is typically more complex than solving linear models, and different solution approaches are required. While an unconstrained problem with a single linear objective function is not considered an interesting optimization problem, unconstrained nonlinear optimization problems are the focus of many research studies and applications [10].

4.4.2. Convex or Non-convex Functions

In classical optimization, the property of convexity is highly valued because many optimization techniques and algorithms are built on the assumption that the objective function is convex. Convexity ensures that the optimization problem has a unique optimal solution that can be found efficiently and reliably. On the other hand, non-convex optimization problems can be more challenging to solve, as they may have multiple optimal solutions or no optimal solution at all. Therefore, identifying whether a function is convex

or non-convex is crucial for selecting the appropriate optimization method to solve the problem [11].

4.4.3. Differentiable or Non-differentiable Functions

The methods used to find solutions in optimization can be categorized into two groups: those that involve derivatives and those that do not – derivative-free methods –. When using derivative-based techniques, it is necessary for the function to be differentiable.

4.5. Nature of Optimization

4.5.1. Global or Local Optimum

The process of local optimization focuses on finding the optimal solution within a particular section of the search space. This may involve identifying the global optimum in cases in which there are no present local optima [12]. Global optimization refers to the process of finding the best possible solution to a problem, considering all possible values of the variables and taking into account the existence of local optima. In other words, it aims to identify the optimal solution across the entire search space, rather than settling for a suboptimal solution due to the presence of local optima [13].

When the objective function has a single optimum or the search region is known to contain the global optima, a local optimization algorithm is suitable. On the other hand, when the structure of the objective function response surface is poorly understood or when the function contains local optima, a global optimization algorithm should be used [13].

4.5.2. Deterministic or stochastic

Optimization problems can be also classified into two main categories: deterministic programming problems and stochastic programming problems. Deterministic programming problems are defined as those in which the input variables produce the same output every time. In this type of problem, all design variables are deterministic. In contrast, stochastic programming problems are those in which some or all of the design variables are expressed in probabilistic terms, i.e., they are non-deterministic or stochastic. The use of stochastic methods in optimization provides the benefit of circumventing the problem of being stuck at local minima, and also the capability of producing solutions for complicated problems [14].

5. Computational complexity

The two essential concepts required to solve a problem that any algorithm needs are time and space.

- To evaluate the efficiency of an algorithm in solving a problem of size n, we consider its time complexity, which refers to the number of the required steps. Typically, this complexity is expressed in terms of the worst-case scenario [15].
- The space complexity refers to the quantity of memory (typically on a computer) that is required to solve a problem [16].

When determining the computational complexity of an algorithm, the objective is not to obtain an exact count of steps, but rather to derive an asymptotic upper bound on the number of steps required.

The computational complexity of a problem is determined through the complexity of the most efficient algorithm solving the problem. That is, if a problem can be solved by an algorithm in polynomial time, it is considered tractable or easy. Conversely, if no polynomial-time algorithm exists to solve a problem, it is considered intractable or difficult.

6. Complexity classes

Categorizing problems into complexity classes is an essential part of computational theory. These classes group problems that can be solved using a specific amount of computational resources. The classification is based on two important classes: P and NP. These classes are used to define the computational complexity of a problem and determine whether a problem can be solved efficiently by an algorithm [15].

6.1. Class P (Polynomial Time)

The complexity class P is composed of decision problems that can be solved by polynomial-time algorithms. A decision problem is a computational problem that requires a yes/no answer. In P, polynomial time refers to the maximum amount of time it takes to solve a problem being a polynomial function of the problem size. For each problem in P, there is a corresponding algorithm that can solve any given instance of the problem within

a time complexity that is bounded by $O(n^k)$. Thus, all problems in P can be solved efficiently even in the worst case. It is important to note that P is widely acknowledged as a class of decision problems with practical and feasible efficient decision procedures [16].

6.2. Class NP (non-deterministic polynomial time)

NP is a class of decision problems that can be solved in polynomial time using a non-deterministic algorithm. The majority of problems that appear to be difficult to solve when they are formulated as decision problems belong to this class [17]. S. Bouamama explained the meaning of non-deterministic in [16] by stating "that no particular rule is followed to make the guess. Note that P is a subset of NP, $N \subseteq NP$."

6.2.1. Class $NP - Complete$

$NP - Complete$ is a term to describe that a decision problem $A \in NP$, and all other decision problems of the NP class can be transformed into A using a polynomial-time algorithm. Simply put, A is the hardest problem in NP, and if a polynomial algorithm is found to solve it, then all other problems in NP can also be solved in polynomial time. [15]

6.2.2. Class $NP - Hard$

This class includes those problems which are not included in NP but, any of these problems is capable to be transformed to NP problem in polynomial time. The class is labeled as hard in the sense of any problem out of NP is at least as hard as any NP problems.

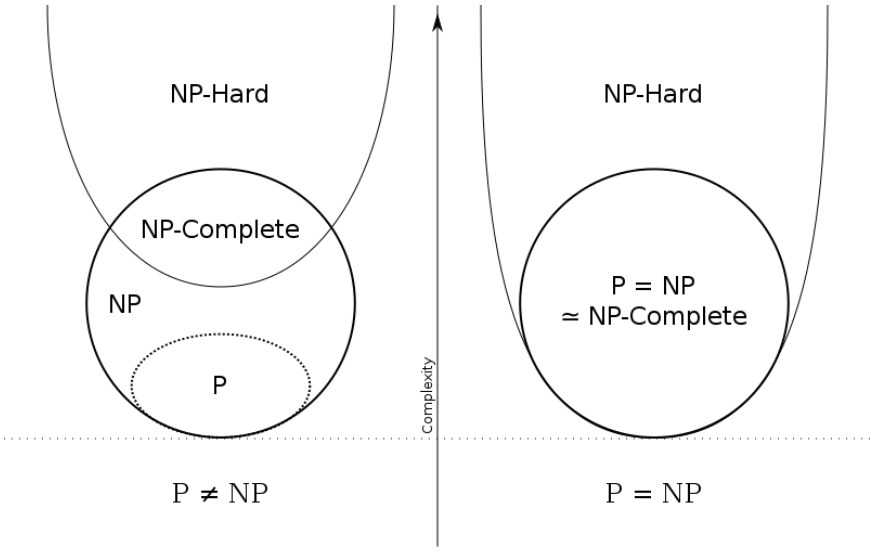

***Figure 1**: The relation between complexity classes [18]*

7. Optimization Method

The resolution of a problem's complexity can be accomplished using either an exact or an approximate method. The exact method provides an optimal solution although it is computationally expensive since it cannot solve $NP - complete$ problems. On the other hand, approximate algorithms (or heuristics) generate high-quality solutions in a reasonable amount of time, but there is no guarantee that a globally optimal solution can be obtained**.**

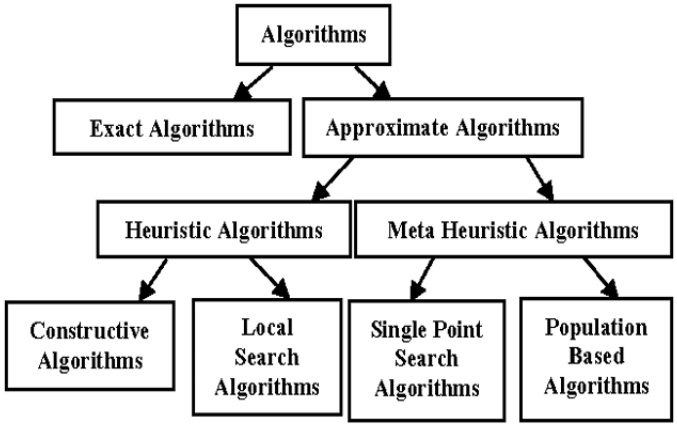

Figure 2: *Optimization Methods Classification [19]*

7.1. Exact

Exact methods aim to find the optimal solution by exhaustively exploring the search space, either explicitly or implicitly. Although they guarantee optimal solutions, their computational time becomes prohibitively long as the problem size and the number of objectives to optimize increase. This limits the use of such methods for small problems. Branch & bound, Branch & cut, and Branch & price are general methods used for solving optimization problems, while linear programming in integers and the A^* algorithm are less general. There are also problem-specific methods like Johnson's algorithm for scheduling. Dynamic programming, the branch and X family of algorithms, constraint programming, and the A^* family of search algorithms are some of the classical algorithms found in the class of exact methods.

7.2. Approximate

Approximate methods, also known as heuristics, are methods that aim to generate solutions of high quality within a reasonable amount of time. Unlike exact methods, they do not guarantee finding the globally optimal solution. Approximate algorithms can be divided into two categories: specific heuristics, which are designed for a particular problem, and metaheuristics, which are more general and can be applied to a wide range of problems. While approximate algorithms do not guarantee optimality, they are often preferred in practice due to their ability to quickly find solutions that are close to optimal [15].

7.2.1. Approximate algorithms

Approximation algorithms provide a way to obtain a solution for an optimization problem that is not guaranteed to be globally optimal, but has a provable bound on how far the solution is from the optimal one. This bound is expressed in terms of a factor called the approximation ratio, which measures the quality of the solution relative to the optimal solution. The smaller the approximation ratio, the closer the solution is to the optimal one. This property makes approximation algorithms a useful tool for solving large-scale optimization problems in which finding the exact solution is computationally infeasible [15].

7.2.2. Heuristic Algorithms

Heuristics are designed and applicable to a particular optimization problem. The popularity of heuristics has surged due to their capability of providing feasible solutions within reasonable computational time for optimization problems [15]. Heuristic algorithms are used to solve NP problems and decrease the time complexity of problems by giving quick solutions.

In this type of Algorithm, we distinguish two types:

- Constructive Algorithms which are algorithms that build a solution step-by-step, starting from an empty or initial solution.
- Local Search Algorithms, on the other hand, aim to improve an initial solution iteratively by making small modifications to it.

7.2.3. Metaheuristics

Metaheuristics are a set of techniques used to guide the search process in order to efficiently explore the search space and find optimal or near-optimal solutions. These techniques vary from simple local search methods to complex learning algorithms.

Metaheuristic algorithms are often approximate and non-deterministic, but may include strategies to avoid getting stuck in local optima. The basic concepts of metaheuristics can be described abstractly, and they are not specific to a particular problem. Domain-specific knowledge may be incorporated into metaheuristics through the use of problem-specific heuristics that are controlled by the overall strategy. Advanced metaheuristics may incorporate a search experience that is stored in memory to guide the search process [20].

- Single Solution Based Metaheuristics is one that operates on a single solution and iteratively searches for an improved solution within the solution space.
- Population Based Metaheuristics, on the other hand, operates on a population of solutions rather than a single solution. These algorithms maintain a diverse set of candidate solutions and use various techniques to explore and exploit the search space efficiently.

8. Greedy Algorithms

Optimization problems typically require making a sequence of decisions, and dynamic programming can be utilized to determine the optimal decision at each step. However, for many optimization problems, there exist simpler and more efficient algorithms. Greedy algorithms adopt a straightforward strategy of selecting the best feasible decision at each step, considering only the information available locally. By iteratively making these choices, the algorithm aims to attain an optimal solution on a global scale [21]. A wide application of greedy algorithms are found in combinatorial optimization, in which they are subjected to rigorous worst-case analysis in many cases. These algorithms are relatively simple to describe and implement, and they are computationally efficient.

Greedy algorithms operate by selecting the locally optimal choice at each step, to eventually obtain a globally optimal solution. They start with an empty solution and iteratively add the best available element until the solution is complete. At each step, the

algorithm evaluates the available choices and selects the one that maximizes (or minimizes) some predefined objective function or criteria. Greedy algorithms are often used in situations where finding the optimal solution is too expensive and an acceptable solution is satisfactory. However, the greedy approach is not always guaranteed to produce an optimal solution and may perform sub-optimally or fail entirely in some cases. Consequently, careful consideration of the problem and algorithm design is necessary to ensure its efficacy [22].

9. Stochastic Local Search

Stochastic local search (SLS) methods are widely recognized as one of the leading and effective approaches for tackling computationally challenging problems across various domains [23]. They have gained widespread popularity as an effective approach to tackle computationally difficult decision and optimization problems across various fields, such as computing science, operations research, engineering, chemistry, biology, and physics.

SLS encompasses diverse techniques, ranging from simple constructive and iterative improvement procedures to more advanced methods, such as simulated annealing (SA), iterated local search, and evolutionary algorithms (EAs). These algorithms are characterized as iterative improvement procedures that solely accept neighboring solutions that are an improvement over the current candidate solution. The algorithm execution terminates when a local optimum has been reached [23].

9.1. Iterated Greedy

The Iterated Greedy (IG) algorithm is a Stochastic Local Search (SLS) method that utilizes a sequence of solution construction procedures. The algorithm starts by generating a complete solution using a constructive method. Subsequently, the algorithm iteratively executes the following steps:

- It removes components of the complete candidate solution S, resulting in a partial solution S_p. This step called the Destruction phase.
- It rebuilds a complete candidate solution S' starting from S_p, using a constructive heuristic. It is known as the Reconstruction Phase.

- It decides whether to continue the process from S or S_p based on an acceptance criterion.

This iterative process continues until a stopping criterion is met.

The iterated greedy is a rather simple method that typically requires only short development times, especially if an already available constructive heuristic exists. Iterated greedy also provides a simple way of improving over the single application of a constructive method and generates high quality solutions for various problems. Additionally, basic versions of iterated greedy only incur few main parameters; their impact on the search process is intuitive to understand. This makes iterated greedy a desirable technique for developers of heuristic algorithms [24].

Conclusion

In summary, this chapter provided a comprehensive introduction of the fundamental concepts in the field of optimization. It delved into the definition of optimization problems and explored its key components. Moreover, it presented the diverse classifications of optimization problems based on various factors. The chapter also touched upon the classes of computational complexity, namely P and NP, and described how they relate to optimization problems. Finally, it examined an array of optimization methods and algorithms that can be employed to effectively solve optimization problems.

CHAPTER 02 MAXIMUM WEIGHTED DISJOINT DOMINATING SETS PROBLEM

Introduction

In this chapter, the fundamental definitions and notations derived from graph theory in the problem of Maximum weighted disjoint dominating sets problem (MWDDS) will be employed. Additionally, a comprehensive overview of the most critical definitions and notations in graph theory will be presented. These foundational concepts serve as a basis for the subsequent discussions and analyses conducted within this study.

1. Graph Theoretical Fundamentals

A mathematics field concerned with graphs is the Graph theory. The graphs are a mathematical construct that comprises a combination of vertices (can be referred to as nodes) and edges (can be referred to as arcs). The edges connect pairs of vertices, representing the relationship between them. Graphs can be used to model various types of dual relationships between objects. [25]

1.1. Formal Definition of a Graph

G is a graph represented as the pair $G = (V, E)$, in which:

- V is the set of Vertices, $V(G)$ denotes the set of vertices of the Graph G.
- E is the set of Edges, $E(G)$ denotes the set of vertices of the Graph G.

Each edge e that links two vertices $u\ and\ v$ can be denoted by any of the following formats:

$$e = (u, v),\ uv,\ vu \text{ Or } \{u, v\}.$$

The graph has an order (the number of vertices) $n = |V|$, and a size (the number of edges) $m = |E|$.

Remark: $\{v, u\}$ and $\{u, v\}$ refer to the same edge.

Each e has at least one and at most two vertices associated to it, these are called endpoints. An edge is said to be incident with its endpoints. Two vertices that are incident with a common edge are called adjacent vertices. In other words, they are connected by an edge. In case they are not adjacent, they are referred to as independent [26]. If an edge has only one vertex associated to it, we say that there is a loop. Also, two edges that have the exact endpoints are called multiple edges, and a graph with these last two affirmations is called a simple graph [25].

1.2.Notation and Definitions

1.2.1. Path

In a graph G, if an edge e has the vertices u and v as endpoints, the way from u to v is said to be a path. Hence, it is also a sub-graph of G.

1.2.2. Neighbors

Two distinct adjacent vertices are commonly referred to as neighbors, which means they share a common edge in the graph.

1.2.3. Open Neighborhood / Closed Neighborhood

For each vertex v, the open neighborhood of v is the set that contains any vertex u which is considered as a neighbor to v, noted $N(v) = \{u \in V | (u, v) \in E\}$. Thus, the closed neighborhood of v is the set $N[v] = N(v) \cup \{v\}$. Note that it is possible to extend the notion of neighborhood beyond individual nodes to include sets of nodes. [27]

1.2.4. Degree of a Vertex

An important number associated with each vertex v is its degree noted $deg(v)$. It is the count of how many edges in the graph have that vertex as one of their endpoints. If v does not have any loops, then its degree is equal to the number of edges in the graph that are incident with this latter. Otherwise, the loop adds 2 to the degree of the vertex instead of 1 [25].

1.2.5. Dominating Set

Let S be a subset of $V, S \subseteq V$. If any node $v \in V \backslash S$ has in minimum one adjacent (neighbor) vertex from S, then S is labeled a *dominating set* of the Graph G. [27]

A vertex $u \in S$ covers all of its neighbors, besides including itself.

1.2.6. Domatic Partition

Consider a set $D = \{S_1, S_2, \ldots, S_k\}, in$ which $S_i (i = 1, \ldots, k)$ are dominating sets in the graph G. If all these sets are disjoint two-by-two, D is named a Domatic partition. [27]

1.2.7. Domatic Number

A Domatic number is termed as $|D^*|$, which is the cardinality of the largest possible domatic partition in the graph, i.e. $D^* = argmax\{|D| \; |D \; is \; a \; domatic \; partition \; of \; G\}$, with $|D^*| \leq \delta + 1$, in which δ is the minimum degree among all $v \in V$. [27]

1.3.Common Types of Graphs

There exist several types of graphs that are identified according to certain aspects including number of edges and vertices, connectivity and their overall structure. Below is a brief explanation of a few of them.

1.3.1. Directed / Undirected Graph

In a directed graph, also known as digraph, the edges are represented by an arrow, and they are labeled arcs. Arcs are directed in such a way in which they start from certain vertex v and terminate with another one u . All the edges in the digraph must be directed.

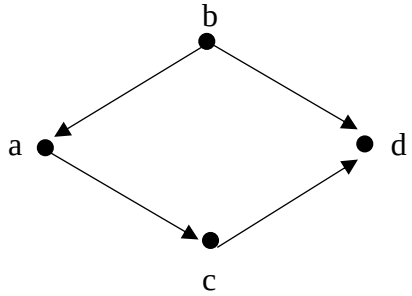

***Figure 3**: Directed Graph (Digraph)*

On the other hand, a non-directed graph is a graph that is not directed. That is, its edges have no specific direction.

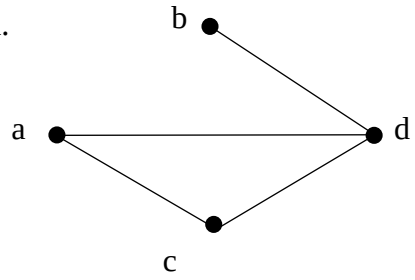

Figure 4: *Undirected Graph*

1.3.2. Connected / Disconnected Graph

The term of connectivity refers to the fact that each pair $v, u \in V$ in the graph must be connected by a path.

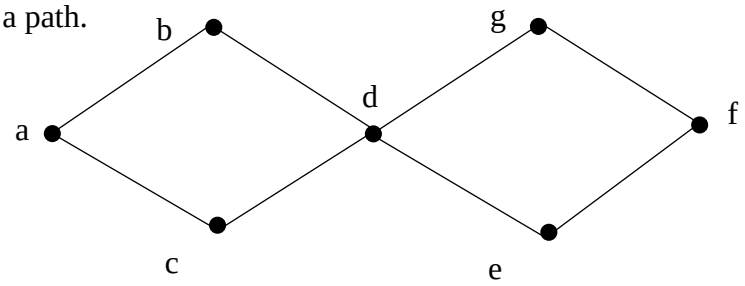

Figure 5*: Connected Graph*

Contrarily, The Disconnected graph is one in which there is at least two vertices without a connecting path between them.

1.3.3. Bipartite Graph

A bipartite graph is one that can be partitioned into two disjoint and non-overlapping sets of vertices such that there are no edges between vertices within the same set.

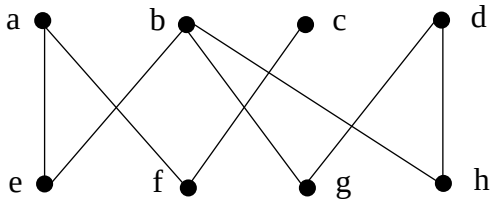

Figure 6*: Bipartite Graph*

1.3.4. Complete Graph

In this type of graph, there is exactly one edge that connect every pair of vertices.

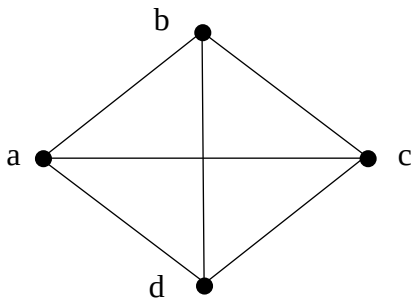

Figure 7: *Complete Graph*

1.3.5. Weighted Graph

A weighted graph is one in which the edges are labeled by a numerical value called weight. Generally, this value is positive and is given to each edge depending on the problem to be solved. It is usually used to represent real objects or processes.

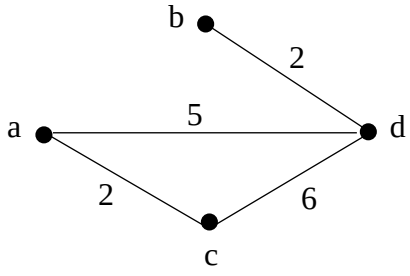

Figure 8: *Weighted Graph*

1.3.6. Simple Graph

A graph that is non-weighted, undirected, and does not have any loops or multiple edges is called a simple graph.

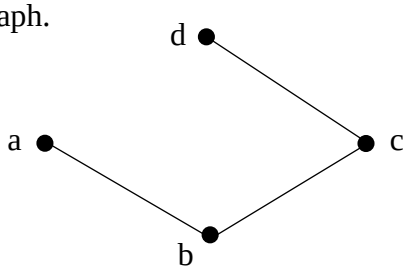

Figure 9*: Simple Graph*

2. MWDDS Problem

2.1.MDDS

Maximum disjoint dominating sets (MDDS) problem is a known problem that focuses on finding a domatic partition in a given undirected graph G. MDDS in graph theory is widely recognized as a challenging NP-hard problem. Its objective is to determine the largest possible collection of dominating sets in a given graph such that these sets are mutually exclusive [27].

2.2.MWDDS

MWDDS is an extended version of MDDS, in which each $v \in V(G)$ has a positive weight [27].

2.3.Adaptation of MWDDSP with WSNs

In the context of wireless sensor networks, the MWDDS problem can be applied to maximize the network lifetime. In a wireless sensor network, sensor nodes are responsible for sensing and transmitting data. Dominating sets are used to ensure connectivity and coverage in the network. A dominating set is a subset of nodes in the network such that every node in the network is either in the dominating set or adjacent to a node in the dominating set. [27]

By formulating the problem as MWDDS and solving it, we can select a set of disjoint dominating sets in the network. The weight associated with each dominating set can represent its significance or importance. In the context of a wireless sensor network, the weight can be related to the energy or battery capacity of the nodes in the dominating set. [27]

The objective is to select a set of disjoint dominating sets with maximum total weight. This means selecting dominating sets that cover as many nodes as possible while minimizing node overlap between the dominating sets. By doing so, we can effectively utilize the available energy resources in the network and maximize the network lifetime. [27]

Each sensor device is represented by a node, the distance between sensors by edges, and the disjoint dominating sets work by shift. The latter are responsible for forwarding and treating data which consumes more energy, while the devices out of this set are in sleep mode just to sense the data that does not consume more energy. The quality of a set is represented by the minimal lifetime value of its sensor devices, in which the first device depletes energy. As a result, the set becomes insufficient to cover all the network, so it is turned off while another set is turned on. Similarly, the latter set keeps taking the role of treating and forwarding data to the center while the other devices are in sleep mode [27].

2.4.Problem Statement

We have an undirected weighted-graph $G = (V, E, Lifetime)$. As mentioned above, V is the set of vertices and E is the set of edges. $Lifetime: V \rightarrow \mathbb{R}^+$is a function assign a positive value $Lifetime(v) \geq 0$ to each $v \in V$.

For any D of G , in which D is a domatic partition, $D = \{D_1, \ldots, D_{|D|}\}$, D is considered as a valid solution for the problem of MWDDS.

An objective function of D is defined as follows:

$$f(D) = \sum_{i=1}^{|D|} min\{Lifetime(v) \mid v \in D_i\} \qquad \textbf{(1)}$$

$$D_i \subseteq V, i = 1, \ldots, k \qquad \textbf{(2)}$$

$$N[D_i] = V, i = 1, \ldots, k \qquad \textbf{(3)}$$

$$D_i \cap D_j = \emptyset, 1 \leq i < j \leq k \qquad \textbf{(4)}$$

That is, the quality of D_i is defined as the minimal lifetime of any $v \in V(D_i)$. *Equation* (3) is to ensure that each D_i is a dominating set, and $Equation$ (4) ensures that for any $D_i, D_j \in D, D_i \cap D_j = \emptyset$.

2.4.1. Example

A graph sample of the MWDDS problem is shown in *Figure 8*. To illustrate, the graph illustrated in the figure shows seven vertices form the undirected graph G. each vertex is labeled in the format x, y, where x refers to the ID of the vertex, and y represents its

lifetime. Consider that each possible dominating set's quality is defined by the minimal weight value of its vertices. [27]

A possible solution for the graph G *is* $D = \{D_1 = \{3,4\}, D_2 = \{1,5,6\}\}$. As illustrated in *(B)*, D contains two dominating sets, $D_1 = \{3,4\}$ and $D_2 = \{1,5,6\}$, in which the lifetime of vertices 3 and 4 are 0.8 and 0.5, respectively. Hence, D_1's weight is equal to 0.5. Similarly, the quality of D_2 is equal to 0.2 given that the weight of 1, 5 and 6 are 0.9, 0.9 and 0.2, respectively.

Calculating the object function $f(D)$ of the feasible solution D is the summation of the lifetimes of D_i, where $D_i \in D$.. Consequently, the quality of D is equal to 0.7.

An optimal solution $D^* = \{D_1 = \{1,3,5\}, D_2 = \{2,4,6\}\}$, pertaining to this problem is validated as shown in *(C)*, where $f(D^*) = 0.8$ in accordance with the weights of $D_1$and D_2, that are 0.8 and 0.2, respectively. Given that any feasible solution to any MWDDS problem includes a maximum possible number of DDS $\leq \delta + 1$, where δ is the minimum degree among all $v \in V$. In this example, the minimal degree is 1; therefore, any feasible solution for this sample includes at most two DDS.

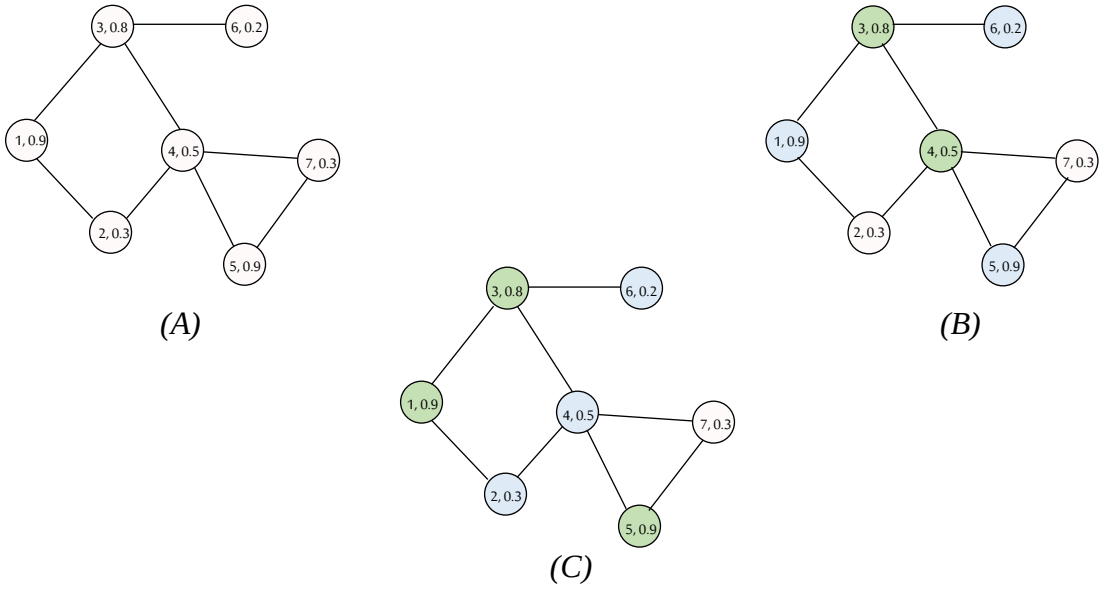

Figure 10: *A graphical sample of the MWDDS problem. (A) Problem instance ;(B) Possible solution* $D = \{D_1 = \{3,4\}, D_2 = \{1,5,6\}\}$ *;(C) Optimum solution* $D^* = \{D_1 = \{1,3,5\}, D_2 = \{2,4,6\}\}$

2.5.MWDDS Complexity

The MWDDS problem, classified within the extensive and significant domain of dominating set problems, is widely recognized as a challenging computational task with $NP-hard$ complexity when considering general graphs. It was demonstrated that the $NP-completeness$ of a specific variant of the MDDS problem is referred to as the 3-disjoint dominating sets problem, this variant involves determining whether a given graph contains three DDS. It was also established that unless $P = NP$ conjecture is true, the MWDDS problem cannot be solved in polynomial time *[16]***.**

Conclusion

To conclude, we employed the fundamental concepts and notations derived from graph theory to tackle the problem of Maximum Weighted Disjoint Dominating Sets (MWDDS) in this chapter. This provided a detailed exploration of the essential definitions and notations in graph theory. These foundational ideas form the groundwork for the subsequent discussions and analyses carried out in this study.

CHAPTER 03: POPULATION BASED ITERATED GREEDY FOR MWDDSP

Introduction

The Population-based Iterated Greedy algorithm integrates the benefits of a population-based technique with an iterated greedy approach in order to investigate the best possible solutions in the solution space. This method seeks to find disjoint dominant sets with maximal weights by keeping a diversified range of potential solutions.

1. Population Based Algorithms

Population-based algorithms are a type of metaheuristic algorithm that can be used to solve optimization problems. These algorithms work through creating a population of solutions, and then iteratively improving these solutions by using information from the other solutions in the population. This process continues until a termination criterion is met, such as a maximum number of iterations or a minimum improvement in the fitness of the solutions.

2. Population Based Iterated Greedy

The population-based iterated greedy (PBIG) algorithm is an extension of the well-known iterated greedy (IG) metaheuristic. It generates a series of solutions by repeatedly applying a constructive greedy heuristic. In each iteration, a portion of the current/incumbent solution is removed in the destruction phase. Then, the greedy heuristic is applied to the remaining partial solution to reconstruct a complete solution in the reconstruction phase.

We begin by the generation and initialization of the first population using the randomized greedy $MWDDSGreedy(D_p)$ procedure that was proposed by Bouamama in [27] with an empty solution D_p as input. When the initial population is generated, it initializes the first best-so-far solution D_{BSF}, next, a for loop starts iterating over the following phases.

First, in the destruction phase each solution of the initial population is destroyed partially resulting in a partial solution. Next, using $MWDDSGreedy$ a construction phase begins to construct a complete solution starting from the earlier partial solution. The resulting solution is added to the new population and the algorithm parameters are adapted using the procedure $AdaptParameters$. Afterwards, the best-so-far solution is picked again from the new population and is compared to the first D_{BSF} of the initial solution, and the best one is considered as the D_{BSF} for the algorithm. Finally, the next population is selected using $SelectNewPopulation$ through comparing the resulting solutions of the initial population and the new population, and taking the best solution from each one to start a new iteration. The loop keeps iterating over these phases until a given CPU time is reached, and finally it returns the D_{BSF}.

2.1. Detailed Outline of the Algorithm

An outline of the PBIG is as follows:

2.1.1 Input Parameters

The algorithm takes as input 7 parameters:

- A problem instance $G = (V, E, lifetime)$.
- p_{size} : The size of the population; in other words, it is the number of solutions to be generated in each population. It is constant in all iterations.
- $destr_{min}\ and\ destr_{max}$: Lower/ Upper bound for the solution's destruction degree.
- $det_{min}\ and\ det_{max}$: Lower/ Upper bound of the solution's construction greediness degree.
- max_{noimpr}: The maximum number of iterations without improvement of the D_{BSF} .
- r_{del}: The partial solution removal's degree (Deletion rate).

2.1.2. Pseudo Code of PBIG Algorithm

$Algorithm1$ below provides a broad overview of the PBIG algorithm for tackling the MWDDS Problem.

Algorithm1 PBIG for the MWDDSP.

Input: A problem instance: $\boldsymbol{G = (V, E, lifetime)}$ and the values of the parameters: $\boldsymbol{p_{size}}$, $\boldsymbol{destr_{min}}$, $\boldsymbol{destr_{max}}$, $\boldsymbol{det_{min}}$, $\boldsymbol{det_{max}}$, $\boldsymbol{max_{noimpr}}$, and $\boldsymbol{r_{del}}$.

Output: A family of disjoint dominating sets $\boldsymbol{\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_k\}}$

1: $\boldsymbol{\mathcal{P} := GenerateInitialPopulation()}$

2: $\boldsymbol{\mathcal{D}_{bsf} := argmmax\{f(D) \mid D \in \mathcal{P}\}}$

3: $\boldsymbol{cnt := 0}$

4: **while** termination condition not satisfied **do**

5: $\boldsymbol{\mathcal{P}new := \emptyset}$

6: **for each** candidate solution $\boldsymbol{\mathcal{D} \in \mathcal{P}}$ **do**

7: $\boldsymbol{\mathcal{D}^P := DestroyPartially(\mathcal{D})}$

8: $\boldsymbol{\widehat{\mathcal{D}} := GreedyMWDDS(\mathcal{D}^P)}$ // Algorithm 2

9: $\boldsymbol{\mathcal{P}new \leftarrow Pnew \cup \{\widehat{\mathcal{D}}\}}$

10: $\boldsymbol{AdaptParameters(\mathcal{D}, \widehat{\mathcal{D}})}$

11: **end for**

12: $\boldsymbol{\mathcal{D}ib := argmax\{f(\mathcal{D}) \mid \mathcal{D} \in Pnew\}}$

13: **if** $\boldsymbol{f(\mathcal{D}_{ib}) > f(\mathcal{D}_{bsf})}$ **then** $\boldsymbol{\mathcal{D}bsf := \mathcal{D}ib}$, $\boldsymbol{cnt := 0}$

14: **else** $\boldsymbol{cnt := cnt + 1}$ **end if**

15: $\boldsymbol{\mathcal{P} \leftarrow SelectNextPopulation(\mathcal{P}, \mathcal{P}new, cnt)}$

16: **end while**

17: **return** $\boldsymbol{\mathcal{D}bsf}$

2.1.3. Procedures

- $\boldsymbol{GenerateInitialPopulation()}$: The initial population is constructed using $MWDDSGreedy(.)$, with $D_0 = \emptyset$ as input. Note that this procedure depends on $DetD$ which is the degree of greediness of the construction, where it is the same for all the initial population solutions, and is initialized with $Det_{D_0} = det_{max}$ that is the upper bound of the construction greediness. Similarly, $destr_{D_0} = dest_{min}$, in which it is initialized by the lower bound of destruction rate.

- ***DestroyPartially*($\boldsymbol{D}$)**: This procedure takes a valid complete solution $D = \{D_1, D_2, .., D_m\}$ as input. Three main phases constitute this procedure, and they are applied on the complete Solution *D* successively:
  - Partial Solution Removal: Randomly removes max $\{1, \lfloor r_{del} . |D| \rfloor\}$ number of Disjoint Dominating sets of the complete solution D, resulting in a partial solution $D^p = \{{D_1}^p, {D_2}^p, .., {D_r}^p\}$ in which r should be less than m.
  - Worst Node Removal: This phase ensures that the quality of each Disjoint Dominating set is elevated by removing the nodes with the minimal $lifetime$ value. This originates in the fact that the set quality is determined as the minimal value of its nodes, and the solution's quality refers to the summation of all these qualities. Hence, removing the worst nodes enhances the quality of the solution.
  - Random node Removal: A group of $\lfloor destr_D \times |D_i^P| \rfloor$ nodes is chosen randomly and removed from each set iteratively, it removes one node at each iteration.
- ***GreedyMWDDS*($\boldsymbol{D^p}$)**: The process of reconstructing follows a commonly used approach known as a greedy algorithm. This algorithm gradually constructs a comprehensive solution by sequentially selecting an additional node during each construction step. However, the unique aspect of this greedy algorithm lies, first, in its ability to incorporate randomized steps, and, second, in accepting a partial solution, which may not necessarily be empty, as input. More explicitly, the Algorithm takes an incomplete solution D^P as a parameter which could potentially be $D^P = \emptyset$.

In case the solution $D^P = \emptyset$ or $i \geq |D^P|$, the ${D_i}^P$ is initialized with ${D_i}^P = \emptyset$, so the sub solution is generated as a new DDS. Otherwise, the construction of D^P that contains an already destroyed DDS is accomplished by iterating over each $D_i^P \in D^P$, and building a new solution D of Complete DDS. Note that V_{rem} refers to the set of nodes that is a part of V but not contained in any $D_i^P \in D^P$. In the first case, V_{rem} is initializd with same nodes as V, in which the solution is empty and no node belongs to it. In the second case, V_{rem} takes the role described in its definition.

Obtaining the DDS is done as follows:

First, the nodes at each iteration, are divided into 3 main classes of colors:

- ***Black Nodes:*** the nodes that belongs to at least one dominating sets, and they are called Dominators.
- ***Gray Nodes:*** All nodes that are neighbors of at least one dominator node, and they are called the dominated.
- ***White Nodes:*** Represent nodes that belong to neither of the last two groups. In other words, they neither dominate, nor are dominated by any other node.

The Greedy function that assigns the node to the ongoing DDS is signified by $Score(.)$; this function esteems any node $n \in V_{rem}$ in the following way:

$$Score(v) = lifetime(v) * whiteDegree(v) \qquad \forall v \in V_{rem} \qquad \textbf{(5)}$$

Where $WhiteDegree$ is a function that returns the number of Open neighborhoods that belongs to V_{rem}.

Next, we have the randomization used in this greedy with a restricted candidate list (RCL) where elements are added according to their quality. The parameter $DetD \in [0,1]$ specifies the size of this list. Nodes that belong to V_{rem} are evaluated and added to RCL using the $Score$ Function in the following way:

$$RCL = \{v \in Vrem | score(v) \geq score_{min} + detD(score_{max} - score_{min})\} \qquad \textbf{(6)}$$

Where $score_{min}$ and $score_{max}$ return the min/max score value of all nodes in V_{rem}.

As a last step with regards to RCL, we randomly choose one of RCL candidates to be added to the current constructing solution. Additionally, each added node must be removed from V_{rem} in parallel.

After D_i fulfills the conditions of a dominating set, the redundant nodes (two or more dominators with same Closed neighborhood) should be removed. If, by chance, a redundant node has a min lifetime in the DDS, the quality of this latter is enhanced. After this step, the DDS is added to the solution that is under construction. This process stops when $V_{rem} = \emptyset$ or each remaining node $n \in V_{rem}$ has no white node in the closed neighborhood.

Algorithm2 below shows a pseudo code of the $GreedyMWDDS$, which is provided in [27].

- ***Adapt Parameters***(): The PBIG, $DetD$ and $DestD$ parameters are given the upper bound of greediness and the lower bound of destruction sequentially. Then, they are adjusted at each PBIG iteration as follows:

$$DetD = DetD - 0.1 \qquad \textbf{(7)}$$

$$DestD = DestD + \frac{Dest_{max} - Dest_{min}}{9} \qquad \textbf{(8)}$$

When $DetD < Det_{min}$, $DetD$'s value is set to Det_{max}. Similarly, when $DestD > Dest_{max}$, $DestD$'s value is set to $Dest_{min}$.

The main goal behind these adaptations is to explore a huge solution space, in case of using a higher degree of greediness, the algorithm prioritizes selecting highly promising choices during the construction phase, which tends to yield better solutions. Additionally, a smaller solution destruction rate means that the algorithm preserves more of the previously constructed solution, preventing excessive disruptions.

However, as the algorithm progresses and reaches a point in which it can no longer find further improvements using the current approach, it adapts by moving towards a lower degree of greediness. This shift allows for exploring alternative paths and diversifying the search, potentially leading to the discovery of new and better solutions. Simultaneously, increasing the destruction rate allows for more extensive changes to the solution, thus, enabling a greater exploration of the solution space.

- ***SelectNextPopulation***$(\boldsymbol{P}, \boldsymbol{Pnew}, \boldsymbol{Cnt})$: This function is concerned with the selection or generation of the next iteration's population solutions.
    - In case $Cnt < max_{noimp}$, the next population is loaded with the best solutions from $P \cap Pnew$.
    - In case $Cnt = max_{noimp}$, only the best solution of P remains and the rest $Psize - 1$ solutions are generated using $GreedyMWDDS(D)$ with $D = \emptyset$. And $Cnt = 0$ in the end.

Algorithm2 Procedure $GreedyMWDDS(\mathcal{D}^p)$

Input: A (possibly empty) partial solution $\mathcal{D}^p = \{\mathcal{D}_1^p, \mathcal{D}_2^p, \dots, \mathcal{D}_k^p\}$
Output: A complete valid solution $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_m\}$

1: $\mathcal{D} := \emptyset$
2: **if** $\mathcal{D}^p := \emptyset$, **then** $V_{rem} := V$ **else** $V_{rem} := V \backslash \bigcup_{i=1}^{|\mathcal{D}^p|} \mathcal{D}_i^p$
3: $stopping\_condition$:= false
4: i := 0
5: **while** not $stopping\_condition$ do
6: $i := i + 1$
7: **for** each node $v \in V$ **do**
8: $color(v)$:= WHITE
9: **end for**
10: **if** ($\mathcal{D}^p = \emptyset$ ***or*** $i > |\mathcal{D}^p|$), **then** $\mathcal{D}_i := \emptyset$ else $\mathcal{D}_i := \mathcal{D}_p$
11: **while** $\mathcal{D}_i$ is not a dominating set of G (that is, $N[D\_i\,] \neq V$) **and** not a $stopping\_condition$ **do**
12: **if** $V_{rem} = \emptyset$ **then**
13: $stopping\_condition$:= true
14: **else**
15: $score_{max}$:= max$\{score(v) | v \in V_{rem}\}$
16: **if** $score_{max}$:= 0 **then**
17: $stopping\_condition$:= true {▶ No node from V_{rem} has a white closed neighbor}
18: **else**
19: $score_{min}$:= min $\{score(v) | v \in V_{rem}\}$
20: $RCL := \{v \in V_{rem} | score(v) \geq score_{min} + det_{\mathcal{D}}(score_{max} - score_{min})\}$
21: Choose v^* uniformly at random from RCL
22: $\mathcal{D}_i := \mathcal{D}_i \cup \{v^*\}$
23: $V_{rem} := V_{rem} \backslash \{v^*\}$
24: **for** each node $u \in N(v^*)$ **do**
25: **if** ($color(u) = WHITE$) **then**
26: $color(v) := GRAY$
27: **end if**
28: **end for**
29: $color(v^*) := BLACK$
30: **end if**
31: **end if**
32: **end while**
33: **if** not $stopping\_condition$ **then**
34: $Reduce(\mathcal{D}_i, V_{rem})$ {▶ Remove redundant nodes}

35: $\mathcal{D} := \mathcal{D} \cup \{\mathcal{D}_i\}$
36: **end if**
37: **end while**
38: **return** $\mathcal{D}$

In addition, we implemented:

- $\boldsymbol{DpQuality(\mathcal{D}_i)}$: The quality of a solution is assessed through the objective function, which serves as a determinant in this problem, takes a solution (set of disjoint dominating sets) as input, and returns the quality which is described before in $Equation\ (1)$.
- $\boldsymbol{BestSolutionSofar(\mathcal{P})}$: This function chooses the best solution in each generated population based on their quality. It does so by calculating the quality of the solution in the population using $DpQuality$ function. The solution with the maximum value of quality is returned as the best solution so far.

Furthermore, in the construction phase, there are several additional functions that aid in the process such as:

- $\boldsymbol{Reduce(\mathcal{D}_i)}$: This function is used in each iteration of the $GreedyMWDDS$, in which it removes the redundant nodes in the final generated dominating set, through following these two steps:
    - First, each two nodes have the same $ClosedNeighbourhood$ list; the one with the lowest $lifetime$ is removed from the set.
    - Second, if a node's $CloseNeighbourhood$ elements are already contained in the other nodes' $ClosedNeighbourhood$ lists, this node is removed.

Conclusion

To conclude, the algorithm presented in this chapter, referred to as the Population-based Iterated Greedy algorithm, merges the advantages of a population-based technique with an iterated greedy approach. Its primary aim is to explore the solution space comprehensively to uncover the most optimal solutions. By maintaining a varied array of potential solutions, this method endeavors to identify disjoint dominant sets with the highest possible weights.

CHAPTER 04: ANALYSIS OF RESULTS AND DISCUSSION

Introduction

In this chapter, we will experiment with the algorithm that we used in a standard dataset and with different parameter values, then discuss the results.

1. Experimental Evaluation

The implementation of the PBIG algorithm is performed using C# language, with a set of known libraries such as LINQ and Collections Generic. The test phase is carried out using a DELL Inspiron 3543 PC, with Windows 10 operating system, Intel ® Core™ i5-5200U @ 2.20 GHz processor, and 4GB RAM.

2. Problem Instances

The considered algorithm was applied to 48 Random graph (RG) Problem instances of Benchmark instances that are used to model the problem of Maximizing Sensor Network Lifetime. These instances are provided for the purpose of the experimental evaluation of algorithms provided by *C. Blum* [28] .

Each instance is composed of an undirected graph with $n \in \{50,100,150,200,250\}$, and different density of edges, in which each vertex is assigned a weight value between 0 and 1. The graph has a total number of n vertices and m edges, and an incidence matrix that shows the relation between the n vertices. This instances set is referred to as the randomly generated graphs (RG).

3. Algorithm Tuning

We used the same tuning values provided in [27] for the algorithm's 7 parameters:

- p_{size} : population size
- $destr_{min}$ and $destr_{max}$: Lower/Upper bound for the solution's destruction degree.
- det_{min} and det_{max}: Lower/Upper bound of the solution's construction greediness degree.
- max_{noimpr}: The maximum number of iterations without improvement of the D_{BSF} .
- r_{del}: Deletion rate.

The parameters values are chosen from the ranges shown in the Table 1 above which are provided in [27].

Parameter	Domain	Value
$\boldsymbol{p_{size}}$	{1, … ,100}	62-*20*
$\boldsymbol{detr_{min}}$	[0,1]	0.91
$\boldsymbol{detr_{max}}$	[0,1]	0.96
$\boldsymbol{destr_{min}}$	[0,1]	0.44
$\boldsymbol{destr_{max}}$	[0,1]	0.61
$\boldsymbol{max_{noimpr}}$	{1,500}	417-*244*
$\boldsymbol{r_{del}}$	[0.1,0.5]	0.11

Table 1: *Parameters, value domain, and initial values for PBIG*

4. Results and Discussion

$Table2$ shows a comparison between the results of the execution of the algorithm we implemented and the original algorithm's results provided in [27]. Consider that the PBIG Algorithm was executed to each one of the 15 problem instances with different densities d, with a computation time limited to $n/2$ CPU seconds for each instance, where n is the number of nodes in each instance. In $Table2$ The first two columns are related to each problem instance in which (n) is the number of nodes in the graph, and (d) is the average number of edges. The Time column demonstrates the CPU time for each problem instance execution which was specified previously.

The next column $Value$ shows the results obtained by executing the PBIG we implemented on the instances where $Value$ indicates the average quality of the Best Solution So far $Dbsf\ Quality$ obtained at the end of each execution of the algorithm on the corresponding instance samples, similarly, the last column indicates the same results obtained by applying the PBIG obtained in [27].

		Our PBIG		PBIG of [27]
n	**d**	**Time(s)**	**Value**	**Value**
50	15-9	25.002	2.408	2.716
	20 -9	25.120	3.444	3.960
	25-9	25.000	4.231	5.244
100	20-10	50.001	2.225	3.475
	30-10	50.012	3.812	5.402
	40-10	50.078	5.020	7.695
150	30	75.032	3.474	4.990
	40	75.001	4.004	6.0830
	50	75.100	5.071	8.618
200	40	100.156	4.005	6.486
	50	100.003	5.100	7.760
	60	100.098	5.972	9.635
250	50	125.009	4.980	7.536
	60	125.012	6.229	9.021
	70	125.000	8.513	10.799

Table 2: *Comparison of numerical results of applying the PBIG Algorithm we implemented and the original Algorithm on a set of benchmark instances*

The results of our implementation are relatively close to the values of the original one with the instance of $n = 50$, while for the rest instances shows lesser close results.

The difference between the results is due to the difference in the execution environment, the implementation and the used language in which in [27] the algorithm was implemented using $C++$ and in our case it was implemented using $C\#$.

Although that, the results can be considered effective and gives reasonable results and good performance in line with the given CPU time.

To show the solution details, we chose a sample of the problem instance in which ($n = 50, d = 25$). The execution of the algorithm on this instance gave us the following solution:

Disjoint set ID	Color	Nodes number	Lifetime	Nodes id
0	Dark Blue	4	0.241	4, 3, 22, 25
1	Green	5	0.051	10, 21, 28, 33, 37
2	Purple	4	0.719	9, 11, 15, 50
3	Black	4	0.186	7, 17, 36, 49
4	Light Blue	5	0.106	8, 16, 19, 24, 48
5	Red	4	0.168	5, 20, 41, 43
6	Light Green	4	0.639	1, 2, 38, 39
7	Light Pink	4	0.235	23, 18, 26, 45
Unused	Gray	16		6, 12, 13, 14, 27, 29, 30, 31, 32, 34, 35, 40, 42, 44, 46, 47

Table 3: *Detailed numerical description of the solution obtained by applying the PBIG algorithm to for a graph instance of n=50, d=25.*

The solution is a set of dominating sets that are totally disjoint, the result we obtained is a list of 8 $DDSs$. The first column shows the id of the dominating set in the solution, and the second one is a specific color given to each DDS which will be helpful later to illustrate the solution as a graph. Also, we have the number of nodes in each dominating set, in other word, the nodes that are considered as dominators at each DDS. Additionally, we have the lifetime of each dominating set, which was as described earlier, the quality of the set equivalent to the min lifetime of its nodes. The last column shows the ids of dominators in each set.

Relying on the already specified colors in *Table 2*, *Figure 10* represents the illustrated solution for the sample instance of $(n = 50, d = 25)$, where each color indicates a specific dominating set. The unused nodes are colored in $Gray$, and they do not belong to any of the dominating sets.

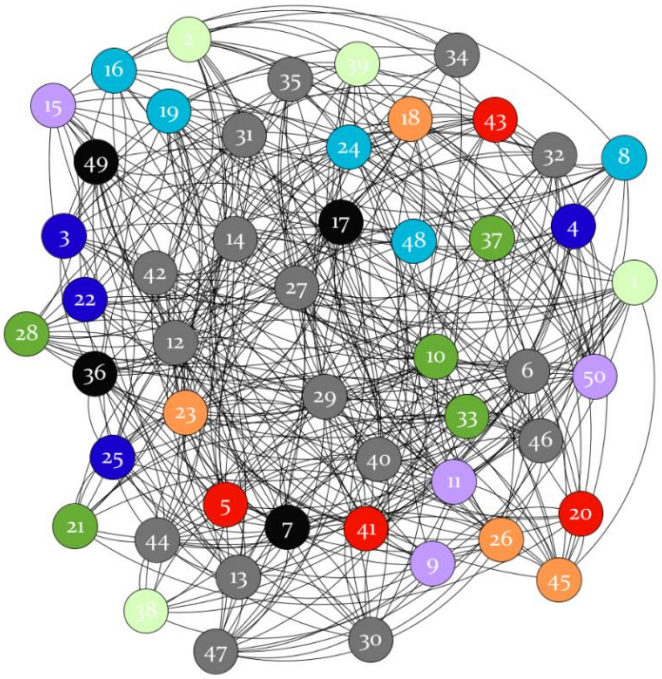

Figure 11: *Graphical representation of the application of PBIG on a graph instance of n=50, d=25*

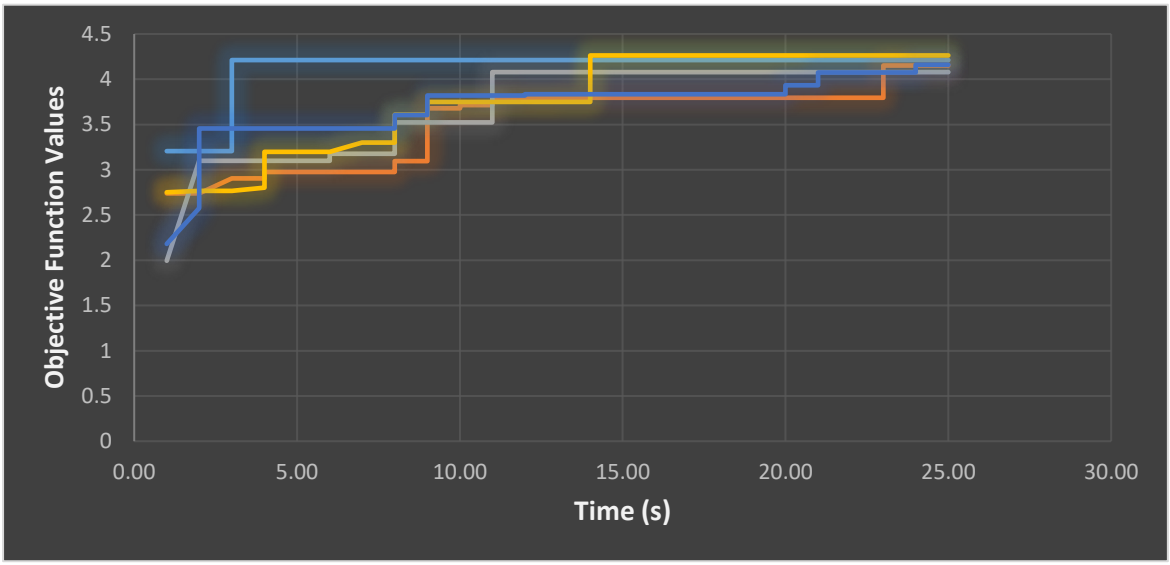

Figure 12: *Evolution of the objective function which indicates the quality of the obtained solutions generated by PBIG over time for of 5 independent PBIG runs for an instance of n=50, d=25.*

Figure 11 shows the growth of the quality of the solutions generated by independent applications of PBIG over time. With the same instance of $(n = 50, d = 25)$. Keep in mind that the computational time was limited to $n/2$ CPU seconds. The figure shows the

growth of 5 independent runs per problem instance. The obtained results assert that the given limit to computational time is commensurate for the objective function to reach convergence before it reaches the end of the limited time. The different growth of each run indicts the use of the randomized constructive greedy method $GreedyMWDDS(.)$ that builds solutions randomly in the construction phase and in similar way it reaches the same convergence value in each one of the independent applications.

Conclusion

The results obtained through applying the PBIG algorithm on the problem instances showed satisfactory outcomes. It was clearly shown that the objective function is developing over time until it reached the convergence in the specified time. And as proven in [16], the problem has an $NP - hard$ complexity assuming $P \neq NP$. Therefore, this problem cannot be solved in polynomial time P, but it can be verified in non-deterministic polynomial time NP.

GENERAL CONCLUSION

In our study we attempted to implement a population-based iterated greedy to solve the problem of MWDDS, in order to prolong the lifetime of wireless sensor network. The presented MWDDS problem aimed at finding the highest possible number of separated dominating sets in a graph.

By applying this algorithm to the problem instances, positive outcomes have been achieved. Through its application, this algorithm worked iteratively to generate each time, enhanced quality of the best solution so far in the population, beginning with destroying solutions slightly. Subsequently, by means of a random greedy heuristic method, a new solutions population is rebuilt to get an evaluation in the quality values by removing nodes that lower the solution's quality. The obtained results show the objective function perform an evident value development the overtime up to the convergence of the function.

Regarding future work, we would incorporate a local search algorithm that improves the quality of the generated solution. In addition, we would like to make some changes concerning the process of eliminating nodes to reduce the size of the obtained in each iteration of the PBIG. Furthermore, we are interested in conducting a test of this algorithm on alternative instances, particularly larger graphs, as well as applying them to diverse problem domains.

BIBLIOGRAPHY

[1] H. Yetgin, K. Cheung, M. El-Hajjar and L. Hanzo, "A survey of network lifetime maximization techniques in wireless sensor," *IEEE Commun*, vol. 19, p. 29, 2017.

[2] M. Cardei, M. Thai, Y. Li and W. Wu, "Energy-efficient target coverage in wireless sensor networks," *In Proceedings of the IEEE 24th*, vol. 3, p. 9, 13-17 march 2005.

[3] G. Cornuejols, Optimization Methods in Finance, Pittsburgh: Cambridge University Press, 2006.

[4] K. I. SOUADIA and . B. DJABOU, "(ACO+GPE) for solving the traveling salesman Problem (TSP)," University of Mohamed El Bachir El Ibrahimi, Bordj Bou Arreridj , 2021.

[5] A. Astolf, "Optimization An Introduction,"Wiley, United States, 2006.

[6] S. J. W. Jorge Nocedal, "Numerical Optimization," Springer Science & Business Media, UnitedStates of America, 2006.

[7] S. Dragan, "Single-objective vs. Multiobjective Optimisation," International Congress on Environmental Modelling and Software, LUGANO, SWITZERLAND, 2002.

[8] S. Lounas, "A Hybrid Metaheuristic for the Minimum," Mouhamed Boudiaf University, Msila,Algeria, 2017.

[9] "neos-guide.org," Neos Guide , 2022. [Online]. Available: https://neos-guide.org/guide/types/. [Accessed 30 05 2023].

[10] J. Arora, Introduction to Optimum Design 3rd Edition, Cambridge, Massachusetts: Academic Press, 2012.

[11] F. S. Hillier and G. J. Lieberman, Introduction To Operations Research Tenth Edition, New York, United States: McGraw Hill, 1969.

[12] S. Boyd and L. Vandenberghe, Convex Optimization, Cambridge, United Kingdom: Cambridge University Press, 2004.

[13] A. P. Engelbrecht, Computational Intelligence: An Introduction, New Jersey, United States: Wiley, 2007.

[14] M. Francisco and S. Revollar, "A comparative study of deterministic and stochastic optimization methods for Integrated Design of processes," *IFAC-PapersOnLine* , p. 6, 2005.

[15] T. El-Ghazali, Metaheuristics: From Design to Implementation, New Jersey, United States: John Wiley & Sons, 2009.

[16] S.Bouamama, "Design of a Learning Method for Automatic Data Extraction," Ferhat Abbas University, Setif, Algeria, 2013.

[17] M. R. Garey and D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, New York, NYUnited States: W. H. Freeman & Co.Subs, 1990.

[18] B. Esfahbod, "Wikimedia,"1 November 2007. [Online]. Available: https://en.wikipedia.org/wiki/P_versus_NP_problem. [Accessed 15 April 2023].

[19] V. V. S. Muthuraman, "A Comprehensive Study on Hybrid Meta-Heuristic Approaches Used for Solving Combinatorial Optimization Problems," *2017 World Congress on Computing and Communication Technologies (WCCCT)*, pp. 185-190, 2017.

[20] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, p. 35, 200.

[21] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Thomas H. Cormen, Introduction to Algorithms, Fourth Edition, United States: The MIT Press, 2022.

[22] V. V. Vazirani, Approximation Algorithms, Berlin, Germany: Springer Science+Business Media, 2003.

[23] H. H. Hoos and T. Stützle, Stochastic Local Search Foundations and Applications, San Francisco, California: elsevier inc, 2005.

[24] P. P. G. Rafael Martí and C.Resende, Handbook of Heuristics, Gewerbestrasse, Switzerland: Springer, 2018.

[25] Guichard.D, An introduction to combinatorics and graph theory, California,USA: eBook (Creative Commons Licensed), 2017.

[26] T. W. Haynes, S. T. Hedetniemi and M. A. Henning, Topics in domination in graphs, New York: Springer, 2020.

[27] S. Bouamama, C. Blum and P. D. Pinacho, "A Population-Based Iterated Greedy Algorithm for Maximizing Wireless Sensor Network Lifetime," *Sensors*, vol. 22, no. 1804, p. 20, 2022.

[28] C. Blum, "Research Interests," Artificial intelligence research institute, [Online]. Available:https://www.iiia.csic.es/~christian.blum/research.html#Instances. [Accessed 10 5 2023].

Abstract:

This study uses a recently proposed enhanced new algorithm that has the subject of recent interest in research aiming to improve the battery life of wireless sensor network batteries. It is the Population-Based Iterated Greedy (PBIG) algorithm, which is a metaheuristic algorithm. It is essentially an algorithm that uses the concept of Maximum Weighted Disjoint Dominating Sets (MWDDS). Our work sought a total comprehension of the algorithm to write the best simulation program for the way it works. To achieve this goal, we used the experimental instances provided by the scientific community. The program's efficiency was proved, and high-quality results with regards to performance and solution rationality were obtained. A detailed explanation of the program's work is provided in the dissertation, including a delineation of the steps of its functioning using tables, figures and graphs.

Keywords: Population Based Iterated Greedy, Maximum weighted disjoint dominating sets, Wireless sensor network, Optimization Algorithms.

ملخص:

تتمحور هذه الدراسة حول خوارزمي محسن مقترح حديثا ضمن الأبحاث التي تهتم بإشكالية تحسين استهلاك بطاريات شبكات الاستشعار اللاسلكية، الخوارزمي يتفرع عن فئة "الميتاهرستك" وهو الخوارزمي الجشع التكراري القائم على مجتمع الحلول الجيدة (PBIG) وتحديدا الذي يعتمد على مفهوم المجموعات المسيطرة غير المتداخلة ذات الأوزان الأعلى (MWDDS). ارتكز عملنا على فهم الخوارزمي بشكل تام من أجل كتابة أحسن برنامج قد يحاكي طريقة عمله ولجأنا لهذا الغرض إلى مجتمعات تجريبية متاحة من طرف المجتمع العلمي وقد أظهر البرنامج كفاءته وقدم نتائج عالية الجودة من حيث الأداء ومنطقية الحلول، قمنا في المذكرة بتقديم شرح مفصل لطريقة عمل البرنامج تضمنت توضيحا لمراحل عمله مع الاعتماد على جداول ومخططات توضيحية.

الكلمات المفتاحية: الجشع التكراري القائم على مجتمع الحلول الجيدة، المجموعات المسيطرة غير المتداخلة ذات الأوزان الأعلى، شبكات الاستشعار اللاسلكية، خوارزميات التحسين.