PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
Ministry of Higher Education and Scientific Research
Mohamed Khider University – BISKRA
Faculty of Natural, Natural and Life Sciences
Computer science department

Order no:3/RTIC/M2/2023

Memory

Submitted for the Academic Master's degree in

Computer science

Journey: **Networks and Information and Communication Technologies (RTIC)**

Metaheuristics for solving the routing problem in Ad-Hoc Mobile Networks

By:

MAHMOUD REBHI

Supported on 19/06/2023 before the jury composed of:

Bahi Naima	President	M.C.B
Berghida Meryem	Supervisor	M.C.B
Belounar Saliha	Examiner	M.A.A

Année universitaire 2022-2023

Table des matières

General introduction

Optimization algorithms are an essential tool for solving complex problems in various domains, including ad hoc networks, pathfinding, and metaheuristics[29]. These algorithms aim to find the best solution to a given problem by exploring the search space and evaluating the fitness of candidate solutions. One of the most challenging problems in optimization is finding the shortest path in a graph.

This document provides a comprehensive overview of optimization algorithms and their applications in solving the shortest path problem in ad hoc networks. The first chapter introduces mobile ad hoc networks (MANETs) and their routing protocols, highlighting the importance of efficient routing in these networks[17]. The second chapter discusses complexity theory and optimization problems, including discrete and continuous optimization problems and introduces heuristics and metaheuristics as general approximation algorithms for solving difficult problems[29] [10] [1].

The third chapter focuses on the Biogeography-Based Optimization (BBO) algorithm[19], a metaheuristic approach that can be used to solve hard combinatorial problems. The chapter provides an introduction to the BBO algorithm and its application in solving the shortest path problem in ad hoc networks. The chapter also compares the BBO algorithm with Ant Colony Optimization (ACO) and discusses their similarities and differences.

The fourth chapter presents a Java application that uses the BBO algorithm to solve the shortest path problem in a given graph. The application includes a user-friendly GUI interface and allows users to input parameters, visualize progress, display the best path, and control the execution. The chapter also compares the performance of the BBO algorithm with that of ACO and shows that the BBO algorithm outperforms ACO in finding the shortest path with lower fitness values.

Finally, the document provides a list of academic papers and books related to optimization algorithms and their applications, including genetic algorithms, dynamic programming, and branch-and-bound algorithms. These sources cover a range of case studies and methods, providing a useful resource for researchers and practitioners in the field of optimization algorithms.

Overall, this document provides a comprehensive overview of optimization algorithms and their applications in solving the shortest path problem in ad hoc networks. The BBO algorithm is shown to be an effective approach for finding the shortest path in a graph, and the Java application provides a useful tool for researchers and practitioners in this field.

Chapitre 1

The Manets networks

1.1 Introduction

1.1.1 General introduction

The scientific and industrial community's interest in communications has recently shifted to more difficult scenarios where a variety of mobile units equipped with wireless transmitters and receivers continue without any fixed infrastructure[17].

1.1.2 Definition of Ad-Hoc network

Ad-Hoc Network is an independent group of mobile phone users that connect to one another with a radio link in a dynamic environment

Network topology changes rapidly and unexpectedly over time. So the connection between the contract can vary in time because of the departure of the node, and the arrival of the new node, and the possibility of having a mobile node to maintain the connection between the nodes in the network.

Each node operates in an Ad-hoc wireless network as a transmitter, host, router. Other functions are also distributed to the contract such as Management, Control, Topology Discovery, Information Transfer and Efficient Use of Battery Power.

Whatever the application, the MANET needs efficient distributed algorithms to determine the organization of the network and the timing of linking, power monitoring and routing.

Due to limited range of transmission, mobile hosts can communicate directly with each other if sufficiently close (peer-to-peer) or (multi-hopper) in a non-direct manner by having other intermediate mobile phone hosts move their packages[17].

1.2 MANET Features

1.2.1 Mobile

— The stations are not fixed in the MANETs networks ;
— They can move and are completely independent ;
— New stations are allowed to enter or leave the network at any time ;
— Modifying the topology of a MANET network over time is a key factor[15].

1.2.2 Wireless

— Stations in a MANET network use wireless support to communicate with each other[18].

1.2.3 Without infrastructure

— MANETs are not dependent on a fixed architecture, so they can be easily deployed[18].

1.2.4 Self-organized and distributed

— MANETs networks do not have a central point to coordinate or centralize exchanges ;
— Nodes must self-organize for proper network operation[11].

1.2.5 Multi-hop

— Since the range of stations is limited, it may be necessary for stations to act as an intermediate bridge to transmit a packet from a source to a destination. therefore, the nodes of a MANET network act as a router and relay the packets they receive to participate in multi-jump routing[18].

1.2.6 limited resources

— Limited resources affect the entire communication chain of a MANET network, from nodes to communication links ;
— As the terminals are mobile, they are mainly battery operated[6] ;
— Wireless link capacity is also limited compared to wired networks ;
— The error rate is much higher than in a wired network[6].

1.2.7 Security challenges

— MANETs are vulnerable to various security threats, such as eavesdropping, interception, and denial of service attacks. The lack of a fixed infrastructure and the decentralized nature of the network makes it challenging to implement effective security measures.

1.3 Classification of ad hoc networks

1.3.1 Classification According to the communication

Depending on the configuration, communication in an ad hoc network can be either single hop or multihop[18].

1.3.1.1 Single-Hop Ad Hoc Network

— Nodes are in their reachable area and can communicate directly ;
— Single-hop ad hoc networks are the simplest type of ad hoc networks.

1.3.1.2 Multihop Ad Hoc Network

— This category differs from the first class in that some nodes are remote and cannot communicate directly with them. So there should be another intermediate node ;
— High performance routing protocols should be adaptable to rapid topology modification[18].

1.3.2 Classification According to the Node Configuration

A further classification of ad hoc networks can be performed on the basis of the hardware configuration of the nodes. There are two types of node configurations : homogeneous networks and heterogeneous networks[18].

1.3.2.1 Homogeneous Ad Hoc Networks

— In homogeneous ad hoc networks, all nodes possess the same characteristics regarding the hardware configuration as processor, memory, display, and peripheral devices. Most wellknown representatives of homogeneous ad hoc networks are wireless sensor networks[18].

1.3.3 Classification According to the Coverage area

— Ad hoc networks can be categorized, depending on their coverage area, into several classes : body area network (BAN), personal area network (PAN), local area network (LAN), metropolitan area network (MAN), and wide area network (WAN)[18].

1.4 Routing protocols in MANET

Several ad hoc protocols have been designed for accurate, fast, reliable routing for a high volume of changeable network topology.

These protocols deal with network topology changes such as high power consumption, low bandwidth and high error rates.

These routing protocols may generally be categorized into three main types : proactive or table-driven, reactive or on-demand-driven, and hybrid. These protocols are classified according to : their techniques, their hop count, link state, and source routing in a route-discovery mechanism[18].

1.4.1 Reactive Routing Protocols

1. This Protocol creates a route to destination only upon request. So whenever a sender node needs to transfer data packets to a receiver node, the sender node commences the route search process in this protocol. As a result, the demand for a route initiates the route search process, hence the name "reactive protocol" ;
2. The network layer (Layer 3 of the OSI reference model) of mobile nodes implements reactive protocols ;
3. The mechanisms utilized for routing are the Route Discovery and Route Maintenance functions.

1.4.1.1 An example of a protocol in this category

1.4.1.2 Ad hoc on-demand distance vector (AODV)

— The AODV protocol is a protocol based on the construction of routing tables. Indeed, each node has its own routing table containing for each destination the next node to contact. A route is discovered by flooding by the issuer of a Route Request (RREQ) package ;
— Upon receipt of one of these packets, if the node knows the path to the source, it sends a RREP (Route Reply) response to the transmitter that stops flooding the network.if the node does not know the path, it transmits the packet to its neighbors while memorizing the previous node that made the request ;
— If the link breaks, a RERR (Route Error) message is sent to the sender who decides whether or not to repeat the sending of the packet according to the rate of use of the route[22].

1.4.2 Proactive Routing Protocols

1. Proactive routing protocols enable each node to keep up-to-date routing information in a routing table ;
2. This routing table is periodically exchanged with all other nodes as well as when network topology changes ;
3. When the node needs to send a message it simply looks on the path to the destination in the routing table ;
4. The biggest challenge faced by this type of protocol is updating routing tables[18].

1.4.2.1 An example of a protocol in this category

1.4.2.2 Destination Sequence Distance Vector (DSDV)

— It is a vector routing system that needs each node to communicate routing changes on a frequent basis ;
— DSDV uses the table-driven routing mechanism ;
— A routing table is stored on each network node and specifies all of the network's destinations as well as the number of hops required to reach them ;
— Each item has a sequence number that can be used to identify stale entries ;
— This approach avoids routing loops in the protocol from arising[22].

1.4.3 Hybrid Routing Protocols

1. Hybrid protocols are created by combining proactive and reactive protocols. Their design combines the benefits of both proactive and reactive techniques to obtain superior results ;
2. To begin, proactive routing is employed to collect any previously unknown routing data ;
3. Reactive routing approaches are used to keep the routing information updated when the topology changes[18].

1.4.3.1 An example of a protocol in this category

1.4.3.2 Zone routing protocol (ZRP)

— Is a hybrid routing protocol that merges the proactive intrazone routing mechanism of (IARP) with the reactive interzone routing mechanism of IERP ;
— If the destination of a packet is in the same zone as the origin, the proactive protocol is used to transport the packet immediately using a previously saved routing table ;

— A reactive protocol takes over if the route extends beyond the packet's originating zone, examining each succeeding zone in the route to see if the destination is within that zone ;
— This reduces the time it takes to process specific routes ;
— After a zone is validated to include the target node, the stored routing table listing is utilized to broadcast the packet[16].

1.5 importance of Routing Protocols in MAnet

The routing protocols for ad hoc wireless networks should be capable of handling multiple hosts with limited resources, such as bandwidth and energy. The main challenge for routing protocols is that they must also deal with host mobility.

There are two main algorithms used and based on path selection for most of the existing MANET routing protocols : the **Bellman–Ford algorithm** and **Dijkstra's algorithm**.

1.5.0.1 Importance of the routing approach to solve problems.

— some routing protocols use *proactive (table-driven)* path discovery to reduce the route discovery delay (time) ;
— Other routing protocols use *reactive (on-demand)* path discovery to reduce and control overhead ;
— Some other approaches merge proactive and reactive path discoveries to reduce delay and control overhead. This type of protocol is called a *hybrid routing protocol*.

1.5.0.2 Routing protocols should have the following features :

— Providing quick and high efficiency, for example, bandwidth, memory, and battery, in adapting to MANET topology change, especially in a high mobility environment ;
— Providing an alternative path in case the primary path fails ; this will save time and power in an ever-hanging MANET network topology ;
— Finding the optimum path instead of the shortest path when applications require QoS, for example, bandwidth, end-to-end delay, and packet losses, to deliver data from the source to the destination ;
— Providing quick establishment of paths, so that they can be used before an existing path becomes invalid ;
— Consideration of QoS parameters, such as data rate, delay, and node battery life, when locating a path between the source and destination[18].

1.6 Conclusion

The chapter has presented the overview of wireless networks and different aspects of MANET, such as, definition, routing protocols, classification, special features and importance of routing protocols in MANET. The MANET characteristic features are also pointed out such as Wirele, Without infrastructure, limited resources and security. This chapter also briefly covered the classification of MANETs in terms of communication procedure (single hop/multi hop), coverage area[18].

Chapitre 2

Metaheuristics

2.1 Introduction

There are many real problems of finding the perfect solution from a limited range of options. These tasks are often considered extremely difficult. So we have seen that approximation algo- rithms are the main alternative for solving this category of problems.

— Approximate algorithms can decompose into two categories : specific heuristics and meta-heuristics. Specific heuristics are problem dependent ; they are designed and applicable to a particular problem ;
— Meta-heuristics represent the most general approximation algorithms. They are applicable to a wide variety of optimization problems, i.e. they can be designed to optimize any problem. meta-heuristics solve problem- cases that they think are difficult, generally by studying the search space efficiently ;
— The meta-heuristics serves three main themes : solving problems faster, solving difficult problems, and getting powerful algorithms, moreover they are easy to design, execute and very flexible ;
— Meta-heuristics is a branch of optimization in computer science and ap- plied mathematics associated with algorithms and computational complexity theory.[29]

2.2 Complexity theory

This section addresses some of the findings regarding the intractability of problems, our focus is on the complexity of resolvable problems ; It means she has an algorithm to solve[29].

2.2.1 Algorithmic Complexity

Algorithmic complexity is a measure of how long an algorithm would take to complete given an input of size n. sometimes complexity is also analyzed in terms of space, which translates to the algorithm's memory requirements[4]."

Complexity classes	Type of complexity	Definition	Example
O(1)	Constant time	An algorithm is said to run in constant time if it requires the same amount of time regardless of the input size[25].	array : accessing any element
O(n)	Linear time	An algorithm is said to run in linear time if its time execution is directly proportional to the input size, i.e. time grows linearly as input size increases[25].	array : linear search, traversing, find minimum
O(log n)	Logarithmic time	An algorithm is said to run in logarithmic time if its time execution is proportional to the logarithm of the input size[25].	binary search
$O(n^2)$	Quadratic time	An algorithm is said to run in logarithmic time if its time execution is proportional to the square of the input size[25].	bubble sort, selection sort, insertion sort

TABLE 2.1 – Complexity classes with their definitions and examples.

2.2.2 Complexity of problem

In computer science : there are some problems whose solutions have not yet been found, these problems are classified into categories called : complexity class.

These categories help scientists classify problems based on the amount of time and distance in memory necessary to solve and verify problems[2].

— The time complexity of an algorithm describe how long it takes to verify the answer[2].

— The space complexity of an algorithm describes how much memory is required for the algorithm to operate[2].

TABLE 2.2 – Types of Complexity Classes | P, NP, CoNP, NP hard and NP complete[2].

Complexity class	Characteristic feature	Example
P	Easily solvable in polynomial time.	The Greatest common factor (GCD) of 3 and 5 is 1
NP	Yes, answers can be checked in polynomial time.	Hamiltonian path problem : Asks whether there is a route in a directed graph from a beginning node to an ending node, visiting each node exactly once.
Co-NP	No, answers can be checked in polynomial time.	Check prime number.
NP-hard	All NP-hard problems are not in NP and it takes a long time to check them.	Halting problem : The problem of determining, from a description of an arbitrary computer program and an input, whether the program will finish running, or continue to run forever.
NP-complete	NP-complete problems are the hard problems in NP.	Traveling salesman problem.

2.3 Optimization Problem

Human beings face daily difficulties - challenges or contradictory situations to be solved to achieve the desired goal, these situations can in practice be called problems. Theoretically, the problem can be summarized as a dif- ficulty that raises an analytical position on a topic to gain knowledge and overcome this problem.

From a computer science perspective, the problem is a task that the algorithm can solve or answer from a practical perspective, computational problems can be broadly classified into six main categories of problems : ordering, counting, searching, decision, function, and optimization. Con- cerning ordering, counting, and searching, Their own definitions describe their objectives as problems. The decision problem is a category of problems that is decided after the analysis of the situation. The return of this type of problem is the answer by "Yes", "No" or 1, 0, Whereas a function problem is an extension of the decision, any data type can be returned, The problem of optimization (OP) is a category of problems solved by methods of optimization aimed at finding an optimal solution in a wide range of pos- sible solutions. Consequently, the output of an optimization problem is the solution itself[23].

Optimization problems are divided into two categories of problems :

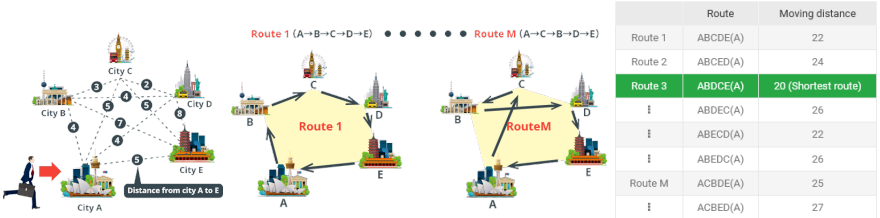
1. Continuous optimization problems.
2. Discrete optimization problems.

2.3.1 Discrete optimization problem

A discrete optimization problem is the act of trying to find out the value (combination) of variables that optimizes an index (value) from among many options under various constraints.

2.3.1.1 Example : Traveling salesman problem

As an example of the traveling salesman problem let's think about the prob- lem of a delivery route problem. This issue is a question of how the seller should navigate for the shortest route (optimal route) when visiting multiple cities when given a list of in- tercity distances. A major caveat is that he will visit each city only once. If the number of cities is small, it is possible to investigate all the solution can- didates. However, as the number of cities increases to 30 cities, for example, meaning the number of candidates has increased, meaning the combination becomes the phenomenal number of 10 to the 30th power. In this case, to comprehensively list all solution candidates and to find the optimum route, even if using a supercomputer, will take the astronomical time of 14 million years ! The obvious feature of the combinatorial optimization problem is that it becomes very difficult to get an accurate solution as the number of parameters increases[3].



	Route	Moving distance
Route 1	ABCDE(A)	22
Route 2	ABCED(A)	24
Route 3	ABDCE(A)	20 (Shortest route)
⋮	ABDEC(A)	26
⋮	ABECD(A)	22
⋮	ABEDC(A)	26
Route M	ACBDE(A)	25
⋮	ACBED(A)	27

FIGURE 2.1 – TSP problem[3]

2.3.2 Continuous optimization problem

The metaheuristics are very often employed for Discrete problems, but there is a class of problems often encountered in engineering, where the objective function is continuous and for which the metaheuristics can be of great help[10].

— **For example in engineering** If we make the side lengths of the garden bigger, the garden space becomes bigger.

what if we have some restriction on how much fencing we can use for the perimeter ?

2.3.2.1 Example : Maximizing the Area of a Garden

A rectangular garden is to be constructed using a rock wall as one side of the garden and wire fencing for the other three sides (**Figure 1.2**). Given 100ft of wire fencing, determine the dimensions that would create a garden of maximum area. What is the maximum area ?

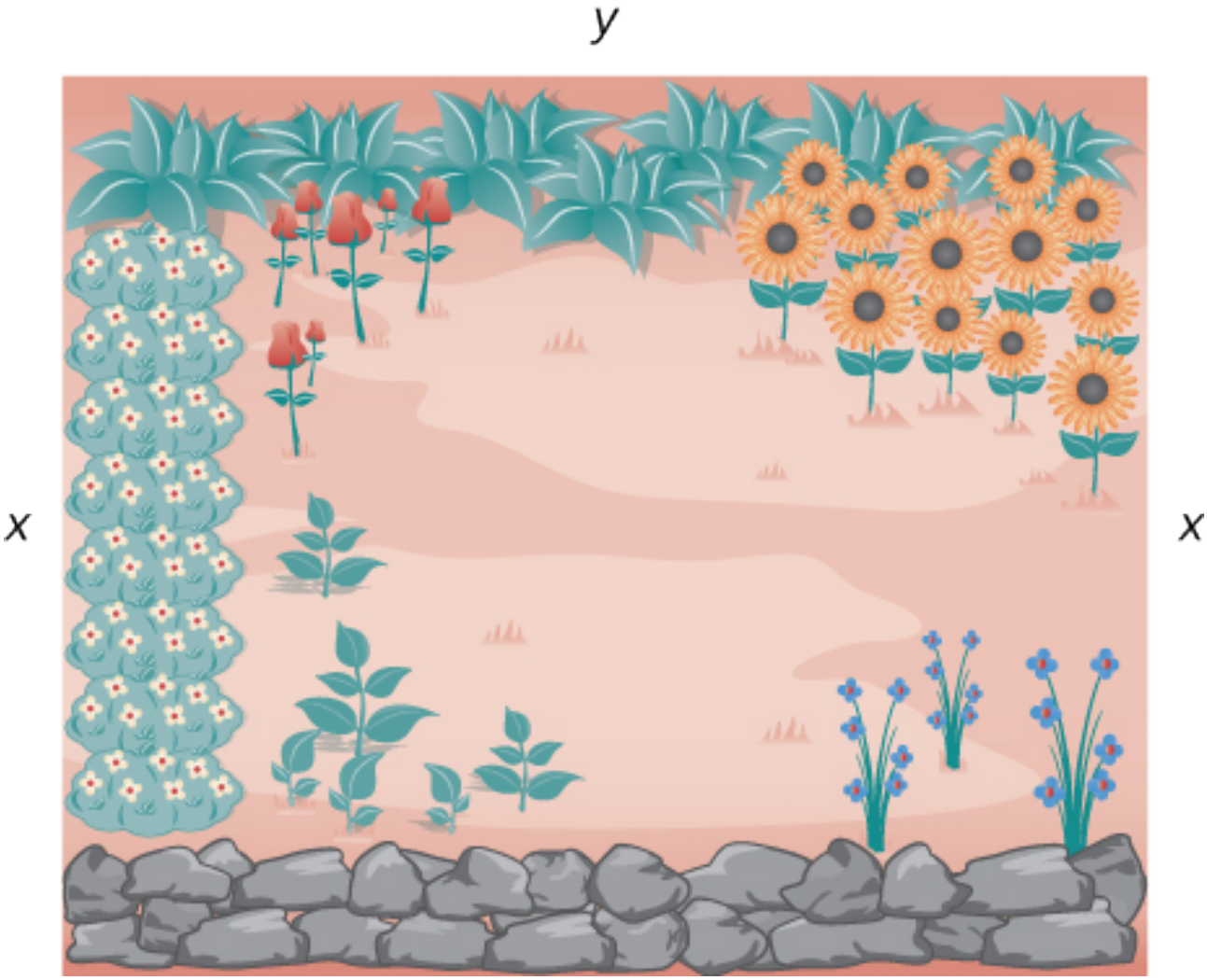


FIGURE 2.2 – We want to determine the measurements x and y that will create a garden with a maximum area using 100ft of fencing[14].

Solution

— Let x denote the length of the side of the garden perpendicular to the rock wall and y denote the length of the side parallel to the rock wall.

— Then the area of the garden is

$$A = x * y.$$

— We want to find the maximum possible area subject to the constraint that the total fencing is 100ft.

— From Figure, the total amount of fencing used will be 2x+y. Therefore, the constraint equation is

$$2x + y = 100.$$

— Solving this equation for y, we have $y = 100 - 2x$ Thus, we can write the area as

$$A(x) = x * (100 - 2x) = 100x - 2x^2.$$

— Before trying to maximize the area function $A(x) = 100x - 2x^2$, we need to determine the domain under consideration. To construct a rectangular garden, we certainly need the lengths of both sides to be positive. Therefore, we need x>0 and y>0. Since $y = 100 - 2x$, if y>0, then x<50. Therefore, we are trying

to determine the maximum value of A(x) for x over the open interval (0,50). Therefore, let's consider the function $A(x) = 100x - 2x^2$ over the closed interval [0,50]. If the maximum value occurs at an interior point , then we have found the value x in the open interval (0,50) that maximizes the area of the garden.

Therefore, we consider the following problem :

$$\textbf{Maximize } A(x) = 100x - 2x^2 \; Overtheinterval \; [0, 50].$$

— As mentioned earlier, since A is a continuous function on a closed, bounded interval, by the extreme value theorem, it has a maximum and a minimum. These extreme values occur either at endpoints or critical points. At the endpoints, A(x)=0. Since the area is positive for all x in the open interval (0,50), the maximum must occur at a critical point . Differentiating the function A(x), we obtain :

$$A'(x) = 100 - 4x.$$

— Therefore, the only critical pointis x=25 (Figure 1.3). We conclude that the maximum area must occur when x=25.

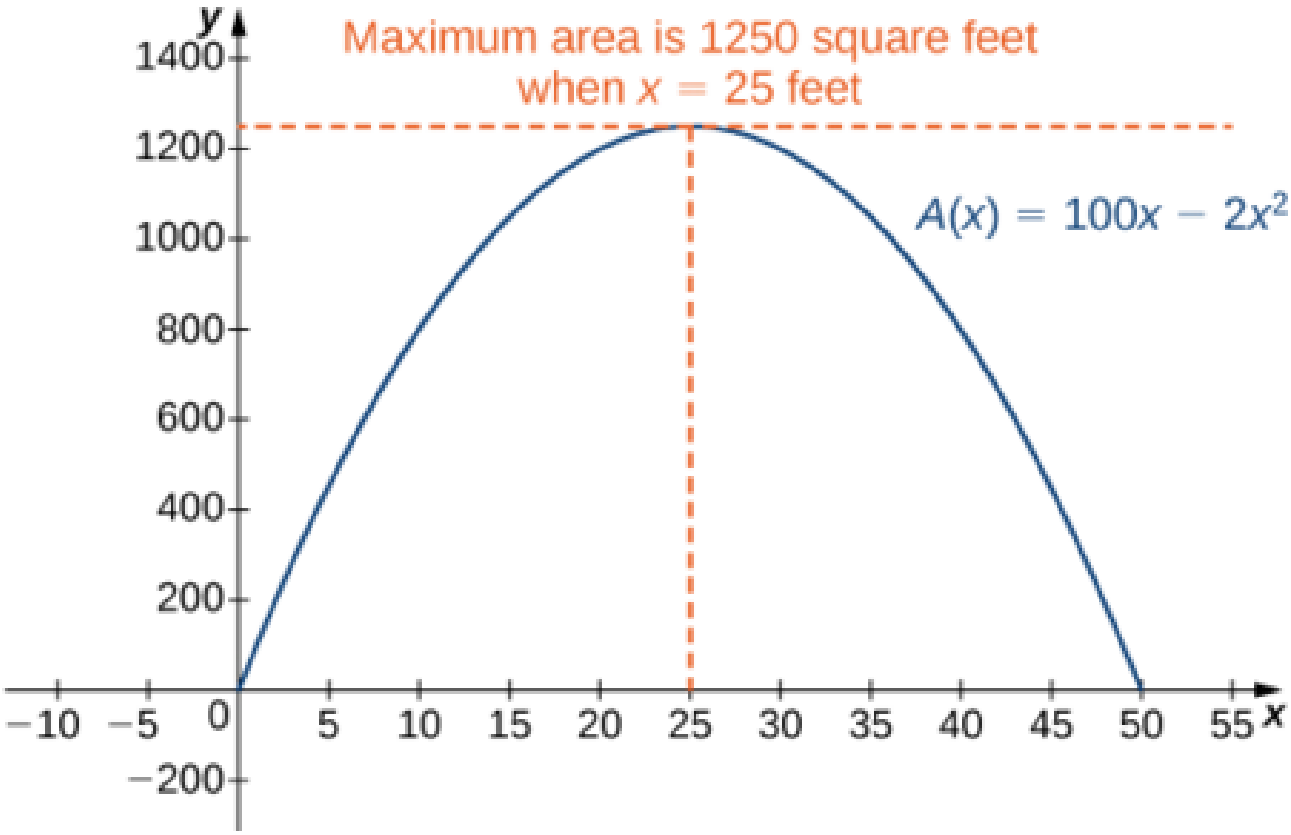


FIGURE 2.3 – To maximize the area of the garden, we need to find the maximum value of the function $A(x) = 100x - 2x^2$[14].

— Then we have $y = 100 - 2x = 100 - 2(25) = 50$. To maximize the area of the garden, let x=25ft and y=50ft. The area of this garden is 1250ft2[14].

2.4 Methods for solving optimization problems

Following the optimization problem, it may be solved by an exact method or an approximate method. (Figure 1.4[29])

2.4.1 Exact methods

— Exact methods (also called complete) produce an optimal solution for a given optimization problem instance ;

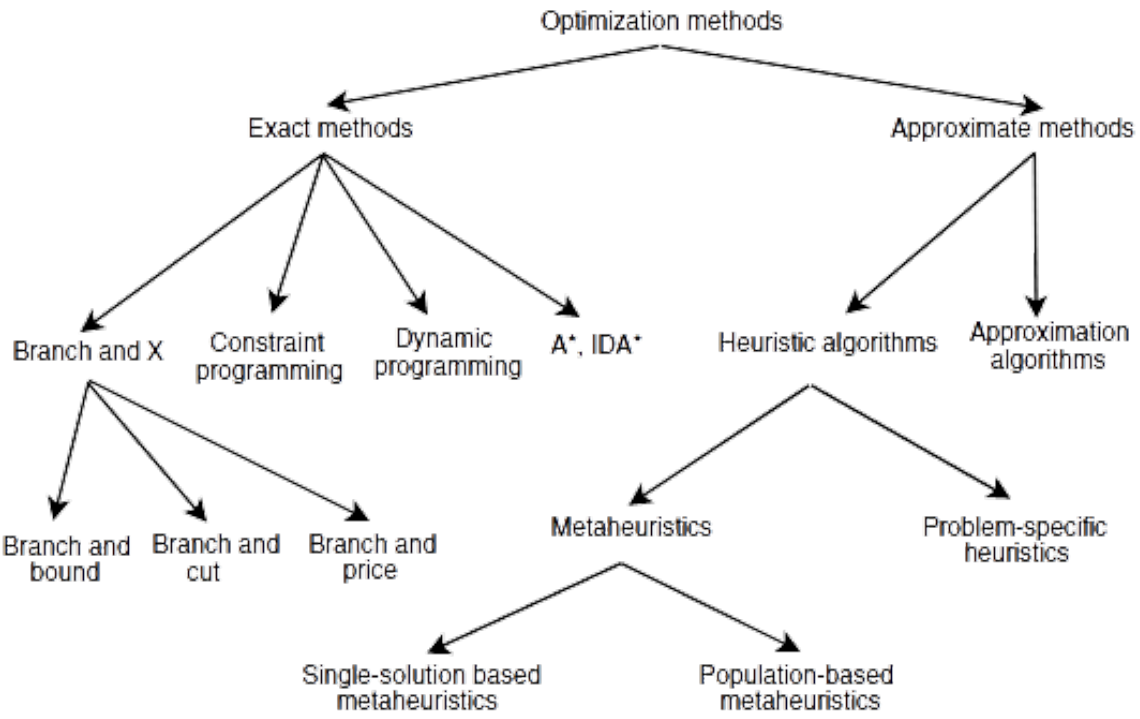


FIGURE 2.4 – Classical methods for optimization problems.

— They are used to find at least an optimal solution to a problem ;
— The most well-known exact algorithms are dynamic programming[24], Branch and bound[28] and A*[12], Constraint programming ;
— The major **disadvantage** of its methods is **the combinatorial explosion** The number of combinations increases with the increase in the size of the problem. The effectiveness of these algorithms is only promising for instances of small-sized problems[8].

Dynamic programming –Developed by Richard Bellman in the 1950s– is based on the recursive division of a problem into simpler subproblems. so "If the branch is optimal, the original is opti- mal" according to Bellman[29].

*Branch and bound and A** is an algorithm design paradigm which is gen- erally used for solving discrete optimization problems. These problems are typically exponential in terms of time complexity and may require ex- ploring all possible permutations in worst case[29].

Constraint programming –Jaffar and Lassez in 1987– is also an efficient approach to solving and opti- mizing problems that are too irregular for mathematical optimization. This includes time tabling problems, sequencing problems, and allocation or ros- tering problems[29].

2.4.1.1 Heuristics

Heuristics find "good" solutions on large-size problem instances. They allow to obtain acceptable performance at acceptable costs in a wide range of problems. In general, heuristics do not have an approximation guarantee on the obtained solutions[29].

2.4.1.2 Meta-heuristics

The basic properties of metaheuristics are[26]

— Metaheuristics are strategies to guide the search for an optimal solution ;
— The goal of metaheuristics is to explore the research space effectively in order to determine (almost) optimal solutions ;
— The techniques that make up meta-heuristic algorithms range from simple local search procedures to complex learning processes ;
— Their goal is to achieve a global optimum while avoiding local ones ;
— Metaheuristics are generally non-deterministic and provide no guarantee of optimality ;

— Metaheuristics can contain mechanisms to avoid being blocked in areas of the research space ;
— Metaheuristics can use heuristics that take into account the specificity of the problem being treated, but these heuristics are controlled by a higher level strategy ;
Metaheuristics are methods that can be divided into two classes :

1. **Single Solution Metaheuristics :** These methods process only one solution at a time, in order to find the optimal solution ;
2. **metaheuristics to population of solutions :** These methods use a population of solutions at each iteration until the overall solution is obtained ;

2.5 Heuristics and meta-heuristics

Comparisons	Heuristics	Metaheuristics
Definition	A strategy that uses information about the problem being solved to find promising solutions.	Similar to heuristics, metaheuristics aims to find promising results for a problem. However, the algorithm used in metaheuristics is general and can deal with different problems
Target	Not necessarily finding the perfect solution but just finding a good enough solution.	Finding promising solutions to different problems.
Characteristics	Problem-based design : means that heuristics is designed for a specific problem (one-to-one relationship)	Independent design of the problem : ability to use the same metaheuristics algorithm to solve countless problems by simply adjusting their input
	Non-measurable success : different from approximate algorithms, heuristics don't provide a clear indication of how close (or far) the obtained result is to the optimal one.	
	Reasonable execution : the time or computational resources for executing a heuristic must never exceed the requirements of any exact method that solves the same problem.	
Depend	gready search-based.	Genetic algorithms, particle swarm optimization, simulated annealing...

TABLE 2.3 – Heuristics and meta-heuristics comparative table[13]

In this section, we'll take heuristics, metaheuristics algorithms. We'll focus on their definition, Target, Characteristics, and Depend.

2.6 Classification of metaheuristics

Many grading criteria can be used for metaheuristics :
— **Nature inspired versus non nature inspired :** Many metaheuris- tics are inspired by natural processes : evolutionary algorithms and artificial immune systems from biology ; ants, bees colonies, and parti- cle swarm optimization from swarm intelligence into different species (social sciences) ; and simulated annealing from physics ;

— **Memory usage versus memoryless methods :** Some metaheuris- tic algorithms are memoryless ; that is, no information extracted dy- namically is used during the search. Some representatives of this class are local search, GRASP, and simulated annealing. While other meta- heuristics use a memory that contains some information extracted on- line during the search. For instance, short-term and long-term mem- ories in tabu search ;
— **Deterministic versus stochastic :** A deterministic metaheuristic solves an optimization problem by making deterministic decisions (e.g., local search, tabu search). In stochastic metaheuristics, some random rules are applied during the search (e.g., simulated annealing, evo- lutionary algorithms). In deterministic algorithms, using the same initial solution will lead to the same final solution, whereas in stochas- tic metaheuristics, different final solutions may be obtained from the same initial solution. This characteristic must be taken into account in the performance evaluation of metaheuristic algorithms ;
— **Population-based search versus single-solution based search :** Single-solution based algorithms (e.g., local search, simulated anneal- ing) manipulate and transform a single solution during the search while in population-based algorithms (e.g., particle swarm, evolutionary al- gorithms) a whole population of solutions is evolved. These two fam- ilies have complementary characteristics : single-solution based meta- heuristics are exploitation oriented ; they have the power to inten- sify the search in local regions. Population-based metaheuristics are ex- ploration oriented ; they allow a better diversification in the whole search space. In the next chapters of this book, we have mainly used this classification. In fact, the algorithms belonging to each family of metaheuristics share many search mechanisms ;
— **Iterative versus greedy :** In iterative algorithms, we start with a complete solu- tion (or population of solutions) and transform it at each iteration using some search operators. Greedy algorithms start from an empty solution, and at each step a decision variable of the problem is assigned until a complete solution is obtained. Most of the metaheuristics are iterative algorithms[29].

2.7 Biogeography-based optimizatio approach :

The problems in NP-hard is not easy to solve using an exact method as the scale getting larger. Nowadays, the number of research which develop and apply metaheuristics methods in solving combinatorial problems are quite high. One of new metaheuristics method which adopt biogeography phenomenon is Biogeography-based Optimization (BBO).

Biogeography-based optimization is a metaheuristics method that newly introduced by Simon. **BBO algorithm**[19] is able to generate a competitive solution for unimodal or multimodal functions compared with other metaheuristics algorithms based on populatio. BBO is able to solve 57 out of 75 instances. BBO has a better performance compared to Particle Swarm Optimization (PSO) with solving 19 out of 75. But, compared to modified Genetic Algorithm which can solve 67 instances, BBO is not better.

Nowadays, there are high numbers of research developing metaheuristics algorithm for solving NP-hard problems. The most popular of metaheuristics are those which based on biology (biology based). These methods adopt nature phenomenon and animal behaviour in daily life. Biology based methods, among of them are, Particle Swarm Optimization, Ant Colony Optimization, Genetic Algorithm, and Biogeography-based Optimization[19].

Biogeography describes the distribution and speciation of biological organisms, and is typically viewed as a process that maintains species equilibrium in their habitats. Equilibrium is achieved when the immigration/speciation and emigration/extinction rates are equal[19].

2.7.0.1 The Structure of Standard BBO

— As with any other Evolutionary algorithms (EAs), we begin with an optimization ;
— Each solution is composed of features, or independent variables ;
— A good solution corresponds to a biological habitat that is well suited for life ;
— poor solution corresponds to a habitat that is poorly suited for life. problem and a population of candidate solutions[19].

Like other EAs, BBO includes two steps : information sharing and mutation. In BBO, information sharing is implemented with migration.

BBO migration is probabilistic. Each solution's migration rate is used to stochastically share features.

Mutation is a probabilistic function that can modify solution features, and can be implemented as in any other EA. The purpose of mutation is to increase diversity among the population[19].

2.8 Conclusion

A heuristic is a technique designed for solving a problem more quickly when classic methods are too slow, or when classic methods fail to find any exact solution. The objective of a heuristic is to produce a solution in a reason- able time frame that is good enough for solving the problem at hand. This solution may not be the best of all the actual solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because finding it does not require a prohibitively long time.

Meta-heuristics are strategies that "guide" the search process. The goal is to efficiently explore the search space in order to find (near-) optimal solutions. Metaheuristic algorithms are approximate and usually nondeterministic. Meta-heuristics are not problem-specific and may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategies. Meta- heuristics can obtain better quality solutions than heuristic methods[9].

Chapitre 3

Adaptive BBO for solving shortest path problem

3.1 Introduction

In this chapter we propose to apply approximate method from a metaheuristics, exactly the BBO(Biogeography-Base Optimization) algorithm, for solving big instances of a hard combinatorial problem, which requires optimization when exact methods are very time consuming, in the ad-hoc networks (MANETs) exactly the shortest path(SP) problem.

3.2 Shortest path problem definition

The shortest path problem in networks is finding the most effi- cient path between two nodes within the network, with the minimum distance between the source node and the destination node in the network[1]. Between these knots are edges where each edge has nu- merical weight that represents the distance associated with crossing that edge[20]

3.3 Objective function

In the context of ad hoc networks, the objective function in the shortest path problem typically refers to the metric or criteria used to determine the "shortest" or most optimal path between two nodes. In this case we will choose :
Minimum Distance : This objective function seeks to find the path with the shortest physical distance between the source and destination.

3.4 Solution encoding

In the problem that we have, is to reduce the distance to cross from the source node to the destination node by finding the shortest path. We want to represent the solution in a specific format, which involves converting the solution into a simple form that can be easily understood and entered into the computer :

3.4.1 Network representation

We will define the network we have in the form of a graph, and to process it effectively by computer later we will enter it as a matrix.

3.4.2 Encoding path

We encode a path (route) by listing up node IDS from its source node to its destination. Each node in the network is represented as a separate island in the BBO algorithm. Islands correspond to potential solutions (routes) connecting source and destination nodes. An island (solution) can be encoded as a sequence of nodes representing the path from the source to the destination. The encoding can be implemented as an array or a linked list or a list[21].

3.4.3 Fitness Evaluation

The quality of a island (solution) is measured by a fitness function. Here, the fitness function is obvious as the goal is to find the minimal cost path. Thus, the fitness of ith island is defined as :

$$f_i = \left(\sum_{j=1}^{N_i - 1} C_{yz} \right)^{-1}, y = \mathbf{PP}^i(j) \quad \text{and} \quad z = \mathbf{PP}^i(j+1)$$

FIGURE 3.1 – Fitness Calculation Process[30]

— PP^i is the set of sequential node IDs for ith island ;
— Ni = $|PP_i|$ = number of nodes that constitute the path represented by ith island ;
— C_{yz} is the cost of the link connecting node y and node z.

Thus, the fitness function takes maximum value when the shortest path is obtained. If the path represented by a island happens to be an invalid path, then its fitness is assigned a penalty value (=0).

3.5 Adaptive BBO algorithm to solve a shortest path problem

Algorithm 1 Algorithm Path

```
data :
nodes : list of integers
fitness : integer
constructor Path(nodes) :
this.nodes = nodes
this.fitness = calculateFitness()
method calculateFitness() :
totalDistance = 0
for i = 0 to nodes.size() - 2 do : do
    totalDistance += graph[nodes[i]][nodes[i + 1]]
end for
return totalDistance
method compareTo(other) :
if this.fitness < other.fitness then then
    return -1
else if this.fitness > other.fitness then then
    return 1
else
    return 0
end if
```

— The **Path** class represents a path consisting of a sequence of nodes. It has two member variables : **nodes** and **fitness**. nodes is a list of integers that represents the sequence of nodes in the path. **fitness** is an integer that represents the fitness or cost of the path ;
— The **constructor Path(List<Integer> nodes)** initializes a **Path** object with the given list of nodes. It assigns the **nodes** parameter to the **nodes** member variable. Additionally, it calls the **calculateFitness()** method to compute and assign the fitness value for the path.

Algorithm 2 Calculate Fitness

```
fun Int calculateFitness(nodes : List<Int>, graph : Array<IntArray>) :
for i = 0 ; i < nodes.size - 1 do
    totalDistance += graph[nodes[i]][nodes[i + 1]]
end for
return totalDistance
```

–6: The **calculateFitness()** function takes a list of integers **nodes** and a two-dimensional array **graph** as input. It calculates the fitness of a path by iterating through the **nodes** list and summing the distances between consecutive nodes using the **graph** array. Finally, it returns the **totalDistance** as the fitness value of the path.

Algorithm 3 Compare Paths

```
1: fun Int Path.compareTo(other : Path) :
2: return this.fitness.compareTo(other.fitness)
```

— The **compareTo()** method is overridden from the **Comparable** interface. It allows instances of the **Path** class to be compared to each other based on their fitness values. The method compares the fitness of the current **Path** object (**this.fitness**) with the fitness of the **other Path** object passed as a parameter. It returns a negative integer, zero, or a positive integer depending on whether the current object's fitness is less than, equal to, or greater than the **other** object's fitness, respectively ;
— By implementing the **Comparable** interface and overriding the **compareTo()** method, instances of the **Path** class can be easily sorted or ordered based on their fitness values, which can be useful in various algorithms that involve finding the best or optimal paths ;
— Overall, this code defines a **Path** class that represents a path in a graph, calculates the fitness of the path based on distances between nodes, and provides a mechanism to compare paths based on their fitness values.

Algorithm 4 Generate Random Path

```
1: path < − empty list of integers
2: for i < − 0 to size of graph - 1 do : do
3:    Add i to path
4: end for
5: for i < − size of path - 1 down to 1 do : do
6:    j < − random integer between 0 and i
7:    temp < − path[i]
8:    path[i] < − path[j]
9:    path[j] < − temp
10: end for
11: Return path
```

— The **generateRandomPath()** function takes a two-dimensional array **graph** as input and returns a randomly generated path as a list of integers ;
— We initialize an empty mutable list **path** and then iterate from 0 until the size of the **graph** array. For each iteration, we add the current index to the **path** list ;
— Next, we create a **Random** object and iterate from **path.size - 1** down to 1. In each iteration, we generate a random index **j** using **random.nextInt(i + 1)** and perform a swap operation to shuffle the elements in the **path** list ;
— Finally, we return the shuffled **path** list.

Algorithm 5 Initialize Population

```
1: function initializePopulation(populationSize)
2: Val population =
3: for i = 1, populationSize do do
4:    Val path = generateRandomPath()
5:    table.insert(population, Path(path))
6: end for
7: return population
```

— the **initializePopulation()** function takes an **populationSize** integer as input and returns a list of **Path** objects representing the initialized population ;
— We create an empty mutable list **population** to store the paths. Then, we iterate from 0 until **populationSize**, generating a random path using the **generateRandomPath()** function and creating a new Path object with the generated **path**. We add the **Path** object to the **population** list ;
— Finally, we return the populated **population** list.
— We assume that **migrationRate** and **populationSize** are defined variables ;

Algorithm 6 Perform Migration

```
fun List< Path > performMigration(population : MutableList< Path >) :
val migrationSize = (migrationRate * populationSize).toInt()
val immigrationSize = (migrationRate * populationSize).toInt()
for (i = 0 ; i < migrationSize) do
    val emigrant = population[i]
    val immigrantIndex = Random.nextInt(migrationSize, populationSize)
    val immigrant = population[immigrantIndex]
    population[i] = immigrant
    population[immigrantIndex] = emigrant
end for
return population
```

— The **performMigration()** function takes a mutable list population of **Path** objects as input and returns the modified population after performing migration ;
— We calculate the **migrationSize** and **immigrationSize** based on the migration rate and population size ;
— Then, we iterate from 0 until **migrationSize**. For each iteration, we select the **emigrant** from the **population** list at index **i**. We use **Random.nextInt()** function to generate a random **immigrantIndex** between **migrationSize** and **populationSize**, and then select the corresponding **immigrant** from the **population** list ;
— Next, we swap the **emigrant** and **immigrant** in the **population** list by assigning **immigrant** to the **i** index and **emigrant** to the **immigrantIndex** ;
— Finally, we return the modified **population** list.

Example of migration

Let's consider two islands (solutions) represented as paths in a graph : Island 1 : $A- > B- > C$ and Island 2 : $A- > D- > C$ **Figure 4**. Here's an example of a migration operation[21] :

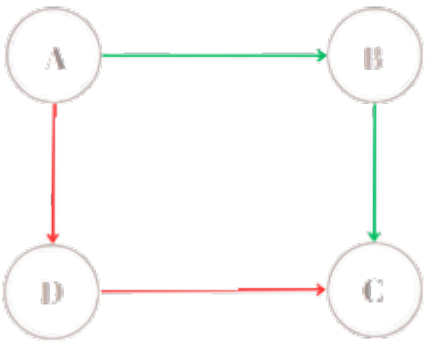


FIGURE 3.2 – Island 1 : $A- > B- > C$ and Island 2 : $A- > D- > C$.

Migration : Immigration from Island 2 to Island 1
— Calculate the fitness of the paths in Island 1 and Island 2 ;
— Based on a migration probability or fitness comparison, select a solution from Island 2 (e.g., $A- > D- > C$) to migrate to Island 1 ;
— Replace a solution in Island 1 with the migrated solution ;
— The updated Island 1 could be $A- > D- > C$.
— We assume that **mutationRate**, **populationSize**, and **graph** are defined variables ;
— The **performMutation()** function takes a mutable list **population** of **Path** objects as input and returns the modified population after performing mutation ;
— We iterate from 0 until **populationSize**. For each iteration, we check if a random double value generated by **Random.nextDouble()** is less than the **mutationRate**. If it is, we perform mutation on the **Path** object at index **i** in the **population** list ;
— Inside the mutation block, we select two random indices, **index1** and **index2**, using **Random.nextInt()** function. Then, we swap the values at those indices in the **nodes** list of the selected **Path** object ;

Algorithm 7 Perform Mutation

```
fun performMutation(population : MutableList< Path >) : List< Path >
for (i = 0 ; i< populationSize) do
  if Random.nextDouble() < mutationRate then
    val path = population[i]
    val index1 = Random.nextInt(graph.size)
    val index2 = Random.nextInt(graph.size)
    val temp = path.nodes[index1]
    path.nodes[index1] = path.nodes[index2]
    path.nodes[index2] = temp
    path.fitness = path.calculateFitness()
  end if
end for
return population
```

— After the mutation, we recalculate the fitness of the mutated **Path** object by calling the **calculateFitness()** method and update the **fitness** field accordingly ;
— Finally, we return the modified **population** list.

Example of mutation

Consider a graph with nodes A, B, C, and D, and the current solution is represented as the path $A- > B- > C- > D$ **Figure 2**. Here's an example of a mutation operation[21] :

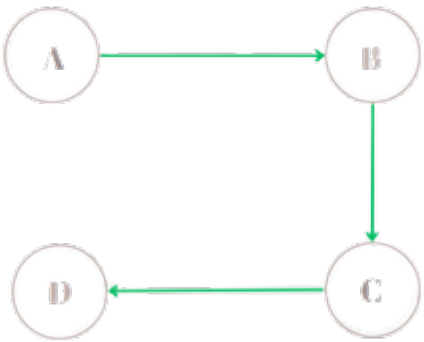


FIGURE 3.3 – path : $A- > B- > C- > D$

Mutation : Swap Mutation
— Randomly select two nodes in the path (e.g., B and C) ;
— Swap the positions of the selected nodes ;
— The mutated solution could be $A- > C- > B- > D$ **Figure 3**.

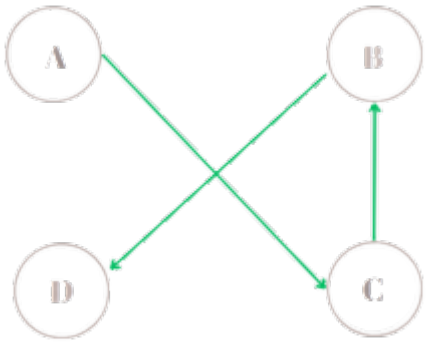


FIGURE 3.4 – Path : $A- > C- > B- > D$

Chapitre 4

Results and Experimentation

4.1 Introduction

The Biogeography-Based Optimization (BBO) algorithm is a population-based optimization technique that uses migration and mutation operations to find the global optimum solution. In this research, a Java application was developed to implement the BBO algorithm for finding the shortest path through a graph. The application includes a graphical user interface for user interaction and allows users to input their own test data. The performance of the BBO algorithm was compared to that of Ant Colony Optimization (ACO) for the same problem. This document provides information on the development environment, programming languages, and presentation of the Java application, as well as details on the hardware and software environment, test data, and experimental results.

4.2 Development environment

4.2.1 Hardware environment

Personal Computer (Laptop)
- Graphics cards : Intel(R) HD graphics ;
- network cards : broadcom network card 802.11n2 ;
- keyboards : standard keyboard ps/2 ;
- Computers : PC with x86 ACPI processor ;
- Processors : Intel(R) pentium(R) CPU B960 @ 2.20GHz

4.2.2 Software environment

Overleaf
- Online LaTeX editor for scientific documents ;
- Collaboration features for real-time co-authoring ;
- Integration with reference managers and publishing platforms.

NetBeans IDE 8.2
- Java Development : NetBeans IDE 8.2 is a feature-rich Integrated Development Environment specifically designed for Java development, providing a comprehensive set of tools and features for writing, debugging, and testing Java applications ;
- Cross-Platform Compatibility : NetBeans IDE 8.2 is compatible with multiple operating systems, including Windows, macOS, and Linux, allowing developers to work seamlessly across different platforms ;
- Plugin Ecosystem : NetBeans IDE 8.2 supports a wide range of plugins, enabling developers to extend and customize their IDE with additional functionality. It offers support for various programming languages, frameworks, and tools, enhancing the flexibility and adaptability of the development environment.

Background Removal Tool
- AI-powered background removal : AI-based tool that automatically detects and removes backgrounds from images ;
- User-friendly interface : Easy-to-use platform for uploading and processing images with a few simple clicks ;
- Instant results : Quickly generates images with transparent backgrounds in seconds, ready for download.

4.3 Programming languages

In this application I used the programming language **Java version 1.8.0-341**, java 1.8.0-341, commonly referred to as Java 8 Update 341, is a specific version of the Java programming language. It is a stable release within the Java 8 series, which introduced significant enhancements and new features to the Java language and platform. Java 8 is known for introducing lambda expressions, functional interfaces, the Stream API, and other improvements that enable more concise and expressive code. Java 8 Update 341 indicates that it is the 341st update of Java 8, incorporating bug fixes and security patches since its initial release.

4.4 Presentation of the application

4.4.1 Application Function

The app in our hands is the GUI (Graphical User Interface) application that implements the BBO algorithm to find the shortest path through the graph.

Here's a breakdown of the application's functionality :

1. Graph Representation :
   - The application defines a static two-dimensional array called graph to represent the weighted graph ;
   - The graph represents the distances between different nodes.

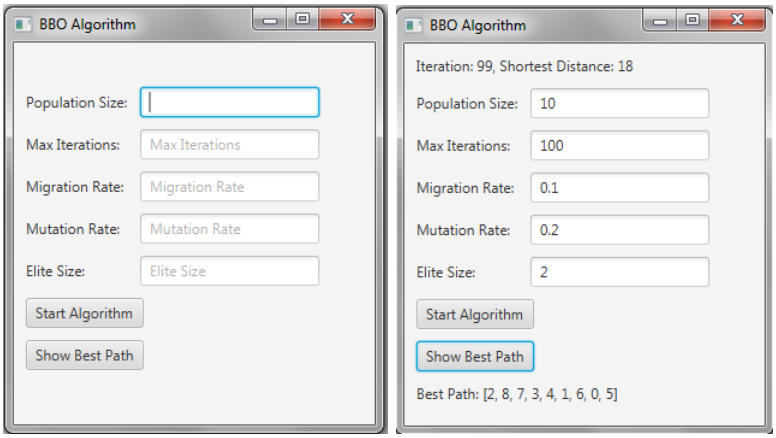


FIGURE 4.1 – Graphic user interface(GUI) to handle an algorithm BBO

2. BBO Algorithm :
   - The application implements the BBO algorithm to find the shortest path in the given graph ;
   - The algorithm works with a population of paths, where each path represents a possible solution ;
   - The fitness of each path is calculated based on the total distance covered ;
   - The algorithm performs operations such as migration and mutation to evolve the population over a specified number of iterations.
3. GUI Interface :
   - The application uses JavaFX to create a graphical user interface for interacting with the BBO algorithm ;
   - The interface includes text fields to input algorithm parameters such as population size, max iterations, migration rate, mutation rate, and elite size ;
   - It also includes buttons to start the algorithm and show the best path ;
   - The algorithm's progress and the shortest distance found are displayed in a label.

Overall, this Java application combines the BBO algorithm with a user-friendly GUI interface to solve the shortest path problem in a given graph.

4.4.2 Main features of the application

The BBO (Biogeography-Based Optimization) algorithm implemented in the provided Java application has the following main advantages :

1. Global Optimization : BBO aims to find the global optimum solution by exploring the entire search space, avoiding local optima ;

2. Flexibility : BBO can be applied to different optimization problems, making it versatile across various domains ;
3. Population-Based Approach : BBO uses a population of solutions, allowing for diverse exploration and parallel search. This increases the chances of finding better solutions and provides robustness ;
4. Evolutionary Operators : BBO employs migration and mutation operations. Migration promotes diversity and information sharing, while mutation introduces randomness and enables exploration ;
5. Efficient Convergence : BBO preserves the best solutions in each iteration, facilitating fast convergence towards the optimal solution ;
6. User Interaction : The Java application combines BBO with a graphical user interface, enabling users to input parameters, visualize progress, display the best path, and control the execution.

4.5 Test data

The code in our hands applies a Biogeography-based optimization algorithm to solve the shortest path problem. The test data is represented by

— A two-dimensional matrix of 9 nodes (0 to 8) represents each element of the matrix the distance between the nodes, except for the value of 0 which means that there is no direct connection between the nodes.

The code also requires many input by the user to configure and operate the algorithm for example :

— Population Size : It is an parameter that determines the size of the population used in the BBO algorithm ;
— Max Iterations : It is an parameter that determines how many times the algorithm will iterate before stopping ;
— Migration Rate : It is an parameter that determines how often individuals move from one population to another ;
— Mutation Rate : It is an parameter that determines how often mutations will occur in the population ;
— Elite Size : It is an parameter that determines how many of the fittest individuals are carried over to the next generation.

These inputs are used to configure and operate the algorithm, which in turn generates a range of potential solutions (paths), Then she assesses her fitness and selects the fittest individuals to reproduce and create the next generation, the process is repeated for a number of repeats until a satisfactory solution is found.

4.6 Experimental results and discussion

After we implemented the code several times we got a set of information as shown in Table 1. This information allows us to discuss the results of the application in terms of : **Execution time**, **the number of steps needed to complete the algorithm**, and the quality of the solutions (**fitness function**).

Execution time

The execution time in the provided code can vary based on several factors. Here's an overview of when it may increase or decrease :

1. Increase :
   — As the population size (populationSize) increases, the execution time may increase because more paths need to be evaluated and manipulated during each iteration ;
   — Increasing the maximum number of iterations (maxIterations) will also lead to longer execution times since the algorithm needs to run for a longer period ;
   — A higher migration rate (migrationRate) will cause more paths to be migrated between subpopulations, potentially increasing the execution time ;
   — A larger mutation rate (mutationRate) will result in more paths being mutated, leading to additional fitness calculations and comparisons.
2. Decrease :
   — A smaller population size, fewer iterations, and lower migration and mutation rates will generally reduce the execution time ;
   — Decreasing the elite size (eliteSize) will decrease the time required for selecting and copying the elite paths in each iteration.

Number of steps

Overall the number of steps corresponds to the number of iterations performed by the algorithm. Each iteration involves :

— Sort population ;
— Select elite paths ;
— Perform migration ;
— Perform mutation ;
— Add elite paths back to the population.

Quality of Solutions

In the context of the studied algorithm (BBO), the fitness function can be considered a basic standard for evaluating the quality of solutions.

— The fitness value represents the total distance traveled along the path ;
— It sums up the distances between consecutive nodes in the path using the graph variable, which represents the distances between nodes ;
— Lower fitness values indicate better solutions, as they represent shorter total distances.

4.6.1 Comparison of the two appropriate methods

In this part, we try to compare two metaheuristics algorithms, Biogeography-based optimization and Ant Colony Optimization, to the problem of finding the shortest path because both are nature-inspired and based-population, and optimization algorithm.

	BBO Algorithm	ACO Algorithm
Similarities between the two algorithms	- Both applications are designed to solve the Shortest path Problem. - Both algorithms employ a population-based approach. - The fitness of each candidate solution (path) is evaluated based on its total distance or cost. - Both algorithms use random numbers for generating initial paths, selecting individuals for migration, and performing mutations.	
Differences between the two algorithms	-Implements Biogeography-Based Optimization (BBO) algorithm. -Solution represented as a list of integers. -Specific parameters : population size, migration rate, mutation rate, elite size. -Initializes population with random paths. -Inspired by biogeographical distribution and migration patterns.	-Implements Ant Colony Optimization (ACO) algorithm. -Solution represented as a pheromone matrix -Specific parameters : number of ants, pheromone evaporation rate, deposition rate, heuristic information -Complexity influenced by number of ants and iterations -Requires initialization of a pheromone matrix -Based on the behavior of ants and collective intelligence
Population size	10	10
Max iteration	100	100
Migration Rate	0.1	/
Mutation Rate	0.2	/
Elite Size	2	/
Number of steps needed to complete the algorithm	100 Steps	100 Steps

TABLE 4.1 – Comparison of BBO and ACO Algorithms for the Shortest Path Problem

After discussing and analysing the results of the application, we were informed that :
For Execution time : Increases as population size or number of iterations increases, and in the event of an increase in the rate of migration as it causes the migration of more paths among the populations, Increasing the rate of mutations also leads to the evolution of more paths and thus additional comparisons of fitness function and thus increased execution time. Reducing elite size will reduce the time required to choose and copy elite tracks in each repeat, therefore reducing execution time.
As for the **number of steps needed to implement the algorithm** it is only affected by the number of iterations.
The **quality of solutions**quality of solutions leavings directly dependent by the size of the elite.

BBO / ACO			
Graph	**Execution Time(ms)**	**Fitness Function**	**Best Path**
G[9][9]	20 ms / 61 ms	12 / 6	[7, 5, 2, 1, 4, 8, 3, 6, 0] / [4, 8, 4, 2, 0, 6, 6, 8, 8]
G[8][8]	22 ms/ 37 ms	22 / 6	[7, 2, 3, 1, 5, 4, 0, 6] / [5, 3, 5, 5, 7, 7, 3, 0]
G[10][10]	19 ms / 45 ms	15 / 21	[6, 9, 0, 1, 2, 3, 8, 5, 7, 4] / [5, 8, 9, 0, 2, 6, 2, 0, 3, 4]
G[7][7]	19 ms / 31 ms	4 / 7	[3, 5, 0, 1, 6, 4, 2] / [5, 5, 3, 5, 4, 0, 5]
G[6][6]	19 ms / 30 ms	0 / 0	[1, 2, 3, 4, 5, 0] / [1, 2, 3, 1, 5, 5]
G[12][12]	20 ms / 33 ms	43 / 17	[5, 2, 6, 1, 11, 8, 3, 9, 7, 0, 4, 10] / [3, 10, 10, 4, 1, 1, 4, 0, 9, 3, 0, 11]
G[5][5]	21 ms / 34 ms	22 / 0	[2, 4, 0, 1, 3] / [1, 2, 2, 3, 3]
G[11][11]	14 ms / 38 ms	25 / 8	[7, 2, 9, 4, 0, 5, 3, 10, 8, 1, 6] / [1, 4, 7, 6, 7, 3, 6, 5, 1, 6, 1]
G[13][13]	20 ms / 36 ms	11 / 7	[9,1,7,10, 5, 6, 0, 11, 12, 8, 3, 4, 2] / [12, 5, 4, 9, 5, 10, 7, 9, 6, 2, 0, 12, 3]
G[14][14]	18 ms / 56 ms	50 / 4	[13, 0, 10, 3, 11, 6, 1, 12, 8, 5, 4, 2, 9, 7] / [3, 2, 8, 10, 2, 3, 9, 3, 12, 11, 10, 0, 11, 11]
Average	19,2 ms / 40,1 ms	20,4 / 7,6	

TABLE 4.2 – Make 10 examples and calculate the average of the differences between the solutions of bbo and that of Aco

4.7 Conclusion

The Java application we developed combines the Biogeography-Based Optimization (BBO) algorithm with a user-friendly GUI interface, enabling users to input parameters, visualize progress, display the best path, and control the execution. Our experimental results showed that the BBO algorithm outperformed Ant Colony Optimization (ACO) in finding the shortest path with lower fitness values. The BBO algorithm demonstrated its advantages of global optimization, flexibility, population-based approach, evolutionary operators, and efficient convergence. Overall, this research contributes to the field of optimization algorithms and provides a useful tool for solving the shortest path problem in various domains.

General conclusion

In conclusion, the use of metaheuristics for solving optimization problems in Ad-Hoc Mobile Networks has shown great promise. This paper has provided a comprehensive overview of the use of metaheuristics for solving routing problems in Ad-Hoc Mobile Networks. The paper covered topics such as complexity theory, optimization problems, methods for solving optimization problems, and classification of metaheuristics. The paper also provided an overview of MANET networks, their features, classification, and routing protocols.

The Adaptive BBO algorithm was presented as a solution to the shortest path problem in MANETs. The paper discussed the objective function, solution encoding, and fitness evaluation of the algorithm. Code examples were provided for implementing the BBO algorithm and initializing the population.

The performance of the BBO algorithm was compared to that of Ant Colony Optimization (ACO) for the same problem. The results showed that the BBO algorithm outperformed ACO in finding the shortest path with lower fitness values. The BBO algorithm demonstrated its advantages of global optimization, flexibility, population-based approach, evolutionary operators, and efficient convergence.

The use of metaheuristics in Ad-Hoc Mobile Networks has the potential to improve network efficiency, reduce energy consumption, and enhance overall network performance. The research presented in this paper provides a useful tool for solving the shortest path problem in various domains.

Overall, this paper serves as a valuable resource for researchers and practitioners interested in the application of metaheuristics in Ad-Hoc Mobile Networks. The paper provides a comprehensive overview of the use of metaheuristics for solving routing problems in Ad-Hoc Mobile Networks, and the Adaptive BBO algorithm is presented as a solution to the shortest path problem in MANETs. The paper also discusses the results of experimentation and comparison of two appropriate methods. The research presented in this paper has the potential to contribute to the development of more efficient and effective routing protocols in Ad-Hoc Mobile Networks.

Bibliographie

[1] Shortest path properties. Last Updated : 06 Jul, 2021.

[2] Types of complexity classes | p, np, conp, np hard and np complete. shorturl.at/nHRTZ. 31 Mar, 2023.

[3] What is the combinatorial optimization problem. shorturl.at/hlrTV.

[4] Algorithmic complexity. shorturl.at/pqCN1, February 2019. Accessed 2023-02-11.

[5] Chang Wook Ahn and Rudrapatna S Ramakrishna. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE transactions on evolutionary computation*, 6(6) :566–579, 2002.

[6] Mohammed Abdulhakim Al-Absi, Ahmed Abdulhakim Al-Absi, Mangal Sain, and Hoonjae Lee. Moving ad hoc networks—a comparative study. *Sustainability*, 13(11) :6187, 2021.

[7] Meryem Berghida and Abdelmadjid Boukra. Ebbo : an enhanced biogeography-based optimization algorithm for a vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows. *The International Journal of Advanced Manufacturing Technology*, 77 :1711–1725, 2015.

[8] Souha BRAHIMI, Mohamed El Amine MIMOUNE, et al. *Les Problèmes d'Ordonnancement Multi-Objectifs dans un Système de Production*. PhD thesis, UNIVERSITY of M'SILA, 2022.

[9] Sachin Desale, Akhtar Rasool, Sushil Andhale, and Priti Rane. Heuristic and meta-heuristic algorithms and their relevance to the real world : a survey. *Int. J. Comput. Eng. Res. Trends*, 351(5) :2349–7084, 2015.

[10] Johann Dréo, Alain Pétrowski, Patrick Siarry, and Eric Taillard. *Metaheuristics for hard optimization : methods and case studies*. Springer Science & Business Media, 2006.

[11] Falko Dressler et al. Self-organization in ad hoc networks : Overview and classification. *University of Erlangen, Dept. of Computer Science*, 7 :1–12, 2006.

[12] Daniel Foead, Alifio Ghifari, Marchel Budi Kusuma, Novita Hanafiah, and Eric Gunawan. A systematic literature review of a* pathfinding. *Procedia Computer Science*, 179 :507–514, 2021.

[13] Vinicius Fulber-Garcia. Heuristics vs. meta-heuristics vs. probabilistic algorithms. shorturl.at/ops26. Last modified : April 2, 2023.

[14] Vinicius Fulber-Garcia. Optimization problems. shorturl.at/jtuvy. Last updated Nov 10, 2020.

[15] Aditya Gupta, Prabal Verma, and Rakesh Singh Sambyal. An overview of manet : Features, challenges and applications. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 4(1) :122–126, 2018.

[16] Nassir Harrag and Abdelghani Harrag. Fuzzy-zrp : An adaptive manet radius zone routing protocol. *Engineering, Technology & Applied Science Research*, 13(2) :10601–10607, 2023.

[17] Hai Liu, Yiu-Wing Leung, and Xiaowen Chu. *Ad hoc and sensor wireless networks : architectures, algorithms and protocols*. Bentham Science Publishers, 2009.

[18] Jonathan Loo, Shafiullah Khan, and Ali Naser Al-Khwildi. Mobile ad hoc network. *Mobile Ad Hoc Networks*, page 1, 2016.

[19] Haiping Ma, Dan Simon, Patrick Siarry, Zhile Yang, and Minrui Fei. Biogeography-based optimization : a 10-year review. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(5) :391–407, 2017.

[20] Ali Moghanni and Marta Pascoal. The rough interval shortest path problem. In *Operational Research : IO 2019, Tomar, Portugal, July 22–24 20*, pages 53–64. Springer, 2021.

[21] Masaharu Munetomo, Yoshiaki Takai, and Yoshiharu Sato. A migration scheme for the genetic adaptive routing algorithm. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, volume 3, pages 2774–2779. IEEE, 1998.

[22] Chetana Hemant Nemade and Uma Pujeri. Comparative study and performance analysis of manet routing protocol. *International journal of electrical and computer engineering systems*, 14(2) :145–154, 2023.

[23] Fernando Peres and Mauro Castelli. Combinatorial optimization problems and metaheuristics : Review, challenges, design, and development. *Applied Sciences*, 11(14) :6449, 2021.

[24] Alessandra Rosso and Ezio Venturino. A dynamic programming approach to ecosystem management. *Algorithms*, 16(3) :139, 2023.

[25] Victor S.Adamchik. Algorithmic complexity. `shorturl.at/nHRTZ`, 2009. Accessed 2023-02-11.

[26] MEHENNI SAID. Résolution du problème de stockage de conteneurs dans un port par un algorithme d'optimisation du coronavirus.

[27] Budi Santosa and Ade Lia Safitri. Biogeography-based optimization (bbo) algorithm for single machine total weighted tardiness problem (smtwtp). *Procedia Manufacturing*, 4 :552–557, 2015.

[28] Chung-Ho Su and Jen-Ya Wang. A branch-and-bound algorithm for minimizing the total tardiness of multiple developers. *Mathematics*, 10(7) :1200, 2022.

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

[30] Jun Wang. A recurrent neural network for solving the shortest path problem. *IEEE Transactions on Circuits and Systems I : Fundamental Theory and Applications*, 43(6) :482–486, 1996.

[31] Qian Zhang, Lisheng Wei, and Benben Yang. Research on improved bbo algorithm and its application in optimal scheduling of micro-grid. *Mathematics*, 10(16) :2998, 2022.