UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: Path Planning Simulation in Controlled Environments using the Ant Colony Optimization Algorithm

Trabajo de integración curricular presentado como requisito para la obtención del título de
Ingeniero en Tecnologías de la Información

Autor:

Guarnizo Cabezas Oscar Vicente

Tutor:

Ph.D Israel Pineda

Urcuquí, marzo 2020

UNIVERSIDAD YACHAY TECH

Urcuquí, 10 de marzo de 2020

SECRETARÍA GENERAL
(Vicerrectorado Académico/Cancillería)
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2020-00009-AD

En la ciudad de San Miguel de Urcuquí, Provincia de Imbabura, a los 10 días del mes de marzo de 2020, a las 15:00 horas, en el Aula CHA-01 de la Universidad de Investigación de Tecnología Experimental Yachay y ante el Tribunal Calificador, integrado por los docentes:

Presidente Tribunal de Defensa	Dr. MANZANILLA MORILLO, RAUL , Ph.D.
Miembro No Tutor	Dr. CHANG TORTOLERO, OSCAR GUILLERMO , Ph.D.
Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.

Se presenta el(la) señor(ita) estudiante **GUARNIZO CABEZAS, OSCAR VICENTE**, con cédula de identidad No. **1718546219**, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, con el objeto de rendir la sustentación de su trabajo de titulación denominado: **PATH PLANNING SIMULATION IN CONTROLLED ENVIRONMENTS USING THE ANT COLONY OPTIMIZATION ALGORITHM**, previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.

Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:

Tipo	Docente	Calificación
Miembro Tribunal De Defensa	Dr. CHANG TORTOLERO, OSCAR GUILLERMO , Ph.D.	10,0
Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.	10,0
Presidente Tribunal De Defensa	Dr. MANZANILLA MORILLO, RAUL , Ph.D.	10,0

Lo que da un promedio de: **10 (Diez punto Cero)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

[illegible]

GUARNIZO CABEZAS, OSCAR VICENTE
Estudiante

[illegible]

Dr. MANZANILLA MORILLO, RAUL , Ph.D.
Presidente Tribunal de Defensa

[illegible]

Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.
Tutor

[illegible]

Dr. CHANG TORTOLERO, OSCAR GUILLERMO , Ph.D.
Miembro No Tutor

[illegible]

MEDINA BRITO, DAYSY MARGARITA
Secretario Ad-hoc

Autoría

Yo, **Oscar Vicente Guarnizo Cabezas**, con cédula de identidad 1718546219, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el autor del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, marzo 2020.

[illegible]

Oscar Vicente Guarnizo Cabezas
CI: 1718546219

Autorización de publicación

Yo, **Oscar Vicente Guarnizo Cabezas**, con cédula de identidad 1718546219, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Urcuquí, marzo 2020.

[illegible]

Oscar Vicente Guarnizo Cabezas
CI: 1718546219

Dedication

"To my family and my friends because their motivation and support helped me to reach this point in my life."

Acknowledgments

Foremost, I would like to express my sincere gratitude to my principal advisor Israel Pineda Ph.D., for the continuous support of my study and research, for his patience, responsibility, motivation, and enthusiasm. His guidance helped me in all the time of research and writing of this thesis. Admittedly without his help and assistance, the publication of the preliminary results of this project would not have been possible.

Besides, I would like to acknowledge Lorena de los Angeles Guachi Guachi Ph.D., who collaborated in the initial stages of my project. Her comments and guidance supported me to focus my work in the early stages. Her support was undoubtedly appreciated, particularly in the drafting and structure of the present document.

I would also like to thank my friends and colleagues, who gave me ideas and comments about my project. Mainly, I have to highlight the observations of Fernando Zhapa, Anthony Ramos, and Joseph Gonzalez, which were considered in this project.

Last but not least, I would like to fondly thank my family, who always motivated and supported me in my college life. Without their guidance, I would not have successfully completed this stage of my life.

Resumen

La *Planificación de Rutas* es un tema ampliamente estudiado debido a sus diversas aplicaciones en robótica, planificación de socorro, planificación de rutas comerciales e incluso en la industria de los videojuegos. En consecuencia, los métodos computacionales de planificación de rutas son diversos, buscan resolver problemas en entornos desconocidos hasta encontrar un camino con la navegación más fluida. Desafortunadamente, esta diversidad provoca que algunos métodos pasen por alto ciertos aspectos al atender un problema de propósito específico. En este proyecto, proponemos analizar características como el tiempo de ejecución, la adaptabilidad y las representaciones del entorno. La literatura muestra que las implementaciones actuales tienen limitaciones en algunos de estos aspectos. En muchos casos, algunas técnicas tienen un rendimiento satisfactorio en una o dos de estas características, pero una deficiencia en las demás. Por esta razón, el presente proyecto tiene como objetivo diseñar un algoritmo de planificación de ruta basado en textit Ant Colony Optimization (ACO), que considera mejoras para estas limitaciones.

El diseño y la parte experimental de este trabajo se basaron en un estilo incremental. En este sentido, propusimos un algoritmo básico, y luego agregamos algunas interacciones globales y locales para tratar algunos de los problemas. Luego, seleccionamos las configuraciones con el mejor rendimiento para definir nuestra propuesta final y compararla con otros métodos ya conocidos. Finalmente, mostramos algunos resultados en una simulación gráfica para mostrar su comportamiento adaptativo. El algoritmo ha demostrado resolver las limitaciones antes mencionadas. Mientras trabajábamos con una metaheurística, no siempre obtuvimos resultados óptimos, sino soluciones factibles lo suficientemente buenas para las aplicaciones de la técnica. Sin embargo, el algoritmo genera resultados prometedores con una precisión de 95% aproximadamente.

Palabras Clave: Planificación de Ruta, Optimización de Colonia de Hormigas, Inteligencia de Enjambres, Simulación Gráfica.

Abstract

Path Planning is a widely studied issue due to its several applications in robotics, relief planning, trade route planning, and even in the video game industry. Consequently, the computational methods of path planning are diverse, seeking to solve problems in unknown environments until finding a trail with the smoothest navigation. Unfortunately, this diversity provokes that some methods overlook certain aspects by attending a specific-purpose problem. In this project, we propose to analyze characteristics as the execution time, the adaptability behavior, and the environment representations. The literature shows that current implementations have limitations in some of these aspects. In many cases, some techniques have satisfactory performance in one or two of these features, but a deficiency in the others. For this reason, the present project aims to design a path planning algorithm based on *Ant Colony Optimization* (ACO), which considers improvements for these limitations.

The design and experimental part of this work were based on an incremental style. In this sense, we proposed a basic algorithm, and then we add some global and local interactions for dealing with some of the issues. Then, we selected the configurations with the best performance to define our final proposal and compare it with other already known methods. Finally, we display some results in a graphic simulation to showcase its adaptive behavior. The algorithm has proved to solve the aforementioned limitations. While working with ACO metaheuristic, we did not always obtain optimal results but good-enough feasible solutions that are fit for the applications of the technique. Nevertheless, the algorithm generates promising results with an accuracy around the 95% approximately.

Keywords: Path Planning, Ant Colony Optimization, Swarm Intelligence, Graphic Simulation.

Contents

Glossary

action An action is the process of change from one state to another state by a specific path planning algorithm or criteria. 3

ant undertaker It is a special type of ant which is in charge of the necrophoresis process. In the computational model, this ant controls the stuck condition reinforcing the necrophoresis value κ_{ij}. 47

cells A cell is a convex simple geometric figure with different shapes, such as squares, hexagons, and others. A cell is used to describe a state in cell decomposition representation. 4

entity An entity is a character, vehicle, robot, or representation of a person who performs the path planning activity. 1

episode An episode is defined as a completed iteration where all the ants already have tried to find a route.. 37

exhaustive search It is a search that analyzes all the states of a graph or another environment representation. This search often spends a long time in the computation of a solution. 25

exploitation It is the ability of an algorithm to focus on the best solution so far. It is the opposite concept of exploration. Thus, if the exploitation is greater, the algorithm will reinforce the current solution, without analyzing additional possible solutions. 55

exploration It is the ability of an algorithm to explore different feasible solutions in the searching space. Thus, if exploration in greater, the algorithm can find more variety of solutions.. 9, 55

extrusion The extrusion of a node provokes the displacement of that node outside the land relief. 51

feasible solution It is a good solution which can be found in a specific problem, not necessarily the optimal solution. 27

heuristic This term is used to define the predefined information, which is used in a problem-solving method. The additional measure or data can be expressed as the cost w_{ij} of an edge in a problem based on graphs. 8

intrusion The intrusion of a node provokes the insertion of that node inside the land relief. 51

maze environment A maze environment is an environment with many obstacles or fewer edges, forming a kind of maze. 54

metaheuristic It is a high-level problem-independent algorithm that provides a set of guidelines or policies to improve a heuristic algorithm. This process does not guarantee the generation of an optimal solution, only a feasible solution. 1

minimal spanning tree It is a sub-graph that contents all the nodes of the original graph without forming any cycle (in other words, a tree). The main feature of this sub-graph is that it contains the lowest possible accumulated weight. 54

necrophoresis It is a natural behavior that happens in real ants. In this event, the ant undertakers carry the dead bodies of colony members from the environment to the nest . 47

optimal solution It is the best solution which can be found in a specific problem. 19

pathfinding It is the ability to determine appropriate motion actions that lead to the desired goal. This approach commonly works with well-known environments. 1

pheromone It is a secreted or excreted chemical substance that provokes a social response in members of the same species. In the computational model, this pheromone is represented by a numerical value. 2

potential field In terms of path planning, a potential field is a three-dimensional representation of a two-dimensional environment. This representation is useful for describing the evolution or variation of space configuration (states). 5

pseudo-random generator It is a method used in ACO algorithms to generate pseudo-random path based on a heuristic measure.. 12

state A state is a representation of the position of an entity in the environment. 3

stigmergy It is an environmental mechanism to regulate the activity of independent actors or agents. 2, 7

time list It is a list or a data structure which stores the priority of an ant for taking a specific state in the environment. 10

visited list It is a list or a data structure which stores the nodes which have already been visited by the ants. 10

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Path planning, from an initial position to a final position, is a widely studied topic by its several applications in engineering problems such as the moving of autonomous robots [4] [5] [6], problems of routing vehicles in traffic [7] [8], natural disaster relief planning [9] [10], and even the moving of non-player characters in the video game industry [11] [12] [13]. For its grand utility, path planning faces different problems and challenges, some of them related to kinematic constraints, dynamic constraints, and decision algorithms. In this sense, this problem challenges researchers to overcome or simulate the ability of human beings to avoidance obstacles and to react to environmental changes in an accurate way.

The way of dealing with path planning is commonly subject to its immediate application or its constraints. Nevertheless, the researchers frequently summarize this problem from two points of view: pathfinding and obstacle avoidance, which can be seen as global and local problems, respectively. On the one hand, the pathfinding focuses on determining appropriate motion actions that lead to the desired goal. This approach commonly works with well-known environments. On another hand, the obstacle avoidance technique seeks to control the immediate reaction of an entity to possible changes or obstructions. This approach commonly works with a partially known or unknown environment. Depending on the approach; we could work in different environments representations such as graphs [14] [15], cell decomposition [16] [17], road map [18], [19], potential field [20] [21], or sampling-based [22] [23]. Besides, we could work with different aims in mind finding routes one to one, one to many, many to one or many to many.

In the present work, we dealt with the specific aim of finding routes one to one in a well-known environment with the use of graphs. The project proposal involves the use of an Ant Colony Optimization (ACO) algorithm for the pathfinding decision and obstacle avoidance. The ACO algorithm is a metaheuristic algorithm based on the natural behavior of ants which tend to find the shortest path when they find a food source [24] [25]. In comparison with the main algorithms, the ACO algorithm has some advantages such as the reduction of time, the immediate parallelization capacity, the collective behavior, and the spontaneous adaptation to new heuristic metrics. However, the ACO algorithm does not guarantee the convergence to the optimal solution, but only a feasible solution which, in many applications, is enough. Finally, we propose to add new features to the ACO basic algorithm to improve its convergence time, its adaptability behavior and its three-dimensional performance.

1.2 Natural Behavior of ACO

In nature, it has been studied the behavior of *Argentine Linepithema humile* ants [26]. These ants have a collaborative phenomenon know as Stigmergy. This behavior works in the following way. First, the ants start to perform a random walk until they reach a food source. Then, the ants return to the colony, releasing pheromones on the traveled path. Other ants feel these pheromones and tend to follow large concentrations of them. Ants repeat this process until they reach a route with a large number of pheromones. The pheromones evaporate by the environmental conditions. In this way, paths not so visited has less concentration of pheromones. The evaporation, therefore, ensures that the ants converge to paths with a high quantity of pheromones. We can describe this mechanism as positive feedback on shorter routes and negative feedback on longer paths [25].

1.3 Problem statement

Path planning is a quite diverse topic by its several approaches at global and local levels. In this sense, the problem promotes extensive research considering different approaches according to its specific applications. Nevertheless, some current approaches present limitations regarding the execution time, the adaptability, and the environment representations. In many cases, some implementations have satisfactory performance in one or two of these features, but a deficiency in the others. For these reasons, this project presents an approach based on the Ant Colony Optimization, which seeks to deal with these limitations.

1.4 Objectives

1.4.1 General Objective

Develop an adaptive path planning simulation able to adjust to environmental changes using Ant Colony Optimization Algorithm in a three-dimensional space.

1.4.2 Specific Objectives

- Implement global and local interactions over ACO algorithm to guarantee the adaptability and the convergence of the algorithm on large environments.
- Compare the proposal method with other methods found in the literature review.
- Visualize the results in a three-dimensional simulation in real time.

Chapter 2

Theoretical framework

In this chapter, the necessary concepts for understanding the present work are introduced. In this way, this chapter starts with the bases of path planning and the current panorama. Then, it is explained the mathematical background of the ACO algorithm and possibles variants of its principal features: parameters, movement choice, reinforcement, and evaporation techniques.

2.1 Path Planning

The explanation below is mostly based on the Path Planning chapter of Wheeled Mobile Robotics, retrieved from [1].

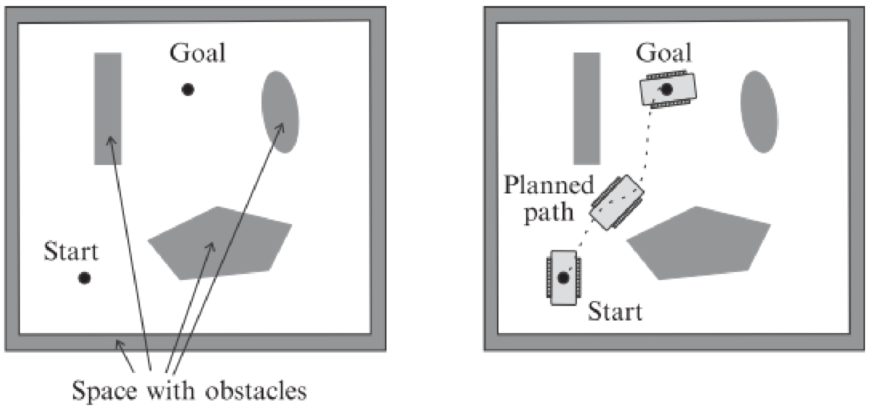


Figure 2.1: Path Planning Environment with Obstacles, retrieved from [1].

Path planning is the process of finding a continuous path that guides an entity from a start position to the final position. This complete path must fill in the free space of the environment (paths that do not collide with obstacles), Fig. 2.1. To this end, path planning considers the idea of ***states*** and ***actions***. The states give the positions of the entity in the environment, and they can be represented by several notions such points, cells, grids, and others. In this sense, an entity must always be in a state. On the other hand, an action is defined as the process for moving an entity from one state to another. A feasible path is, therefore, a sequence of actions that guide the entity from start state to the target, through some intermediate states. Thus, it could be possible to find several feasible solutions for each pair of states. The actions of an entity are subject to the path planning algorithm and used criteria. In this sense, the algorithm can take several directions based on the desired optimal solution:

- The optimal path must have the shortest possible length.
- The optimal path is the one in which the entity can pass in the shortest time.
- The optimal path should be as far as possible from the obstacles.
- The optimal path must be smooth without sharp turns.
- The path must consider motion constraints (e.g. where at the current time, not all directions are possible).

Besides, the path planning can take additional directions based on the primary purpose of the algorithm.

- **Start and Final Positions**: algorithms with solutions one to one, one to many, many to one, or many to many.
- **Knowledge of the Environment**: algorithms based on local analysis [6] [23] (unknown or partially unknown environment) and algorithms based on global pathfinding techniques (completely known) [19] [27].
- **Number of Entities**: single navigation [19] [6] and multiple navigation algorithms [9] [4].

2.1.1 Environment Representation

1. **Graph Representation**

   The idea of graph representation is to reduce the free space to a subset of states that includes the start and final state. This subset is formed by the nodes and the edges of a graph. In this sense, the states are the nodes, and the possible actions are the edges (connections). The graph can be weighted or directed. In a weighted graph, each edge contains a measure of the cost that is needed for transition (actions) between the connected nodes (states). In a directed graph, the connections also have a possible direction. Thus, the cost can be different depending on the direction. Some examples of this representation are [14] [15] [27].

2. **Cell Decomposition**

   In this technique, the configuration space could be partitioned to *cells*. These structures can adopt different simple geometric shapes, such as squares, hexagons, and others. In this sense, the states are the cells, and the actions are the connections between neighbor cells. Thus, if the entity falls in some points into a cell, the entity occupies that state. For this reason, cells must be convex; any straight line segment connecting any two points inside the cell must lie entirely in that cell. In this kind of representation, it is possible to contain obstacles of different shapes into the cells implementing a previous approximation, Fig.2.2. Some examples of this representation are [16] [17] [28].

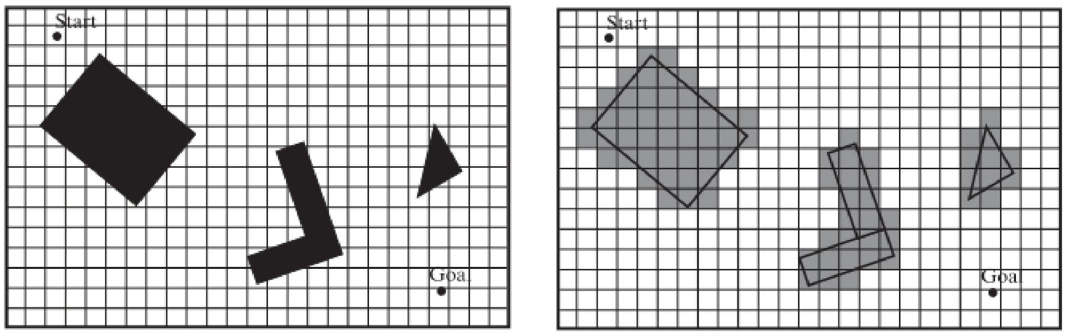


Figure 2.2: Obstacle Approximation Decomposition of Cells. Retrieved from [1].

3. **Road Maps**

A road map (a map that contains roads) consists of lines, curves, and points of intersections that occupy the configuration space according to specific criteria. In this sense, the states are represented by the intersections points, also known as vertices. Furthermore, the actions are the lines or curves which connect the vertices. The challenge is, therefore, to find a minimum number of roads that allow access to any free part of the environment, including the start and final state. Two common examples of road maps are Visibility Map and Voronoi Graph, Fig. 2.3. Some examples of this representation are [29] [18] [19].

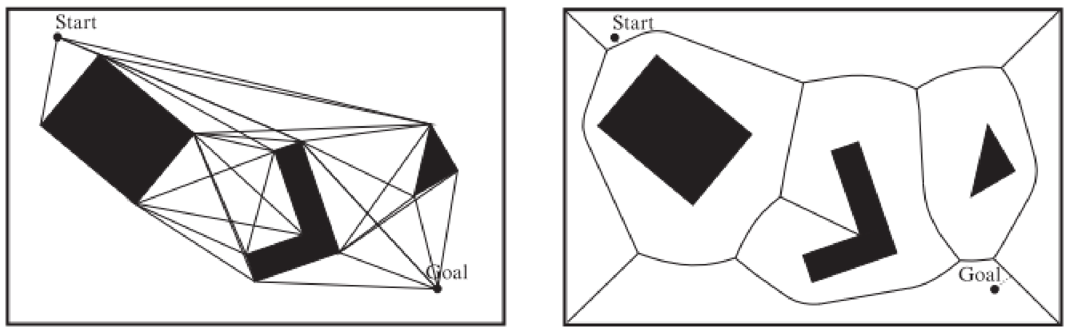


Figure 2.3: Visibility Map and Voronoi Graph. Retrieved from [1].

4. **Potential Field** This technique represents the environment with a *potential field*, which can be thought of as an imaginary height. In this representation, the final

state is in the bottom, and the height increases with the distance to the final state; the obstacles even have a higher height. In this sense, the path planning criteria can be explained as the movement of a ball that rolls downhill to the final state, Fig. 2.4. Some examples of this representation are [20] [21] [30].

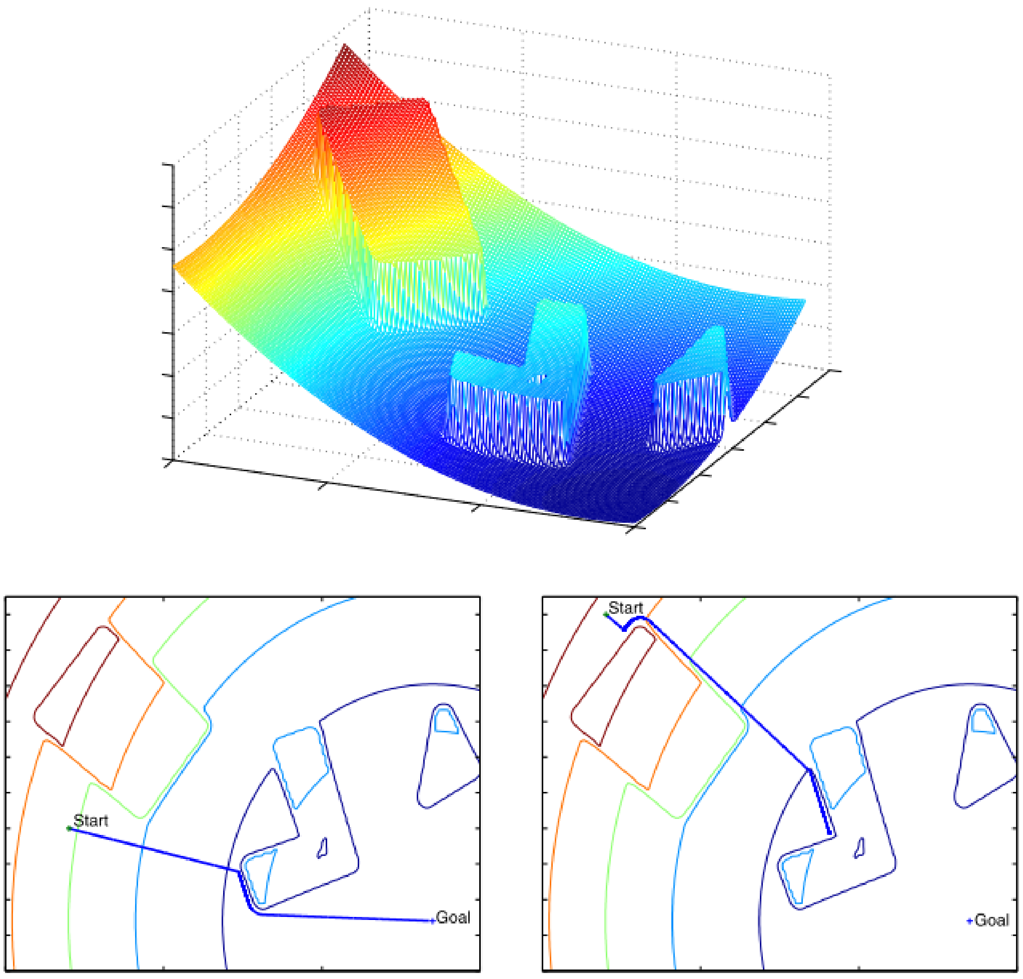


Figure 2.4: Potential Field Method Example. Retrieved from [1].

5. **Sampling-Based**

   The previous techniques require an explicit representation of the free space environment. These methods, therefore, tend to become time-consuming by increasing the dimension of the space. On another hand, in sampling-based methods, the free space is not known a prior; instead the algorithms must make a decision based on the local information. To this end, this method implements a collision detection criteria that implements random points, used to verify free spaces. These points and their connections allow producing a path between the start state and the final state. Additionally, to reduce processing time, the collision detection is checked only for obstacles that are close enough and present a potential hazard of collision. Some common examples of this method are Rapidly Exploring Random Tree (RTT) and Probabilistic Road Maps (PRM), Fig. 2.5. Some examples of this representation are [22] [23] [31].

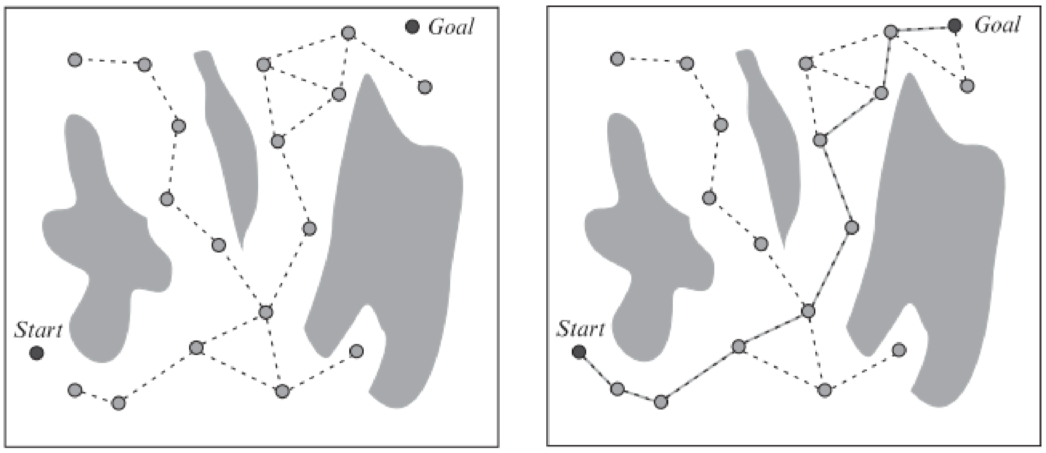


Figure 2.5: PRM Method. Learning phase, and path searching phase. Retrieved from [1].

2.2 Path Planning and Ant Colony Optimization

The origin of the Ant Colony algorithm arises in 1992, by Marco Dorigo in his doctoral dissertation [26]. His algorithm was based on the behavior of ants, called the Ant System (AS), for resolving the classical Travelling Salesman problem. Since then, it was researched the algorithm, especially in its applications for resolving combinatorial optimization problems and was called Ant Colony Optimization (ACO) [25] [32]. Later, this algorithm will be used in other applications like clustering [33], data mining [34], path planning, and others. From the last examples, the *Path Planning* has a particular research interest because it seems to have a connection with the real behavior of ants on nature. In a nutshell, the path planning and the stigmergy behavior seem to be similar problems, Section 1.2.

The present work proposes to solve the Path Planning problem using an Ant Colony Optimization algorithm. In this direction, it is essential to describe all the preliminary considerations and already known techniques. The first step to solve a path planning problem is to interpret the environment of work. In this way, this project implements an **undirected weighted graph** $G(N, E)$ where N is the set of nodes, and E is the set of edges. This representation helps us to scale the real problem to a bounded specific representation. Besides, we pretend to solve the **path planning problem one to one** where the aim is to find **the least cost path**. The cost of a path $(n_1, n_2, ..., n_k)$, therefore, can be denoted by Equation 2.1.

$$C = \sum_{i=1}^{k-1} w_{i,i+1} \tag{2.1}$$

where $w_{i,i+1}$ is the cost of each edge of the path. This cost can be represented by any kind of heuristic measure.

Besides, this specific path planning problem can appear in different conditions according to the initial node s, and the target node t. The variants of the problem can be:

- SP between a pair of nodes with s and t known.
- SP with s known and t unknown.
- SP with t known and s unknown.

After considering the preliminary information, it is necessary to establish a background of the state of the proposed research. In this way, the explanation below reflects the principal considerations that can be taken to drive the research of this specific problem. The following information is mostly based on some articles, retrieved from [25], and [35]. From this point we will refer to the algorithm of Path Planning based on Ant Colony Optimization as PPACO.

The PPACO algorithm includes several sub-problems that can be analyzed separately. This division allows researchers to choose and adjust each sub-problem of the algorithm according to their needs. Besides, these sub-problem categorization allows that the model can deal with different variants of the problem [25].

2.2.1 Initial Parameters

The PPACO algorithm uses the following parameters:

- A: population size of ants.
- α: a parameter that defines the influence of pheromones on the choice of the next node.
- β: a parameter that defines the influence of heuristic criteria $w_{i,j}$ on the choice of the next node.
- ρ: a parameter that defines the evaporation speed of the pheromones on the environment. It takes values between the interval $[0, 1]$.
- τ_0: the initial amount of pheromone on edges.
- τ_{max}, τ_{min}: the minimum and maximum acceptable amount of pheromone on edges.
- Q: a parameter that defines the number of pheromones that can be distributed on the edges of a specific path.

The population size of ants A alters the accuracy and the execution time of the solution. A larger population size of ants allows us to generate more feasible solutions, which can improve the global search ability. This better searching can help us to find more accurate solutions. However, if there are multiple optimal paths, the concentrations of pheromones of both paths will not vary too much. These concentrations produce more randomness in the choosing of next nodes, and hence more execution time. On the other hand, if A is smaller, some paths never will be analyzed. It is because the amount of pheromone will be completely volatilized, converging to optimal local solutions. In that way, this parameter allows us to negotiate between exploring ability and the algorithm time reduction. In some cases, we can also apply an adaptive parameter, Equation 2.2, retrieved from [35]. By this parameter, the population size is reduced sharply in the early stages, and less in the last stages.

$$A = \begin{cases} A_0 - A_{min} \cdot \frac{ep_i}{ep_{max}} & if \quad A \geq A_{min} \\ \\ A_{min} & otherwise \end{cases} \tag{2.2}$$

where A_0 is the initial population size, A_{min} is the minimum population size, ep_i is the current episode, ep_{max} is the maximum number of episodes.

The parameter α and β allow us to modify the choosing of the next node. Thus, you can give predominance to the pheromones over the heuristic criteria, as time passes. In other words, the values of α and β follow this criteria $\alpha > \beta$. Moreover, it could be important to maintain a short distance between these two parameters in some specific implementations. In other words, it could be necessary establish something like $|\alpha - \beta| < e$, where e is a small constant value. This avoids to give a lot of predominance to a pheromones over the heuristic criteria.

The parameter ρ states the speed of evaporation. On one hand, a quicker evaporation speed guarantees the exploration of new solutions. On the other hand, a slower speed allows focusing in the already found solutions, in other works it guarantees the convergence. The objective is finding an appropriate value which allow us to explore new solutions whereas the algorithm converges to a solution. Moreover, one of the possibilities is reducing the evaporation speed whereas the time passes. It is translate to implementation of the Equation 2.3, retrieved from [35].

$$\rho(ep_{i+1}) = \begin{cases} \phi \cdot \rho(ep_i) & if \quad \rho(ep_i) \geq \rho_{min} \\ \\ \rho_{min} & otherwise \end{cases} \tag{2.3}$$

where $\phi \in [0, 1]$ is a parameter that reduces the ρ value, ρ_{min} is the minimum value of ρ (evaporation factor), ep_i is the current episode, ep_{i+1} is the next episode. As a result, ρ has a large value in the early stage of the algorithm, and has a minimal value in the last stages.

The parameters which can adjust the contribution of pheromones are τ_0, τ_{min}, τ_{max}, and Q. The parameters τ_{min} and τ_{max} states lower and upper bounds of the number of pheromones per edge. These bounds can allow controlling the level of pheromones in synchronization with the evaporation speed. Moreover, these bounds can help to explore new solutions without losing the previous information. If we choose a small value of τ_{max}, this value can be reached quickly. In that way, it can fill with τ_{max} more of one feasible solution. However, it also can produce problems if the algorithm fills τ_{max} over bad paths. On another hand, in many cases, the τ_0 is defined like the minimum value of pheromones, $\tau_0 = \tau_{min}$. The Q parameter can work in two different ways. It can be the amount of contribution per edge of a path or the total contribution of the path. The use of these parameters can vary in dependency of the problem conditions. These parameters had been analyzed in a better way on the evaporation and reinforcement techniques analysis.

Lists with some information also can be taken depending on the path planning variants. These additional structures can be an occupation time list and the already visited list of nodes or edges. The occupation time list helps us to avoid that several ants being in the same node at the same moment. On the other hand, the visited list of nodes or edges allows us to avoid returning to back positions.

2.2.2 Finding Paths Methods

One of the principal features of a PPACO algorithm is the set of rules which will be used to establish the computational dynamics of the algorithm. For this reason, it is essential to establish the procedures for finding a path from the initial node to the end node. Two principal computational dynamics highlight in the researches: *Step by Step* and *Time List*.

1. **Step by Step Method**

   This method is an iterative sequential process where each ant finds a whole path one after another.

   On the one hand, The reinforcement of pheromones can occur in two ways:

   - The reinforcement occurs when an ant finds a complete path.
   - The reinforcement occurs when an ant moves from one node to another.

   On the other hand, the evaporation of pheromones can occur in different stages of the algorithm.

   - The evaporation occurs after each ant finds a whole path.
   - The evaporation occurs after all the ants find a complete path
   - The evaporation occurs in both of the previous cases.

   The reinforcement and evaporation techniques will be explained in a better way in the Sub-sections 2.2.4, 2.2.5, respectively.

2. **Time List Method**

   This method uses a time list for recording the time in which an ant reaches a given node. It is a concurrent method where the evaporation occurs with each transition of the next ant in the time list. The reinforcement of pheromones occurs after an ant reaches the target node. The value $\Delta\tau$ lays down in each edge of the path. The implementation of the time list allows that an ant that reaches the target node can lay down pheromone over trail earlier than the other ants. In that way, the ants, which do not reach the target node yet, will be influenced by the pheromones earlier. However, this method introduces some additional procedures which can provoke some redundancies. These redundancies can be compensated by the more effective reinforcement of short paths and better adjustment. The reinforcement and evaporation techniques will be explained in a better way in the Sub-sections 2.2.4, 2.2.5, respectively.

2.2.3 Movement Choice

The choice of next node uses a probability function, Equation 2.4, which states the probability of the ant k for moving from the node i to the node j.

$$p_{ij}^{k} = \frac{q_{ij}}{\sum_{l \in N} q_{il}} \tag{2.4}$$

where N is the set of all the next possible nodes j, and q_{ij} is the coefficient of the edge e_{ij}. After measuring the probability of each edge, we throw a dice or random number r_n uniformly distributed between $[0, 1]$. Using this random number, we choose an edge randomly but given more opportunity to the nodes with more probability.

- **Edge Coefficient** q_{ij}

  The choice of the next node is directly influenced by the edge coefficient of q_{ij}. This value considers the parameters α and β, which controls the influence of the pheromones τ_{ij} over the heuristic criteria η_{ij}. The heuristic criteria can be any additional measure that can be known previously or be measured during the execution of the algorithm. Another additional information, which can be useful, is the information about the already visited nodes or edges. According to this information, we can use a q_{ij} with different behavior. Some examples of the possible edge coefficients, retrieved from [25], are given by Equation 2.5.

$$q_{ij} = \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta} \tag{2.5a}$$

$$q_{ij} = \tau_{ij} \cdot \alpha + (1 - \alpha) \cdot \eta_{ij} \tag{2.5b}$$

$$q_{ij} = \begin{cases} \tau_{ij}^{\alpha} \cdot (1 + \beta) & if \quad is\_visited(n_j) = false \\ \tau_{ij}^{\alpha} & otherwise \end{cases} \tag{2.5c}$$

$$q_{ij} = \begin{cases} \tau_{ij}^{\alpha} \cdot (1 + \beta) & if \quad is\_visited(e_{ij}) = false \\ \tau_{ij}^{\alpha} & otherwise \end{cases} \tag{2.5d}$$

$$q_{ij} = \begin{cases} \tau_{ij}^{\alpha} \cdot (1+\beta)^2 \quad if \begin{cases} is\_visited(n_j) = false \\ is\_visited(e_{ij}) = false \end{cases} \\ \tau_{ij}^{\alpha} \cdot (1+\beta) \quad if \begin{cases} is\_visited(n_j) = true \\ is\_visited(e_{ij}) = false \end{cases} \\ \tau_{ij}^{\alpha} \cdot (1+\beta) \quad if \begin{cases} is\_visited(n_j) = false \\ is\_visited(e_{ij}) = true \end{cases} \\ \tau_{ij}^{\alpha} \qquad otherwise \end{cases} \tag{2.5e}$$

The edge coefficient method is crucial for the performance of the algorithm. In this way, according to the method, we can get different behaviors. If we can prioritize a quickly convergence (not necessarily the most optimal solution), we can choose the Equations 2.5a, 2.5b. If we can to explore features of the nodes and edges, we can choice the Equations 2.5c, 2.5d, 2.5e. Moreover, the last options can help us to avoid to return to back positions. However, it can take more execution time and memory consumption.

- **Pseudo-Random Generator and Initial Edge Coefficients q'_{ij}**

In complex graphs, the edge coefficient, q_{ij}, is not enough to choose the next node correctly. Because in the early stages of the algorithm, the environment is limited to fewer pheromones. In that way, the convergence can get stuck in the first feasible paths found. For solving this problem, we can use a ***pseudo-random generator*** which includes randomness to the edge selection, using an initial edge coefficient, q'_{ij} [25]. This kind of problem frequently happens when we have a high number of edges with uniform (equal) probabilities. In that way, the Equation 2.4 is replaced by the Equation 2.6.

$$p_{ij}^{k} = \frac{r_{ij}}{\sum_{l \in N} r_{il}} \tag{2.6}$$

where r_{ij} follows the Equation 4.2b.

$$r_{ij} = \begin{cases} q'_{ij} \quad if \quad ep_i < \delta \quad or \quad \tau_{ij} = 0 \\ q_{ij} \qquad otherwise \end{cases} \tag{2.7}$$

where ep_i is the current episode, and δ is a constant scalar number that can be fixed. There are several ways of writing the initial edge coefficient, q'_{ij}. In the Equation 2.8, we can see some examples, retrieved from [25]. In these examples, the parameter α is reused to limit the number of parameters. Thus, the α parameter is used as the power of the heuristic measure or its inverse.

$$q'_{ij} = \eta_{ij}^{\alpha} \tag{2.8a}$$

$$q'_{ij} = (\frac{1}{\eta_{ij}})^{\alpha} \tag{2.8b}$$

$$q'_{ij} = \begin{cases} \eta_{ij}^{\alpha} \cdot (1+\beta)^2 & if \begin{cases} is\_visited(n_j) = false \\ is\_visited(e_{ij}) = false \end{cases} \\ \eta_{ij}^{\alpha} \cdot (1+\beta) & if \begin{cases} is\_visited(n_j) = true \\ is\_visited(e_{ij}) = false \end{cases} \\ \eta_{ij}^{\alpha} \cdot (1+\beta) & if \begin{cases} is\_visited(n_j) = false \\ is\_visited(e_{ij}) = true \end{cases} \\ \eta_{ij}^{\alpha} & otherwise \end{cases} \tag{2.8c}$$

$$q'_{ij} = \begin{cases} (\frac{1}{\eta_{ij}})^{\alpha} \cdot (1+\beta)^2 & if \begin{cases} is\_visited(n_j) = false \\ is\_visited(e_{ij}) = false \end{cases} \\ (\frac{1}{\eta_{ij}})^{\alpha} \cdot (1+\beta) & if \begin{cases} is\_visited(n_j) = true \\ is\_visited(e_{ij}) = false \end{cases} \\ (\frac{1}{\eta_{ij}})^{\alpha} \cdot (1+\beta) & if \begin{cases} is\_visited(n_j) = false \\ is\_visited(e_{ij}) = true \end{cases} \\ (\frac{1}{\eta_{ij}})^{\alpha} & otherwise \end{cases} \tag{2.8d}$$

2.2.4 Pheromone Reinforcement

The reinforcement techniques are related with the *Finding Paths Methods*, Subsection 2.2.2. However, according to the conditions of the problem, we can combine different reinforcement techniques for finding new strategies. These techniques influence the decision made by the ants on the move stage. According to [25], the main reinforcement techniques are progressive, backward, overall, and selective.

- **Progressive**
  In this technique, the algorithm reinforces the pheromone trail when an ant moves from one node to another, during the search of a path.
- **Backward**
  In this technique, the reinforcement of the trail starts when an ant finds a whole path.

At this moment, the ant goes back from the end node to the initial node, reinforcing the edges of that path.

- **Overall**
  In this technique, the reinforcement of the trail starts when an ant finds a whole path. Then, the whole path is reinforced in one iteration.

- **Selective**
  In this technique, the reinforcement of the trail occurs only in a certain subset of the paths. This subset is defined under some conditions respecting to the best path. Some examples are the best path of a given iteration, the current best path, the paths with a cost into a pre-defined threshold, and others.

The *Progressive* technique is the most intuitive way of upgrading pheromones. However, this strategy causes the reinforcement of paths not yet found, which could later produce inappropriate effects as the reinforcement of not optimal paths. For this reason, a good method to counteract this problem is the reinforcement of the whole path. There are two options of the whole path reinforcement: the *Backward* and *Overall* techniques. In *Backward* technique, after finding a whole path, an ant goes back to the initial position, reinforcing the path one edge at a time. This way of taking the whole path allows us to take information about the accuracy of the path. This information can be used to reinforce the path proportionally to the quality of the solution. In that way, these features make this technique depend on the length of the path. However, the algorithm will spend much time if the path is too long (which happens in the first stages). To reduce this problem, we can, after reaching the end node, immediately reinforce the whole path instead of doing it in a series of steps. In other words, we can apply a *Overall* technique, which allows us to reduce the execution time and maintain some advantages such as the knowledge of the path quality. However, this technique introduces other problems related to transition periods of the ants. The transition periods are leveled, producing a more static reinforcement. In this way, the ants will be affected by pheromones later than if they would be using a time list. In other words, the pheromone updating will be static, and consequently, the probability of selection of given edges will be late. Finally, the *Selective* technique can be used when all the above techniques return no satisfactory results. This technique focuses on reinforcing a subset of paths. Some options of subsets are:

- Reinforce the $\frac{m}{2}$ best path in a given iteration.
- Reinforce the best path in a given iteration.
- Reinforce those paths that are better than the best path in a given iteration.
- Reinforce the best paths which have a cost into a pre-defined threshold.

This technique helps us to reduce the convergence time because the solutions are centered in a specific case. However, in some cases, this quick convergence could produce a high error range between the solution and the optimal solution. It is because the space of searching for the new path is reduced.

The above consideration illustrates the difficulties of adjusting a technique for pheromone reinforcement, which matches the conditions of a specific problem. In that way, it is necessary to perform an early stage where we can examine and experiment for finding the techniques which adjust in a better way to our problem. Moreover, it is important to use a variable reinforcing policy in initial iterations, which can reduce finding sub-optimal solutions. Another consideration is to control the transitions periods (not let them be too long), which can produce an unnecessary prolongation of the algorithm. The Table 2.1 summarize the behavior, the advantages and disadvantages of each reinforcement technique.

Technique	**Behavior**	**Advantages**	**Disadvantages**
Progressive	Reinforce on moving from one node to another node.	-Intuitive implementation. -Close to real behavior.	-Reinforcement of paths not yet found. -Reinforcement of not optimal paths.
Backward	Reinforce on a whole path and backpropagation of pheromones.	-Take information on the quality of the path solution. -Reinforce proportionally to the quality of the solution.	-Spent additional time on long paths.
Overall	Reinforce on a whole path in one iteration.	-Take information on the quality of the path solution. -Reinforce proportionally to the quality of the solution. -Transition periods are leveled.	-Spent additional time on long paths, but less than the Backward technique. -The effect of pheromone reinforcement appears less quickly.
Selective	Reinforce on a certain subset of the paths	-Reduce the convergence time. -Specific purpose technique, and adjustable to the problem conditions.	-Produce a high error range. -Reduce the searching space.

Table 2.1: Reinforcement Techniques

Updating Factor ($\Delta\tau$)

The control of deposited pheromones $\Delta\tau$ is also very essential because it directly affects the decisions made by the ants. If we choose a better $\Delta\tau$, the algorithm will converge quicker, and the probability of avoiding invalid results will be higher. The Equation 2.9 shows some possible ways of defining $\Delta\tau$ will be deposited on a found path.

$$\Delta\tau = const \tag{2.9a}$$

$$\Delta\tau = \frac{1}{c_p} \tag{2.9b}$$

$$\Delta\tau = \frac{Q}{c_p} \tag{2.9c}$$

$$\Delta\tau = \frac{c_{best}}{c_p} \tag{2.9d}$$

where $Q = \max_{e_{ij} \in E} c_{ij}$ is the maximum cost c_{ij} of the edges $e_{ij} \in E$, P denotes the path found by an ant, $c_p = \sum_{e_{ij} \in P} c_{ij}$ is its total cost, and c_{best} is the total cost of the best path found so far. In Equation 2.9d, the $\Delta\tau$ values is given by the ratio between the length of a path and the length of the shortest path found so far. In this sense, the pheromone value τ_{ij} is reinforced using Equation 2.10.

$$\tau_{ij}(k+1) = \tau_{ij}(k) + \Delta\tau \tag{2.10}$$

where k is the current state, $k+1$ is the new state.

2.2.5 Pheromone Evaporation

The evaporation technique is an intuitive process that consists in multiplying a value by the pheromone value of an edge. This behavior is represented by the Equation 2.11.

$$\tau_{ij}(k+1) = \tau_{ij}(k) \cdot (1-\rho) \tag{2.11}$$

where ρ is the evaporation factor, τ_{ij} is the pheromone value on the edge e_{ij}, and k is the current state.

Besides, it could add some bounds to the pheromone values. In that way, if the new pheromone value is lower than the minimum value, the new value is set to τ_{min}. Moreover, if the new pheromone value is greater than the maximum value, the new value is set to τ_{max}. The mechanism works in the same way like a *MAX-MIN* Ant System (*MMAS*) [36], [37].

$$\tau_{ij}(k+1) = \begin{cases} \tau_{min} & if \quad \tau_{ij}(k) \cdot (1-\rho) < \tau_{min} \\ \tau_{ij}(k) \cdot (1-\rho) & otherwise \end{cases} \tag{2.12}$$

where ρ is the evaporation parameter which states evaporation speed, τ_{ij} is the pheromone value on the edge e_{ij}, and k is the current state.

The above considerations shows that the evaporation has an easy implementation. However, the selection of frequency of evaporation implies a more difficult decision. We have to consider the actions of a single ant or the whole colony to choose the moment of pheromone evaporation. In that way, the following instances can be established [25]:

- Evaporation after the step of an ant.
- Evaporation after the step of all ants.
- Evaporation after finding a path by ant
- Evaporation after finding paths by all ants.
- Evaporation after a time change.
- Evaporation per time unit.

In this stage, it essential to consider an evaporation instance that adjusts to our problem condition and the method for finding paths, Section 2.2.2. In this way, the instances 1 can be applied with both methods of Section 2.2.2. Then, instances 2, 3, and 4 can be applied with the step by step method. Finally, instances 5 and 6 can be applied with a time list method. Moreover, it essential to adjust the parameter ρ because it establishes the speed of evaporation. If this value is too high, the algorithm can converge to non-optimal solutions. On another hand, if this value is too low, the algorithm will not converge.

2.2.6 Algorithm Ending

Previously it was mentioned that the ACO is a metaheuristic technique. In this way, it is not necessary to guarantee finding the optimal solution, but a good solution. This feature allows us to establish the ending of the algorithm under some factors and conditions. Thus, the algorithm can be stopped in three ways [25]: convergence, percentage, and time or iteration limit.

- The **convergence** occurs when all the ants follow the same path. In other words, this situation happens when there is a high concentration of pheromones on a path such that the next iterations do not generate significant changes.
- The choice of ants **percentage** is related to the convergence. When there is a considerable size population of ants, the convergence could take much time. Therefore, it can choose a percentage that defines how many ants must converge to stop the algorithm. This action will reduce the convergence time.
- The **time or iteration limit** is the most frequently used action because it establishes a boundary that promotes good experimentation. The algorithm stops when it reaches the threshold. This technique helps us to avoid unnecessary infinite repetition of the procedure.

In practice, the best way to proceed is to establish stability between the three methods. This stability is achieved by analyzing the credibility of the results and experimenting.

Chapter 3

State of the Art

In this chapter, we presented some main methods used for solving the path planning problem based on graphs or easily adapted to graphs. Figure 3.1 shows a conceptual map with a brief categorization of the main topics found in this research part. Furthermore, we did a brief description of each method. In some cases, we did a further explanation with some additional information. These techniques were used in the experimental stage for comparing the performance of our final proposal. These best-explained methods are Dijkstra, A* Search, and Genetic algorithms.

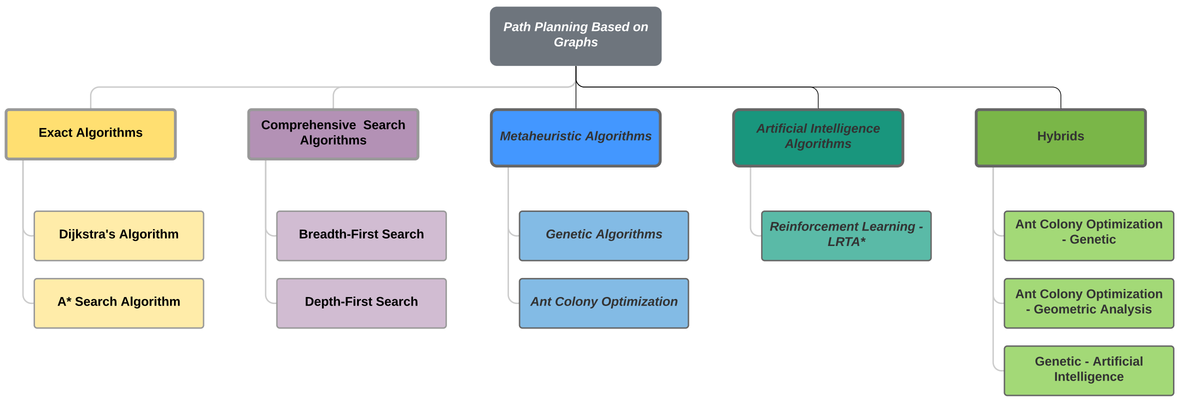

Path Planning Based on Graphs
Exact Algorithms
Comprehensive Search Algorithms
Metaheuristic Algorithms
Artificial Intelligence Algorithms
Hybrids
Dijkstra's Algorithm
A* Search Algorithm
Breadth-First Search
Depth-First Search
Genetic Algorithms
Ant Colony Optimization
Reinforcement Learning - LRTA*
Ant Colony Optimization - Genetic
Ant Colony Optimization - Geometric Analysis
Genetic - Artificial Intelligence

Figure 3.1: State of Art: Path Planning Algorithms based on Graphs

3.1 Exact or Complete Algorithms

The principal characteristic of these algorithms is the finding of the optimal solution of the path planning problem in a finite computational time.

3.1.1 Dijkstra Algorithm

Edsger W. Dijkstra presented this algorithm in 1959 [38]. The *Dijkstra Algorithm* works on undirected or directed graphs with nonnegative weights [39], [38]. It is essential to consider that this algorithm solves the single-source shortest path problem; in other words, it finds the shortest paths from a given node source to the rest of the nodes. As such, the algorithm explores the nodes, storing their accumulated distance, and marked some already analyzed nodes. In order to store the paths, this version establishes a structure, called *Tag*, that stores the information of an analyzed node. The following parts compose the *Tag*:

- Visited: a boolean value which indicates if a node already been visited totally. In other words, if the node has analyzed all these possible neighbor nodes.
- Goal distance $g(n)$: this field stores the accumulated distance of the node.
- Parent: this field indicates which is the parent node of the current node.

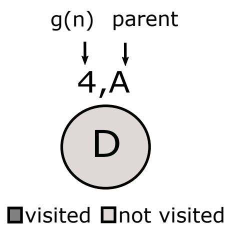


Figure 3.2: A* Tag Information

Procedure:

1. Initialize a list of x Tags with the information of each node. This list can be replaced by a priority queue structure for reducing the computational complexity. The information of the Tags is marked with the following information:
   - `visited = false`
   - `goal distance = Infinity`
   - `parent = null`

   The source node is marked with a `goal distance = 0` because the distance from the source to the source is 0.
2. Take the current node as the node with the least goal distance $g(n)$ and not visited. In the beginning, this node is the source node.
3. Take the neighbor nodes of the current node and iterate over them if **they are not marked as visited**. For each not visited neighbor, calculate the tentative distance $g_{k+1}(n')$ and assign it following the Equation 3.1.

$$g_{k+1}(n') = g(n) + d(n, n') \quad if \quad g(n) + d(n, n') < g_k(n') \tag{3.1}$$

where n' is the neighbor node, n is the current node, g_{k+1} is the tentative distance or the goal distance in step $k+1$, g is the goal distance, $d(n, n')$ is the distance between the current and neighbor node. Moreover, if the condition of Eq. 3.1 is met mark the parent node of neighbor node with the current node n.

4. Mark the current node as visited.

5. Repeat steps 2, 3, and 4 until all the nodes are marked as visited. Finally, the best-found distances are stored in the variable of the goal distance $g(n)$.

6. Reconstruct the route taking the final node and applying a backtracking process using the parent node information.

In Fig. 3.3, an example of the Dijkstra algorithm is illustrated. Besides, in Fig. 3.2, the components of the tag are shown for understanding the process.

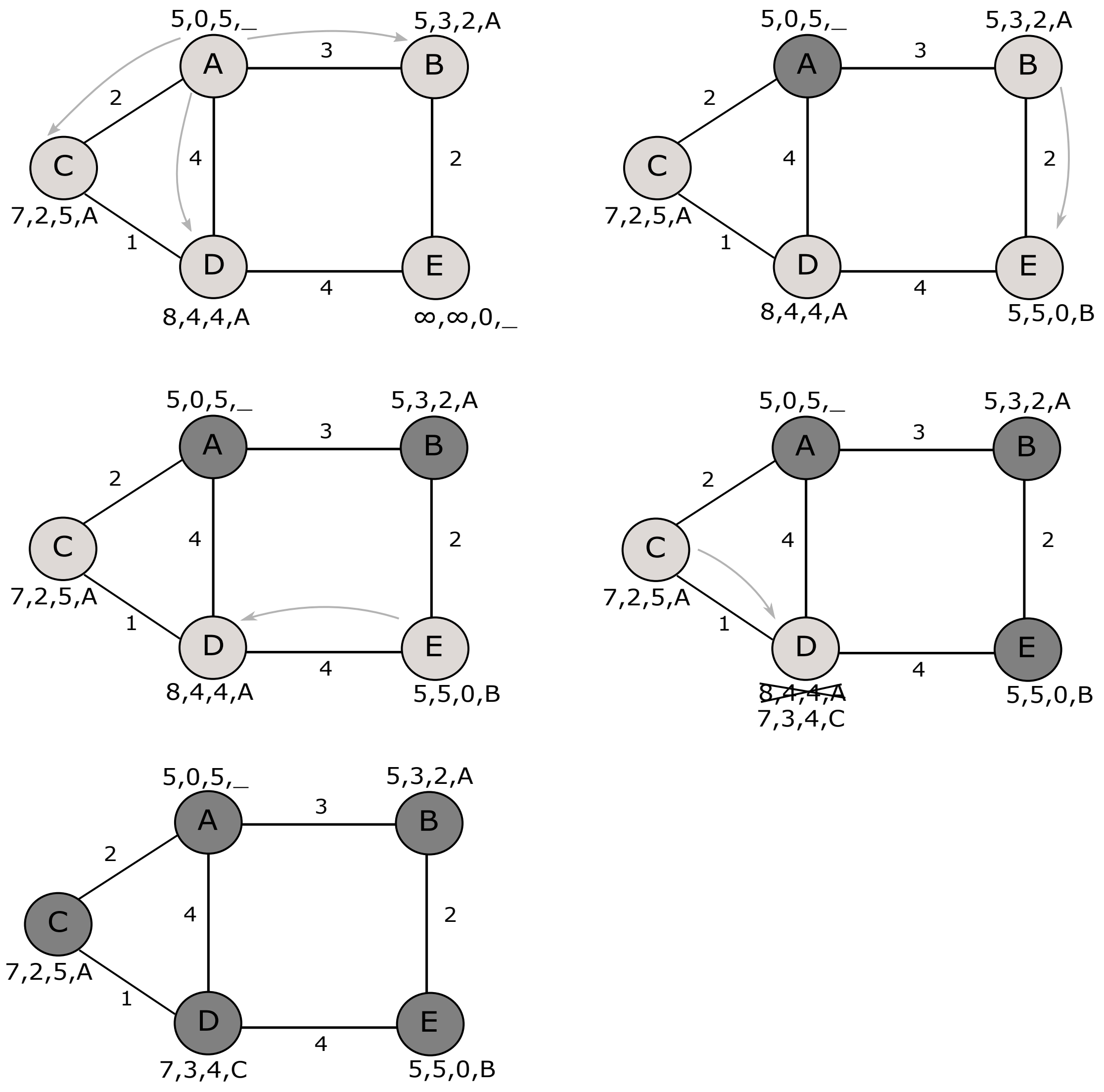


Figure 3.3: Dijkstra Algorithm Process

Pseudocode Dijkstra Algorithm

```
Input:  $G(N, E)$, $v_s$ (source node)
Output:  $P$ (path)
Main:
   tags_list = InitTagsList()
   while (!all_nodes_visited)
   |    $n$ = SelectNotVisitedLeastNode(tags_list)
   |    neighbors = SelectNotVisitedNeighbors($n$)
   |    for ($n'$ in neighbors)
   |    |    if ($g(n) + d(n, n') < g(n')$)
   |    |    |    $g(n') = g(n) + d(n, n')$
   |    |    |    $parent(n') = n$
   |    |    end
   |    end
   |    $visited(n) = true$
   end
   P = BackTracking(final_node)
```

3.1.2 A* Search Algorithm

The A* algorithm is a best-first search algorithm which is based on the *Dijkstra* algorithm. However, A* uses a preprocessing heuristic measure of each node. It evaluates nodes using $g(n)$, the cost to reach the node, and $h(n)$, the cost to get from node to the target [3].

$$f(n) = g(n) + h(n)$$

Therefore, the $f(n)$ value estimated cost of the cheapest solution through n. In that way, instead of take the least $g(n)$ as in *Dijkstra Algorithm*, the A* algorithm selects the least value of $f(n)$ in the tags list. In order to store the $f(n)$ and $h(n)$ values, this algorithm needs extra variables by each Tag. In that way, the following parts compose the *Tag*:

- Visited: a boolean value which indicates if a node already been visited totally. In other words, if the node has analyzed all these possible neighbor nodes.
- Global Goal $f(n)$: this field stores the global accumulation of the node.
- Local Goal $g(n)$: this field stores the accumulated distance of the node.
- Heuristic measure $h(n)$: this field stores the precalculated heuristic measure.
- Parent: this field indicates which is the parent node of the current node.

Besides, in order to find the optimal solution, the heuristic measure must be admissible and consistent [3]. A *admissible* heuristic is one that never overestimates the cost to reach the goal. On the other hand, a heuristic is *consistent* if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from

n is no higher than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

$$h(n) \leq c(n, a, n') + h(n')$$

Some admissible and consistent heuristic are straight-line distance, and proximity, Equation 4.1.

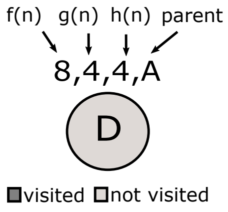


Figure 3.4: A* Tag Information

Procedure:

1. Initialize a list of x Tags with the information of each node. This list can be replaced by a priority queue structure for reducing the computational complexity. In the beginning, the information of the Tags is marked with the following information:

   - `visited = false`
   - `global goal = Infinity`
   - `local goal = Infinity`
   - `heuristic measure =` $h(n)$ `(precalculated)`
   - `parent = null`

   The source node is marked with a `local goal = 0` because the distance from the source to the source is 0, and a `global goal =` $h(n)$.

2. Take the current node as the node with the least $f(n)$ value and not visited. In the beginning, this node is the source node.

3. Take the neighbor nodes of the current node and iterate over them if **they are not marked as visited**. For each not visited neighbor, calculate the tentative distance $g_{k+1}(n')$ and assign it following this condition:

$$g_{k+1}(n') = g(n) + d(n, n') \quad if \quad g(n) + d(n, n') < g_k(n') \tag{3.2}$$

   where n' is the neighbor node, n is the current node, g_{k+1} is the tentative distance, g is the local goal, $d(n, n')$ is the distance between the current and neighbor node. Moreover, if the condition of Eq. 3.2 is met mark the parent node of the neighbor node with the current node n and set the global goal to:

$$f(n') = g(n') + h(n')$$

4. Mark the current node as visited.
5. Repeat steps 2, 3, and 4 until all the nodes are marked as visited. Finally, the best-found distances are stored in the variable of the local goal $g(n)$.
6. Reconstruct the route taking the final node and applying a backtracking process using the parent node information.

In Fig. 3.5, an example of the A* Search algorithm is illustrated. Besides, in Fig. 3.4, the components of the tag are shown for understanding the process.

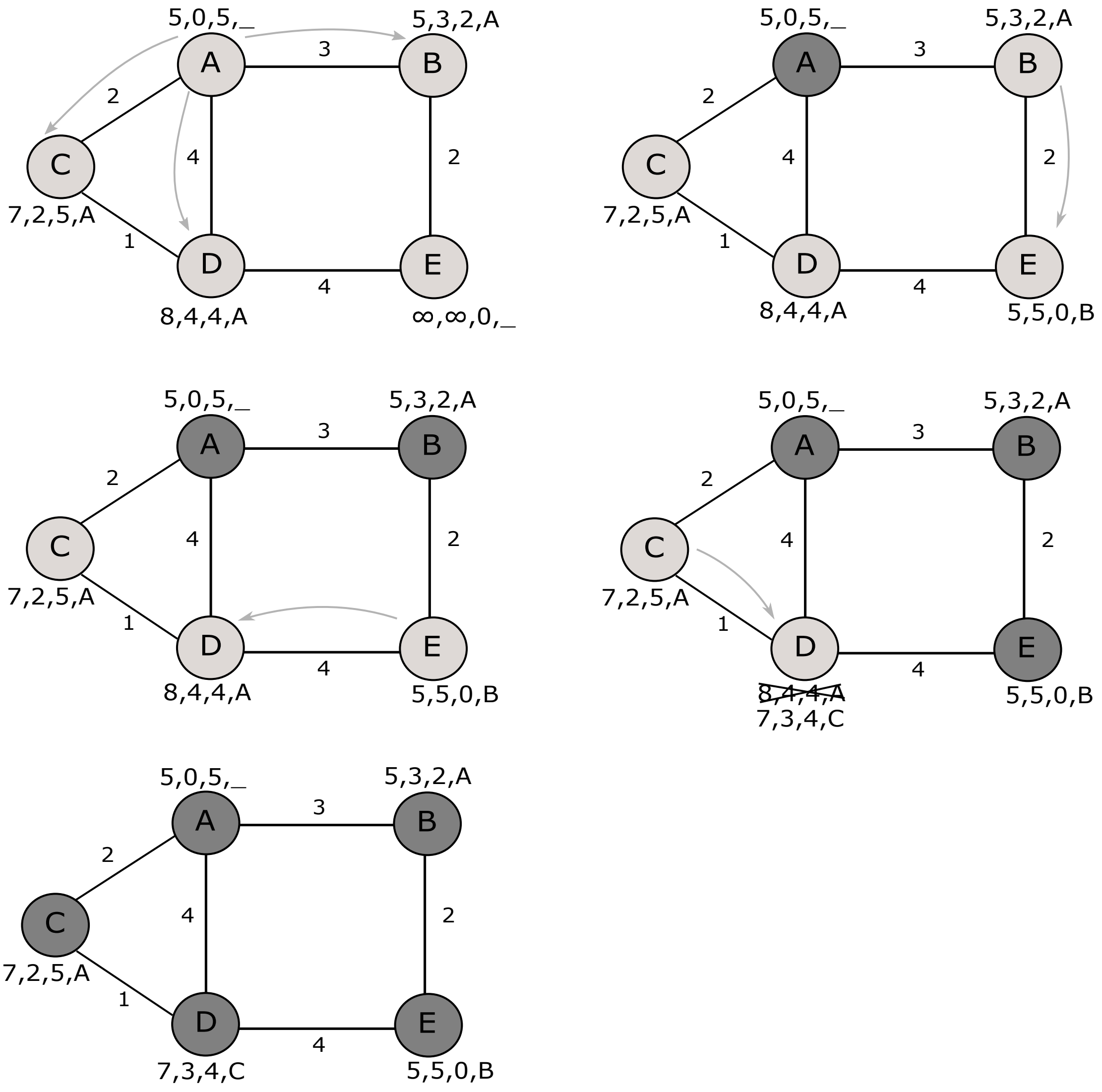


Figure 3.5: A* Search Algorithm Process

Pseudocode A* Search Algorithm

```
Input:  $G(N,E)$, $v_s$ (source node), $h(n)$ (precalculated heuristic)
Output:  $P$ (path)
Main:
   tags_list = InitTagsList($h(n)$)
   while (!all_nodes_visited)
   |    $n$ = SelectNotVisitedLeastNode(tags_list)
   |    neighbors = SelectNotVisitedNeighbors($n$)
   |    for ($n'$ in neighbors)
   |    |    if ( $g(n) + d(n,n') < g(n')$ )
   |    |    |    $dt(n') = g(n) + d(n,n')$
   |    |    |    $parent(n') = n$
   |    |    |    $f(n') = g(n') + h(n')$
   |    |    end
   |    end
   |    $visited(n) = true$
   end
   P = BackTracking(final_node)
```

3.2 Comprehensive Search Algorithms

The principal characteristic of these algorithms is a exhaustive search of the graph for finding the solution of the path planning problem.

3.2.1 Breadth-First Search

The following explanation was based on [1]. The *Breadth-First Search* belongs to a class of uninformed graph search algorithms. It first explores the nodes that are close to the starting node. In other words, this algorithm prefers to analyze all the nodes that can be accessed in k steps, instead of $k+1$ steps.

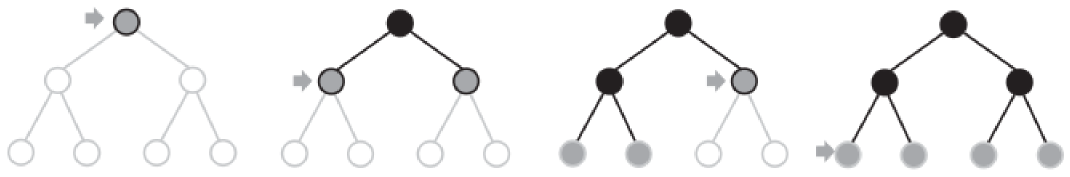


Figure 3.6: Breadth-First Search Algorithm Process. Retrieved from [1]

The algorithm implements an open and closed list. In Fig. 3.6, the arrow marks the current explored node, the gray nodes are in the open list, and the black nodes are in the closed list. In the open list Q, the nodes are sorted using a FIFO method (First-In First-Out). The newly opened nodes are added to the end of the list Q, and the nodes for continuing the search are taken from the beginning of the list Q. The algorithm is *complete* because it finds a solution if it exists. Nevertheless, the technique generally is high memory and time consuming; both features increase exponentially as graph branching progresses.

3.2.2 Depth-First Search

The following explanation was based on [1]. The *Depth-First Search* is a non-informed graph search algorithm where the nodes are extended by depth. First, it explores the nodes that are farthest away from the starting node. Thus, the search continues in depth until the current node has no further successors. The search is then continued with the next deepest node whose successors have not been explored yet, as shown in Fig. 3.7.

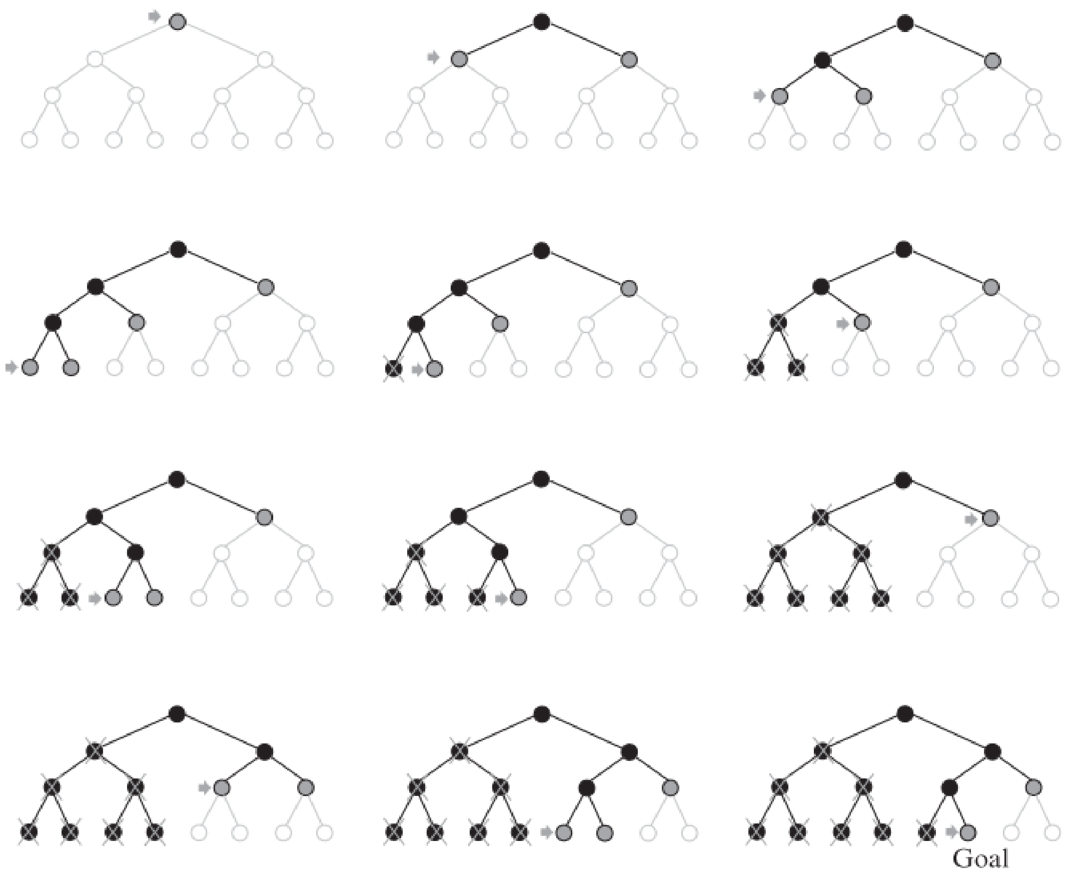


Figure 3.7: Depth-First Search Algorithm Process. Retrieved from [1]

The algorithm also implements an open list of Q, which is sorted by the LIFO method (Last-In First-Out). Thus, the newly opened nodes are added to the beginning of the list Q. Besides, the nodes for continuing the search are also taken from the beginning of the list Q. The algorithm is not *complete*; even it can get stuck if it falls in a cycle. For this reason, it can be limited to a certain depth in some cases. This algorithm has a high time consuming, but a low memory usage. This advantage is because when some nodes and all their successor are completely explored, these nodes no longer need to be stored in the memory.

3.3 Metaheuristics Algorithms

The principal characteristic of these algorithms is the finding of feasible solutions to the path planning problem, not necessarily the optimal solution, just a good solution.

3.3.1 Genetic Algorithm

A genetic algorithm seeks to solve the path planning problem using individuals that evolve, improving its features over time. The algorithm implements some particular functions: Random Population Generation, Fitness, Mutation, and Crossover. These functions help us to guarantee to find a solution. For purposes of comparison, the present work implements an already implemented version [2], [5] with some changes for adjusting with our environment representation.

Procedure:

1. Define the principal parameters of the algorithm:
   - ϕ_c: stop condition, percentage of same individuals.
   - n_i: number of individuals or population size.
   - ρ_s: probability of survivals.
   - ρ_m: probability of mutations.
   - ω_e: coefficient of obstacle edges.
   - ω_n: coefficient of obstacle nodes.
2. Create an initial random population of n_i individuals
3. Measure the fitness of each individual of the current population.
4. Take the survivors, individuals with the best fitness, according to ρ_s.
5. Apply a crossover between survivors, which generates new individuals. In this step, the population size must not exceed the n_i individuals.
6. Apply the *Genetic Operators* to the new individual: Mutation, Node Repair, Line Repair, Deletion, and Improve.
7. Repeat steps 3, 4, 5, 6, until reach a percentage of ϕ_c of the same individuals.

Individual

In this algorithm, an individual is encoded as a path, which is a sequence of nodes. This sequence starts with the initial node and ends with the final node. A path can be either feasible (collision-free) or infeasible. A path is infeasible if its intermediate nodes fall on any of the states with obstacles.

Random Initial Population

The algorithm creates a random initial population generating some random nodes (not necessarily linked) between the initial and final positions. Then, a procedure links all the nodes of this pseudo route implementing a process that considers the best proximity measure 4.1. This process is similar to the Random Walk 2, Section 4.2, which will be explained later.

Fitness Function

The algorithm uses fitness function over the individuals of a generation. This function considers the nodes and edges, which can be obstacles in infeasible paths. Therefore, if the road is infeasible, the fitness function value will be few, Equation 3.3a.

$$F_{cost} = \sum_{i=1}^{N}(d_i + \beta_i * \omega_e) + len(O_n) * \omega_n \tag{3.3a}$$

$$\beta_i = \begin{cases} d_i \quad if \quad i \in O_e \\ \\ 0 \quad otherwise \end{cases} \tag{3.3b}$$

where N is the number of edges, d_i is the distance of an edge, O_n is the set of obstacle nodes in the current path, O_e is the set of obstacle edges in the current path, ω_e is the cost factor of obstacle edges, and ω_n is the cost factor of obstacle nodes.

Crossover

The crossover procedure takes two paths of the best individuals (survivors) and joins them following a pre-defined criteria, Fig. 3.8. First, the algorithm fragments both paths in a random middle point. Then, a process selects the first fragment of path one and the second fragment of path two and tries to join them. Commonly these parts are unlinked, so the algorithm connects them using the best proximity measure for select the next nodes.

Genetic Operators

- **Mutation:** the mutation operator selects a node of the path randomly and changes it by another random node. If the new path is unlinked, their nodes are joined following the best proximity criteria.
- **Node repair:** the node repair operator works only in infeasible paths. This operator selects an obstacle node of the path randomly, and change it by a feasible node if it is possible, Fig. 3.8.
- **Line repair:** the node repair operator works only in infeasible paths with edges obstacles. This operator selects an edge obstacle randomly and changes it by a connection with a feasible node if it is possible, Fig. 3.8.
- **Deletion:** the deletion operator works in both feasible and infeasible paths. This operator selects a node of the path randomly. If the deletion of this node increases the fitness, the process deletes the node.

- **Improve:** the improve operator works only with feasible paths. This operator selects a node randomly, and change it with another feasible close node. If the fitness of this new path increases, the change is conserved.

Some of these operators can not work with our environment representation (graph representation) because each node has at most four possible connections. Therefore, if we tried to replace some of the nodes of the path, we can get unlinked routes. In the present work, we compare our ACO version with a Genetic version, which only uses a Mutation and Improve operator. These operators fit with our graph representation and are enough to generate good results.

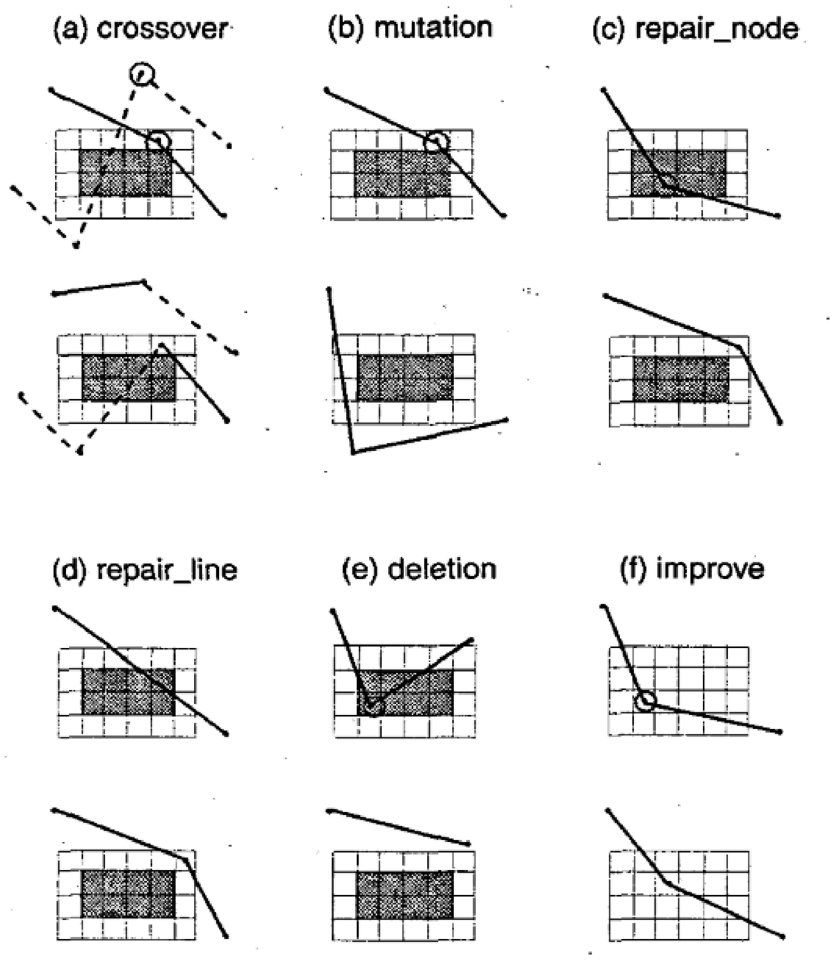


Figure 3.8: Genetic Operators. Retrieved from [2]

Pseudocode Genetic Algorithm

```
Input:  $\phi_c$, $n_i$, $\rho_s$, $\rho_m$, $\omega_e$, $\omega_n$
Output:  $P$ (path)
Main:
   population = CreateRandomPopulation($n_i$,)
   while (!stop_condition($\phi_c$))
   |    MeasureFitness(population, $\omega_e$, $\omega_n$)
   |    survivors = GetBestIndividuals(population,$\rho_s$)
   |    population = survivors
   |    while(population.len < $n_i$)
   |    |    parent1 = GetRandomParent(survivors)
   |    |    parent2 = GetRandomParent(survivors)
   |    |    child = Crossover(parent1, parent2)
   |    |    Mutation(child, $\rho_m$)
   |    |    NodeRepair(child)
   |    |    LineRepair(child)
   |    |    Deletion(child)
   |    |    Improve(child)
   |    |    population.Add(child)
   |    end
   end
   P = SelectBestIndividual(population)
```

3.4 Artificial Intelligence Algorithms

3.4.1 Reinforcement Learning - LRTA*

The following explanation is based on the following articles [40] [41] and [3]. The Learning Real-Time A* algorithm, also known as LRTA*, was introduced by Korf in 1990 [40]. It is the first and best known learning real-time heuristic search algorithm or online local search algorithm. The aim focuses on the balance between acting and improving the heuristic estimations. As well as the A* algorithm, this method implements an estimated heuristic $h(n)$, which denotes the cost to get from the node to the target and the precisely known value $g(n)$, which denotes the cost to reach the node. The algorithm then goes through the states with lowest $f(n) = g(n) + h(n)$, just like in A* search algorithm. Nevertheless, the LRTA* adds another process that allows it to look ahead, updating the estimated heuristic values.

The algorithm consists in to store a “current best estimate” $H(n)$ of the cost to reach the goal from each state that has been visited. $H(n)$ starts out being just the heuristic estimate $h(n)$ and is updated as the algorithm gains experience in the graph space. The algorithm therefore is guide by the nodes with the least $f(n) = g(n)+H(n)$, instead of the previous $f(n)$. In this way, the $H(n)$ is updated to $C(n, a, n')+H(n')$ only if the minimum calculated value $f(n')$ is greater than the $H(n)$. Thus, in the worst case, this algorithm explores the environment in $O(n^2)$. Figure 3.9 illustrates the algorithm process in a better way.

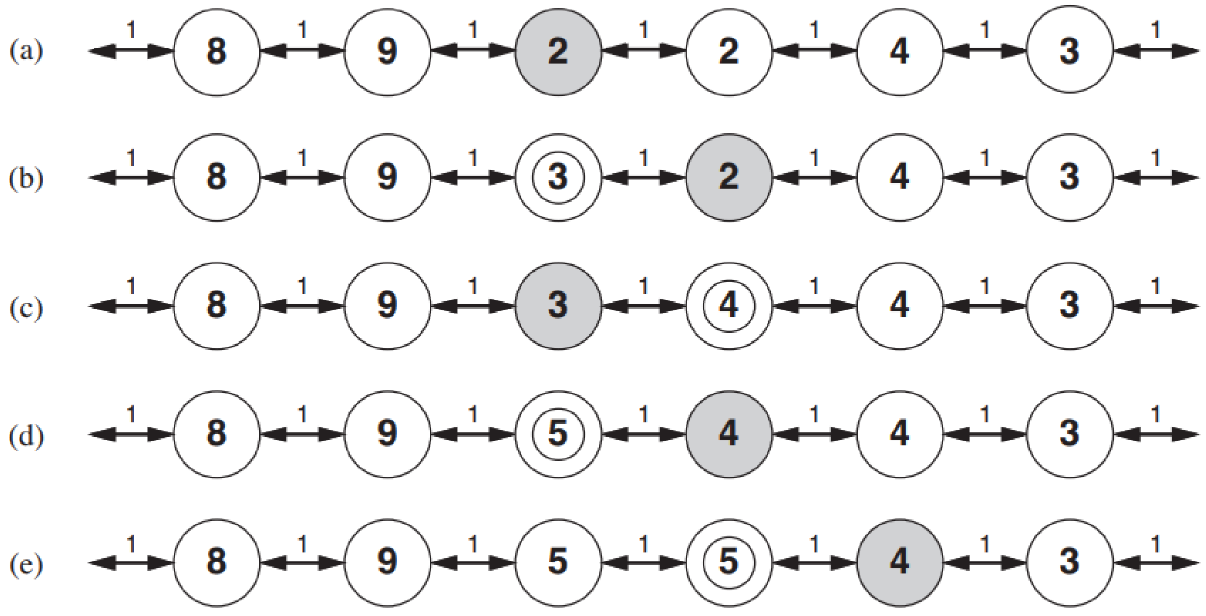


Figure 3.9: LRTA* Search Algorithm Process. Retrieved from [3]

Fig. 3.9 shows a one-dimensional graph space where each node is labeled with $H(n)$, and the edges are labeled with the step cost. The idea is moving from left to right. In this sense, in Fig. 3.9(a), the algorithm seems to be stuck in a flat local minimum at the shaded state. In the example, there are two actions, with estimated costs of $1 + 9$ and $1 + 2$, so it seems best to move right. Now, it is clear that the cost estimate of 2 for the shaded state was overly optimistic, so the $H(n)$ should be updated to 3 following $H(n) = C(n, a, n') + H(n') = 1 + 2$, Fig. 3.9(b). The updated $H(n)$ are circled, and the current state is colored with a gray color in Fig. 3.9. By continuing this process, the algorithm will move back and forth twice more, updating $H(n)$ each time and "flattening out" the local minimum until it escapes to the right.

Chapter 4

Methodology

This chapter presents the procedures implemented to reach the objectives, together with a justification for their use. In that way, this chapter starts with an explanation of the chronological phases for problem resolution. Then, we present the model proposal, which includes the ACO equations and parameters, the global and local interactions, and the features implemented in the three-dimensional simulation. Finally, we describe the data sets and the methods for analyzing the results.

4.1 Phases of Problem Solving

The following diagram shows the work-flow which had been taken to perform this project.

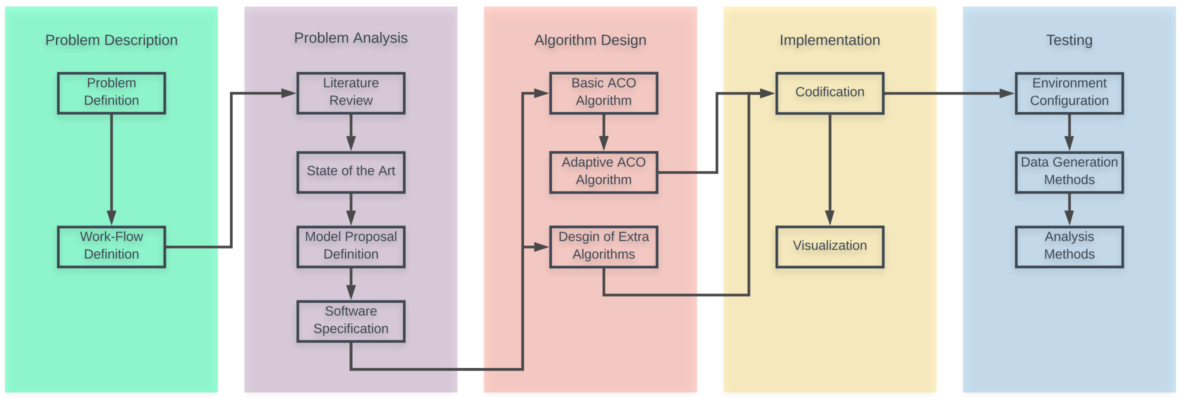


Figure 4.1: Phases of Problem Solving.

4.1.1 Description of the Problem

This phase acts as the base of the project development because it presents the main topic, its possible problems, and how to deal with it. In this way, in the first steps, we raise planted the essential meaning of path planning and its general features. Then, we detected some possible problems and decided what problem to address. Later, we defined a brief model proposal to deal with this problem. To this end, we also defined a workflow to design and develop the model proposal, Fig. 4.1. In this case, we decide to use an incremental technique to design and codify the algorithms progressively. This style helped to detect possible bugs easily. The Chapters 1 and 4 consolidate the steps mentioned in this phase.

4.1.2 Analysis of the Problem

This phase consists of the background and fundamental knowledge to understand in a better way the present project. Chapter 2 presents this information in a systematic and ordered way where we started analyzing the path planning problem, and then we deal with the Ant Colony Optimization. In this sense, we focused on the path planning problem one to one based on graphs. Besides, this information was useful to describe a brief *State of the Art*, Fig. 3.1. According to this background, we defined a more conscious model proposal, where we describe the local and global interactions, as well as the simulation features, its extensions, and prerequisites. Finally, it was determined the software specifications for the simulation that are Unity 3D for visualization and C# for algorithm codification. The Chapters 2 and 4 consolidate the information mentioned in this phase.

4.1.3 Algorithm Design

In this phase, it was designed the different alternatives of the model proposal. These algorithms are a basic ACO algorithm, the ACO variants which considers local and global interactions, the Dijkstra algorithm, the A* algorithm, and the Genetic algorithm. These algorithms are explained in the Chapter 3 and Section 4.2.

4.1.4 Implementation

This phase consists of the codification in a specific programming language and the visualization in a specific framework. For the visualization, we took the Unity 3D framework because it has already implemented physical dynamics that help in the creation and moving of objects. For the algorithm, we decided to use the C# programming language for avoiding compatibility errors with the visualization. It is because Unity 3D uses C# as its principal programming language. However, some initial prototypes were made in Python because it is a language that allows detecting errors quickly.

4.1.5 Testing

This phase focuses on measuring and analyzing the performance of the ACO algorithm. It is achieved by comparing the basic results with the addition of local and global interactions, and with other implementations. For this reason, it was necessary to establish some criteria for each variant can have similar initial conditions before the analysis. Then, we have to establish how to generate the data for the experimentation. Finally, the results was evaluated using some techniques for getting information about its performance. All these steps are detailed in the Section 4.3.

4.2 Model Proposal

The proposal model pretends to deal with a path planning problem one to one. To this end, we selected a standard environment representation of graphs that also can quickly adapt to other representations such as cell decomposition or road maps. In this sense, this project proposes a computational model based on global and local interactions and some principles of the literature review. In order to design and develop this model, we use an incremental style. In this way, the first step was to define the environment where the algorithm will take place. Then, we developed a basic version of the ACO algorithm, which is used as a base algorithm. Later, in order to produce a better performance, we established some global and local interactions which generate different versions of the ACO algorithm. Finally, we consider the best ACO variants, to develop a three-dimensional graphical simulation. In this part, additionally, we add another heuristic, which confers a notion of height to the simulation.

4.2.1 General Mesh Environment

The mesh environment was represented by an undirected weighted graph $G(N, E)$, where N is the set of nodes n_i, and E is the set of edges e_{ij}. We take the nodes of this graph from Wavefront .obj files, which allows simulating almost real environments. These environments have several characteristics which list below:

- The nodes of the graph connect with their neighbors in four directions: north, south, west, and east.
- The nodes stores information about their neighbors to faster access.
- The nodes are uniformly distributed in two-dimensional scenarios and with intrusions and extrusions in three-dimensional scenarios, Fig. 4.14.
- The nodes can store the value $\frac{1}{d_{it}}$, the list H_i of proximity values ν_{ij}, and the list S_i of slopes values s_{ij}.
- The edges can store several heuristic measures: distance d_{ij}, pheromone τ_{ij}, and necrophoresis value κ_{ij}.
- The ants can only move between nodes, through the four possible edges.
- A completed path is a set of node $C = n_0, n_1, ..., n_i, ..., n_k$.

Proximity

In this project, we proposed the use of another heuristic instead of the distance, know as **proximity**. This measure indicates how distant is the node n_j to the target node n_t, regardless of the obstacles present, Equation. 4.1.

$$\nu(n_i, n_j) = d_{it} \cdot \frac{1}{d_{jt}} \tag{4.1}$$

where n_i is the current node, n_j is the neighbor node, n_t is the target node, d_{it} is the relative euclidean distance from the current node n_i to the target node n_t, and d_{jt} is the relative euclidean distance from the current node n_j to the target node n_t. In this way, if the value of $\nu(n_i, n_j)$ is high, the node n_j is close to the target node.

An obvious thing could be that we could use the value $\frac{1}{d_{jt}}$ over each node, and we can have the same behavior. However, according to the position of the node, the value $\frac{1}{d_{jt}}$ can vary on a large scale. For this reason, the normalization step of multiply by d_{it} is essential to reduce the variability. This technique allows reducing the values to an approximated interval between $(0, 3]$.

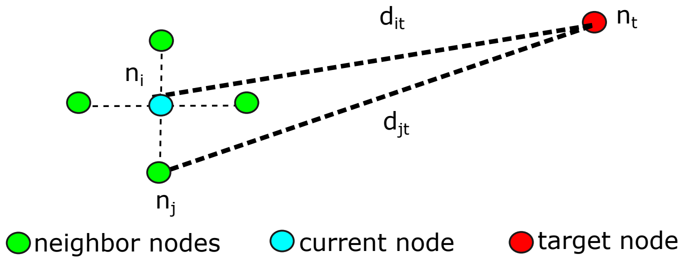


Figure 4.2: Proximity

One limitation of this heuristic is that it must be calculated several times. Each node have a list of four proximity values H_i. This effect is produced because $\nu(n_i, n_j) \neq \nu(n_j, n_i)$, Fig. 4.2. Despite this limitation, these values can be stored in preprocessing time.

The fundamental to propose this heuristic is that the use of the distance d_{ij} between nodes is not useful enough for finding a feasible path using the ACO algorithm. The environment configuration provokes that the d_{ij} values are almost the same in each step. Therefore, if we use only the distance in the early stages of the ACO algorithm, we can get similar probabilities. This fact can provoke a high randomness behavior that can increment the convergence time or even not produce it. In this sense, the proximity measure can achieve behavior that disgorges in a convergence. Besides, another feature of this heuristic is that it can be applied to an environment with more connected nodes. For example, a node that is connected with eight nodes.

4.2.2 Basic ACO algorithm - *ACOv0*

This algorithm implements the fundamental principles of the ACO algorithm. In this sense, the algorithm is only able to work in an environment with few obstacles or no one. This algorithm served as a base to include new interactions that allow a more adaptive behavior. The parameters used in this algorithm are the following:

1. *Main Parameters*

    - ϕ_c: convergence percentage. This parameter defines the stop conditions of ants. In other words, if ϕ_c percentage of the ants are following a route, then the algorithm stops.

- A: number of ants per episode.
- α: a parameter that defines the influence of pheromones on the choice of the next node.
- β: a parameter that defines the influence of another heuristic measure on the choice of the next node.
- τ_0: a parameter that defines the initial amount of pheromones per edge.
- ρ: a parameter that defines the evaporation speed of the pheromones on the environment. It takes values between the interval $[0, 1]$. It could be strong ρ_{strong} or light ρ_{light}, depending on reinforcement and evaporation technique.
- δ: a parameter that defines the initial iterations of the pseudo-random generator.

Furthermore, in this basic version, the algorithm **checks previous positions** to guarantee the convergence. This interaction avoids that an ant can visit previous or already visited positions. However, this condition provokes that the ants get stuck in some cases.

Procedure:

1. Initialize the environment, parameters, and colony of ants.
2. One ant of the colony tries to find a route to the final node using a *Movement Choice* dynamic, Eq. 4.2.
   (a) By **checking previous positions**, the ant could get stuck in some situations, and not find a route.
3. If the ant finds a route, a reinforcement over the environment takes place, using Eq. 4.3.
4. Repeat steps 2 and 3 until complete an episode. An episode occurs when all the ants have tried to find a route.
5. After each episode, an evaporation step occurs if at least one ant of the colony found a route or if the ants get stuck several times with the same last node. In this step, the Equation 4.5 is used.
6. The algorithm continues until the most ants can get the same route. The converge percentage of ϕ_c defines this stop condition.

The principal points of this ACO version are Movement Choice, the Reinforcement Pheromone, and the Evaporation Pheromone. The explanation of each one of these methods, implemented in this basic version, is below.

Movement Choice

Let the current node n_i; there are a previous nodes checking condition and a probability choice for selecting the next node.

1. *Checking Previous Nodes Interaction* Let N the set of all the next possible nodes n_j, where $\#N \in [2,4]$ due to the connections of the graph. If n_j is already visited $\Rightarrow$ $N = N - \{j\}$.

2. *Probability Choice.* This simulation proposed an algorithm which can have a quick response. In that way, we decided to work with the classical implementation of *Edge Coefficient*, which uses powers with pre-defined parameters, Equation 2.5a. Moreover, using the classical implementation, it is easy to add the additional heuristic criteria: proximity and slope equation. Furthermore, for the initial randomness, it is necessary to implement a **pseudo-random generator** r_{ij} in the early stages of the algorithm to guarantee an accurate convergence.

In this sense, let N, the set of the possible next nodes n_j. The probability of each node n_j is measured with the Equation 4.2.

$$p_{ij}^k = \frac{r_{ij}}{\sum_{l \in N} r_{il}} \tag{4.2a}$$

where r_{ij} is:

$$r_{ij} = \begin{cases} q'_{ij} \quad if \quad episode < \delta \quad or \quad \tau_{ij} = \tau_0 \\ \\ q_{ij} \qquad\qquad otherwise \end{cases} \tag{4.2b}$$

, and

$$q_{ij} = \tau_{ij}^{\alpha} \cdot \nu_{ij}^{\beta} \tag{4.2c}$$

$$q'_{ij} = \nu_{ij}^{\alpha} \cdot \frac{1}{d_{ij}}^{\beta} \tag{4.2d}$$

where p_{ij}^k is the probability that the ant k moves from node n_i to node n_j. Then, we get a random number r_n uniformly distributed between $[0,1]$, where 1 is the accumulative probability of all the next possible nodes. Finally, we accumulate each probability of node n_j until the sum is greater or equal to r_n. We stop and selected that node n_j. In that way, the nodes with more probability have more opportunity of being selected.

Pheromone Reinforcement

In this work, the **overall** technique for pheromone reinforcement was chosen by the large size of the environments. In the experimental stage, this complexity can provoke problems in the response of ants to the pheromones and time of convergence. The chosen technique can deal with this kind of problem, leveling the transition periods of the ants. Moreover, this technique allow to take information about the quality of the solution in each step. This information will be useful for defining the updating factor $\Delta\tau$, Equation. 4.4.

In this sense, let the set P, which stores the nodes of a specific path. The pheromone reinforcement equation is represented in Equation 4.3.

$$\tau_{ij}(k+1) = \tau_{ij}(k) + \Delta\tau \tag{4.3}$$

where k is the current state, and

$$\Delta\tau = \frac{c_{best}}{c_p} \tag{4.4}$$

where $c_p = \sum\limits_{e_{ij} \in P} c_{ij}$ is its total cost of path P, c_{best} is the total cost of the best path found so far.

The **overall** technique evaluates the Equation 4.3 each time that an ant finds a path. The update process is performed only in the pheromones values, which correspond to the current route.

Pheromone Evaporation

The evaporation technique uses the ρ parameter to control the number of pheromones per edge. Moreover, we decided to use intense evaporation after each episode.

In this sense, let the pheromone value τ_{ij}, the evaporation equation is given by the equation:

$$\tau_{ij}(k+1) = \tau_{ij}(k) \cdot (1-\rho) \tag{4.5}$$

where k is the current state of τ_{ij}.

Pseudocode Basic ACO Algorithm

```
Input: $G(N,E)$, $\phi_c$, $A$, $\alpha$, $\beta$, $\tau_0$, $\rho$, $\delta$
Output: $P$ (path)
Main:
   colony = CreateColonyAnts($A$)
   while (!stop_condition($\phi_c$, colony))
   |    for (ant in colony)
   |    |    route = TryFindRoute(ant)
   |    |    if (route != null)
   |    |    |    child = Reinforcement($G(N,E)$,route)
   |    |    |    CheckAndSetBestRoute(route)
   |    |    end
   |    end
   |    if (CheckAntsFindSolutionsOrSeveralStucks(colony))
   |    |    Evaporation($G(N, E)$)
   |    end
   end
   P = SelectConvergencePath(colony)
```

Some researchers [25] [42] show that the algorithm works fine in an environment with fewer states or nodes, but it has more unusual behaviors with many states. Therefore, for broader environments, it is necessary to establish some interactions to guarantee a better performance of the algorithm.

4.2.3 Global Interactions - Random Walks

The following proposes adds global interactions to the standard algorithm, known as *Random Walks*. The aim of these techniques is reducing the convergence time of the algorithm. To this end, a random walk helps to initialize the pheromone values of edges with random values for reducing the convergence time of the ACO algorithm. In this way, the random walk can be seen as a preprocessing technique, which allows estimating randomly good pheromone values. Moreover, the addition of these random walk confers the initial randomness of the algorithm. For this reason, it is not necessary to implement the pseudo-random generator technique, and the Equation 4.2 is reduced to Equation 4.6.

$$p_{ij}^k = \frac{q_{ij}}{\sum_{l \in N} q_{il}} \tag{4.6a}$$

where q_{ij} is:

$$q_{ij} = \tau_{ij}^{\alpha} \cdot \nu_{ij}^{\beta} \tag{4.6b}$$

The evaporation and reinforcement techniques are equal to *ACOv0* which follows the Equations 4.3, and 4.5. In this work, we proposed two different criteria for the random walk: Proximity Random Walk and Semi-Proximity Random Walk.

1. **Proximity Random Walk.** It is a quasi-random process based on the *proximity* of a node and its neighbor nodes, Equation 4.1. This criterion seeks to favor the finding of linear paths. The steps of this process are the following.

   1) Initialize the environment, the parameters of random walks (N_{rw} number of random walks, M_{rw} random movements by each ant), and a random walker ant.
   2) The random walker ant tries to find a route to the final node.
      a) In each step, the ant gets the proximity of each of the next possible nodes. Then, the M_{rw} nodes with the best proximity are filtered, Fig. 4.4. Later, the algorithm choices randomly one of these nodes.
      b) Repeat a) until reaching the final node. The proximity measure gave us a high probability that the process reaches the target node. However, it could appear stuck conditions, resulting in the discarding of the current route.
   3) If the ant finds a route, a reinforcement over the environment takes place using Equation 4.3.

4) After each random walk finding, an evaporation step occurs using Equation 4.5.
5) Repeat steps 2, 3 and 4 until complete a N_{rw} number of random walks.

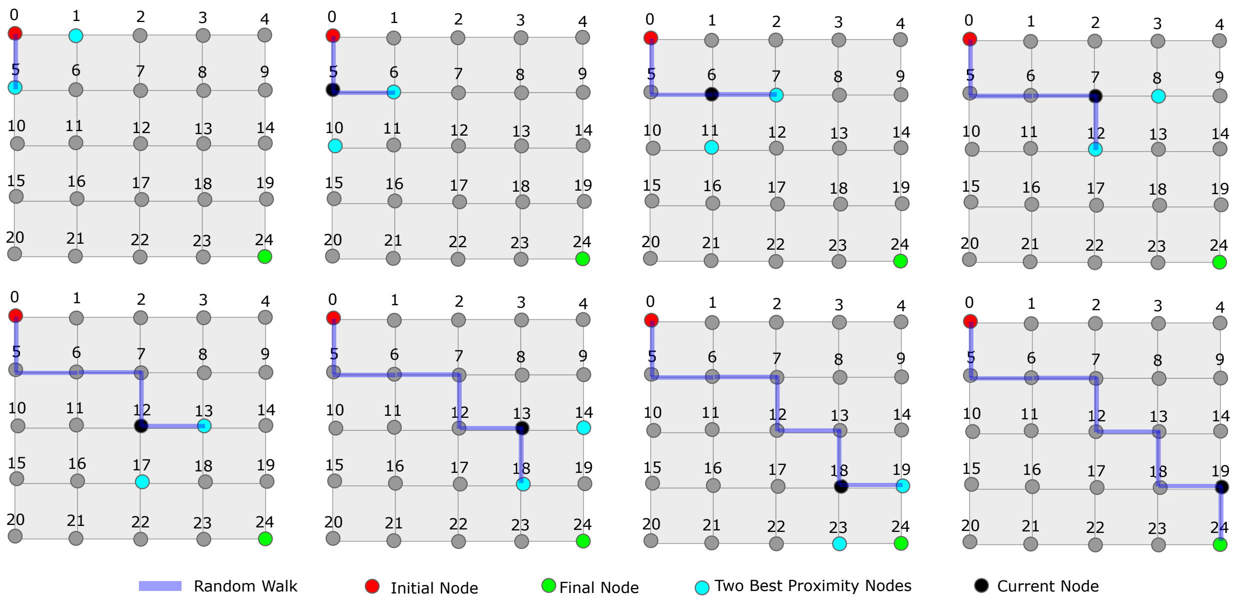


Figure 4.3: Proximity Random Walk Process with $M_{rw} = 2$

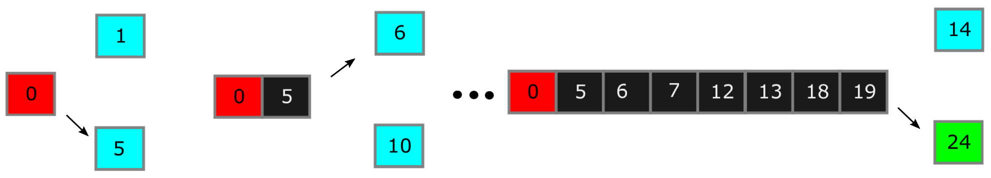


Figure 4.4: Final Route of Proximity Random Walk

2. **Semi-Proximity Random Walk.** It is also a quasi-random process based on the random choice of nodes and the *proximity* heuristic, Equation 4.1. This criterion seeks to favor the finding of not linear paths. The steps of this process are the following:

   1) Initialize the environment, the parameters of random walks (N_{rw} number of random walks, RN_{rw} number of intermediate random nodes), and a random walker ant.
   2) The random walker ant gets a RN_{rw} number of intermediate random nodes between the initial and final node, Fig 4.5. The algorithm always selects a new random number that is closer to the objective than the previous one. This action allows to select more distant random nodes. In this point, we have an order list of RN_{rw} unlinked nodes, Fig. 4.6.
   3) The random walker ant tries to link the nodes of this list, for finding a feasible route to the final node.

a) In each step, the ant gets two nodes of the list and tries to connect them following the best proximity in each step. In other words, it is like to apply a proximity random walk, but using $M_{rw} = 1$ in each step. If the algorithm does not achieve to connect the nodes, the ant gets stuck.

b) Repeat a) process with each pair of nodes until complete the path or stuck condition.

4) If the ant finds a route, a reinforcement over the environment takes place using Equation 4.5.

5) After each random walk finding, an evaporation step occurs using Equation 4.5.

6) Repeat steps 2, 3, 4 and 5 until complete a N_{rw} number of random walks.

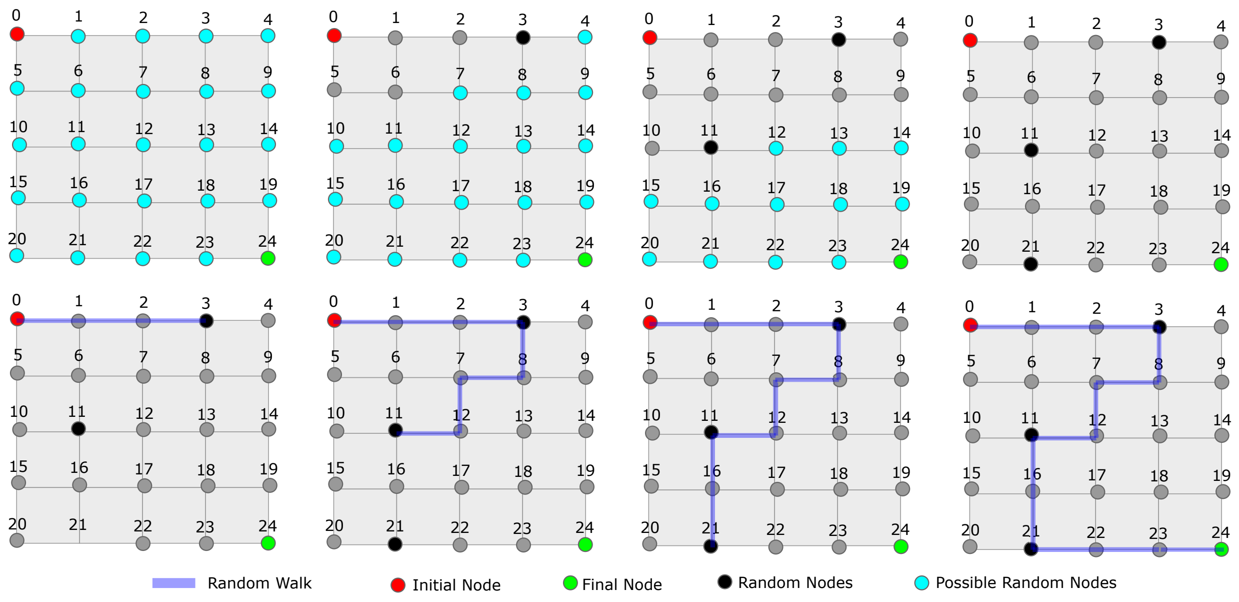


Figure 4.5: Semi-Proximity Random Walk Process with $RN_{rw} = 3$.

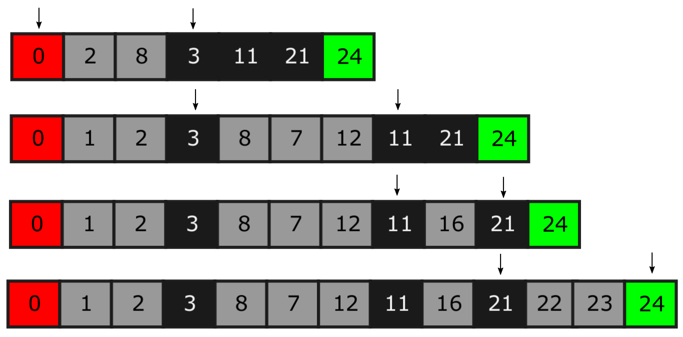


Figure 4.6: Final Route of Semi-Proximity Random Walk

4.2.4 Local Interactions

The local interactions aim to control the behavior of ants concerning obstacles that can appear in preprocessing time or execution time. The principal problem of the addition of many obstacles is that it can generate several stuck conditions that delay the algorithm and even prevent it from converging. The *ACOv0* can deal with the addition of obstacles if the environment is free or partially free. However, the *ACOv0* is not able to solve environments with many obstacles (or maze environments) by itself. For these reasons, we added some local interactions which can deal with these problems. The basic idea of a local iteration is removing ants and its possible contribution once the ant gets stuck.

In comparison with nature, these local interactions can simulate the real behavior of ants, when they die while searching a path. Furthermore, these additional features allow us to generate a more adaptive behavior when we generate obstacles in execution time. The obstacle apparition is commonly on real-life path planning simulation, where the obstacles can represent real risks (floods, fires, or others). The present work proposes three different variants of local interactions based on local search, pheromone decay, and additional heuristic.

1. **Local Search Interaction (Forbidden List) - *ACOv1***

   This algorithm uses the same parameters and conditions as the *ACOv0* but adds a new interaction known as **forbidding nodes**. This feature forbids one by one, the nodes that bring us to a stuck condition. To this end, the algorithm requires a list of forbidden nodes and a local search. The local search detects if a node can be forbidden, and the list stores it. If a node is forbidden, any ant can visit it. In this variant, we altered the **movement choice** technique, by the following process:

   (a) *Checking Previous Nodes:* If n_j is already visited $\Rightarrow N = N - \{j\}$.
   (b) *Forbidden List Interaction:* If $n_j \in F$ (forbidden list) $\Rightarrow N = N - \{j\}$.
   (c) *Probability Choice:* Following Eq. 4.2.

   The reinforcement and evaporation of pheromones of *ACOv0* are not altered.

   Procedure:

   (a) Initialize the environment, parameters, and colony of ants.
   (b) One ant of the colony tries to find a route to the final node, using the new movement choice technique.
      i. By the **checking previous nodes** interaction, the ant could get stuck in some situations, and not find a route.
      ii. By the interaction of the **forbidden nodes**, the ant could not move to the nodes of the forbidden list.
   (c) If the ant gets stuck, the algorithm takes the last node and performs the local search. Then, if it is possible to forbid the node, the algorithm stores it in the list.
   (d) If the ant finds a route, a reinforcement over the environment takes place, using Eq. 4.3.

(e) Repeat steps 2, 3, and 4 until complete an episode. An episode occurs when all the ants have tried to find a route.

(f) After each episode, an evaporation step occurs if at least one ant of the colony found a route, or if the ants get stuck several times with the same last node. In this step, the Equation 4.5 is used.

(g) The algorithm continues until the most ants can get the same route. The converge percentage ϕ_c defines this stop condition.

Local Search:

The local search is the main feature in this algorithm. As we have seen, it uses the last node of a stuck route. This analysis is performing in the following way:

- If the current node has one neighbor node, forbid the current node, Fig. 4.7.
- If the current node has two, three, or four neighbor nodes, check the possibility of arrives at all these nodes without using the current node. If this condition is possible, then forbid the current node, Fig. 4.7.

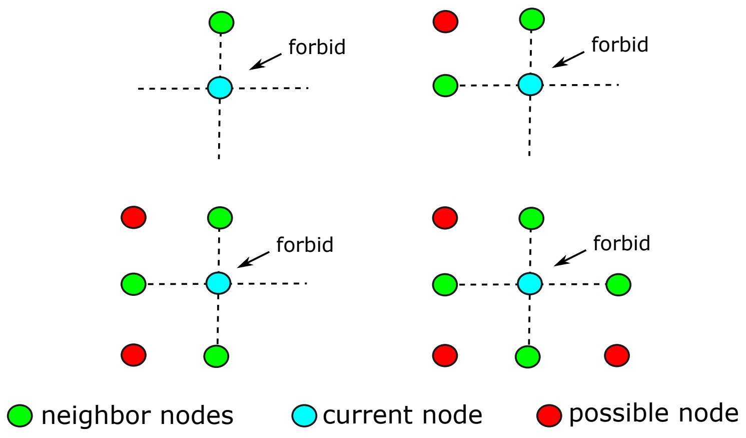


Figure 4.7: Local Search - Minimal Conditions for Forbidding a Node

2. **Local Pheromone Reduction - *ACOv2* and *ACOv4***

This version implements another local interaction based on the pheromones. The idea is to **reduce the pheromones** each time that an ant gets stuck. In this sense, the reduction can give a notion of what action can bring us to a stuck situation in the long term. This pheromone reduction can be produced of two ways over the wrong path, or the neighbor nodes using a vicinity value.

- ***Wrong Path Variant - ACOv2:*** The pheromone reduction only takes place in the last nodes of that wrong path. A new parameter RN_k indicates the number of the nodes, which will be reduced. Figure 4.8 indicates this process in a graphical way.

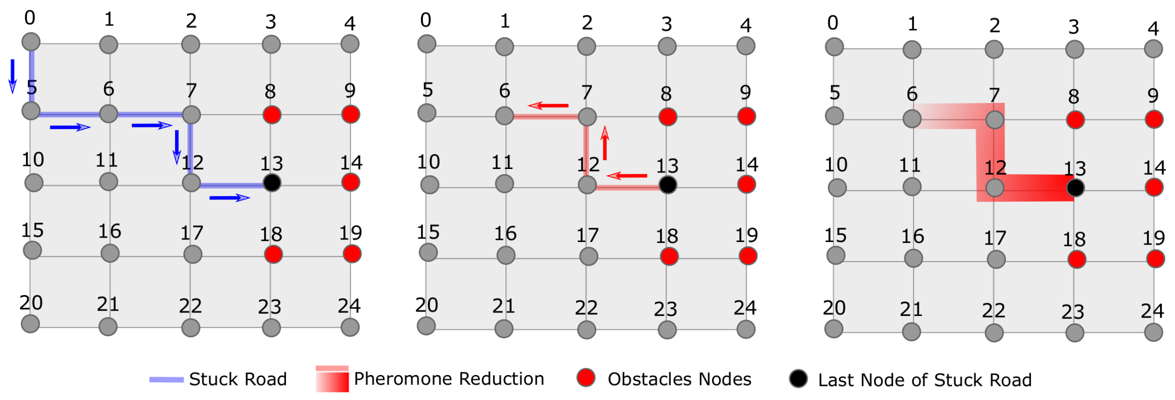


Figure 4.8: *ACOv2* Graphical Process with $RN_k = 3$

- ***Neighbour Nodes Variant - ACOv4:*** This variant implements a reduction of pheromones according to the vicinity V_k of the last node. In other words, each time that an ant gets stuck, the algorithm takes the last node and applies a reduction over the neighbor edges of the last node. It is essential to set a parameter V_k to specify the level of propagation. Figure 4.9 indicates this process in a graphical way.

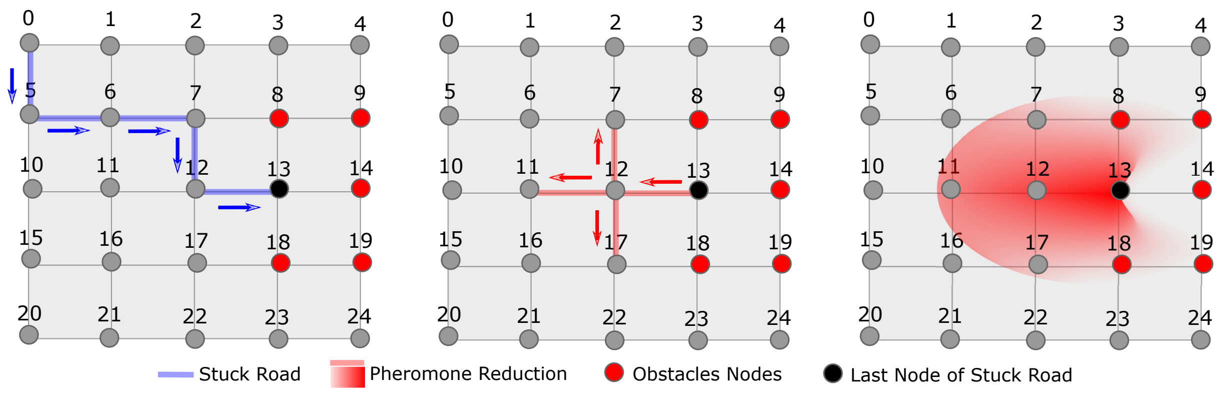


Figure 4.9: *ACOv4* Graphical Process with $V_k = 2$

It is essential to mention that this algorithm uses the same parameters, reinforcement, and evaporation process of the *ACOv0*. However, the environment must have an initial amount of pheromones greater than zero; it is $\tau_0 > 0$. Besides, in the initial stages, the ants must know about this initial amount of pheromones. For this reason, the pseudo-generator is changed to the Equation 4.7.

$$p_{ij}^k = \frac{r_{ij}}{\sum_{l \in N} r_{il}} \tag{4.7a}$$

where r_{ij} is:

$$r_{ij} = \begin{cases} q_{ij}^1 & if \quad \tau_{ij} = \tau_0 \\ q_{ij}^2 & if \quad episode < \delta \\ q_{ij} & otherwise \end{cases} \tag{4.7b}$$

, and

$$q_{ij} = \tau_{ij}^{\alpha} \cdot \nu_{ij}^{\beta} \tag{4.7c}$$

$$q_{ij}^1 = \nu_{ij}^{\alpha} \cdot \frac{1}{d_{ij}}^{\beta} \tag{4.7d}$$

$$q_{ij}^2 = \nu_{ij}^{\alpha} \cdot \frac{1}{d_{ij}}^{\beta} \cdot \tau_{ij}^{\beta} \tag{4.7e}$$

Procedure:

(a) Initialize the environment, parameters, and colony of ants.

(b) One ant of the colony tries to find a route to the final node, using the variant of the pseudo-generator, Eq. 4.7.

  i. By the **checking previous nodes** interaction, the ant could get stuck in some situations, and not find a route.

  ii. By **reducing the number of pheromones**, the ant should avoid edges with less amount of pheromones.

(c) If the ant gets stuck, the algorithm takes the last nodes, applies a specific variant *ACOv2* or *ACOv4*, and reduces the pheromones over the edges. This reduction is progressive; in other words, the closest node receives more reduction than the furthest nodes, Fig. 4.8, 4.9.

(d) If the ant finds a route, a reinforcement over the environment takes place, using Eq. 4.3.

(e) Repeat steps 2, 3, and 4 until complete an episode. An episode occurs when all the ants have tried to find a route.

(f) After each episode, an evaporation step occurs if at least one ant of the colony found a route, or if the ants get stuck several times with the same last node. In this step, the Equation 4.5 is used.

(g) The algorithm continues until the most ants can get the same route. The converge percentage ϕ_c defines this stop condition.

3. **Local Necrophoresis Value Reinforcement *ACOv3* and *ACOv5***

The previous algorithm alters the normal pheromones, which are needed to guarantee the convergence of the algorithm. This action could cause unexpected behaviors in the solutions of the algorithm. In order to avoid this direct interaction, it was necessary to establish another method to control the stuck routes. For this reason, this version proposes a method based on another natural behavior of the ants, know as *Necrophoresis* [43]. In this natural event, another type of ants participate called *Ant Undertaker*. These ants carry the dead bodies of colony members from the environment to the nest. This action aims to confer healthcare to the environment, for preventing diseases or infections.

According to *Necrophoresis* behavior, we proposes a local interaction that consists in the reinforcement of another heuristic value called **necrophoresis value** κ_{ij}. This heuristic value indicates a measure of how likely it would get stuck if we take a direction. Therefore, the ants should avoid the places with high κ_{ij}, avoiding stuck conditions in the long term. The algorithm is similar to version two; however, now, instead of reducing the pheromones, we reinforcement the necrophoresis value. Thus, each time that an ant gets stuck, an **ant undertaker reinforces the necrophoresis value**. In this way, the analogy with real behavior is the presence of **ants undertakers** that indicates a possible unfeasible path. Furthermore, in this version, it is not necessary to initialize the number of pheromones with a number different of zero.

This version could perform a reinforcement of necrophoresis value κ_{ij} in two ways: over the last nodes of a wrong path, or the neighbors of the last node.

- ***Wrong Path Variant - ACOv3:*** The necrophoresis value κ_{ij} reinforcement only takes place in the last nodes of that wrong path. A parameter RN_k indicates the number of the nodes which will be reinforced. Figure 4.10 indicates this process in a graphical way.

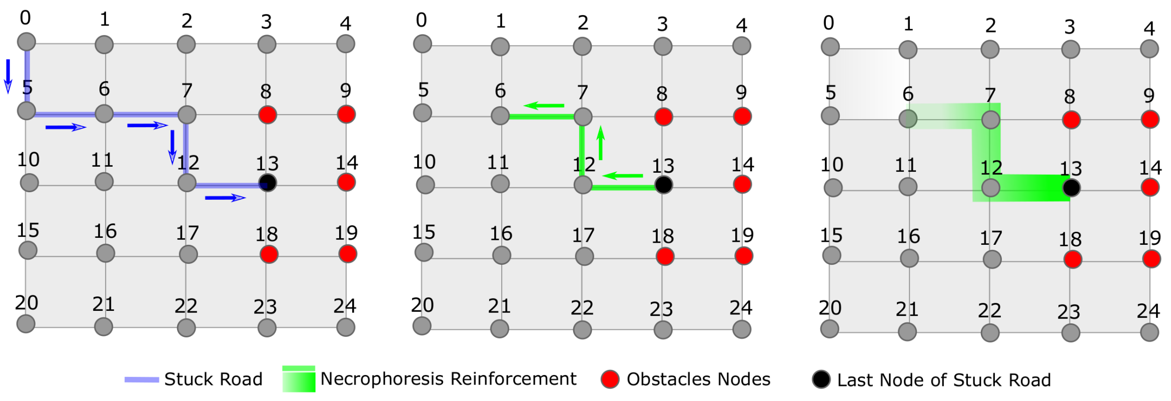


Figure 4.10: *ACOv3* Graphical Process with $RN_k = 3$

- ***Neighbour Nodes Variant - ACOv5:*** This variant implements a reinforcement of necrophoresis value κ_{ij} according to the vicinity V_k of the last node. In other words, each time that an ant gets stuck, the algorithm takes the last node and applies a reinforcement over the neighbor edges of the last node. It is essential to set a parameter V_k to specify the level of propagation. Figure 4.11 indicates this process in a graphical way.

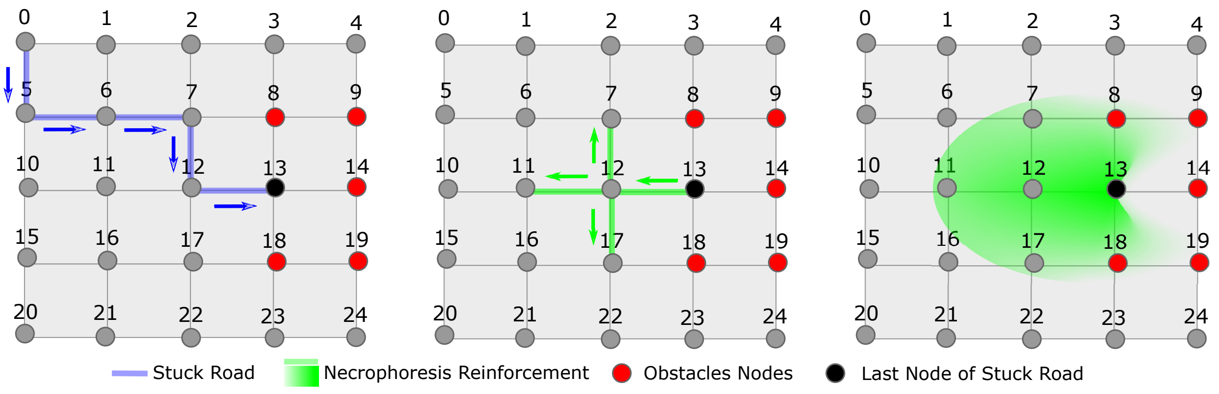


Figure 4.11: *ACOv5* Graphical Process with $V_k = 2$

In this version, the ant must take into account the necrophoresis value of κ_{ij}. Therefore, the algorithm changes the Equation 4.2 to the Equation 4.8. The parameters, pheromone reinforcement, and evaporation are the same as the *ACOv0*.

$$p_{ij}^k = \frac{r_{ij}}{\sum_{l \in N} r_{il}} \tag{4.8a}$$

where r_{ij} is:

$$r_{ij} = \begin{cases} q_{ij}^1 & if \quad \tau_{ij} = \tau_0 \\ q_{ij}^2 & if \quad episode < \delta \\ q_{ij} & otherwise \end{cases} \tag{4.8b}$$

, and

$$q_{ij} = \tau_{ij}^{\alpha} \cdot \nu_{ij}^{\beta} \cdot (\frac{1}{\kappa_{ij}})^{\beta} \tag{4.8c}$$

$$q_{ij}^1 = \nu_{ij}^{\alpha} \cdot \frac{1}{d_{ij}}^{\beta} \tag{4.8d}$$

$$q_{ij}^2 = \nu_{ij}^{\alpha} \cdot (\frac{1}{d_{ij}})^{\beta} \cdot (\frac{1}{\kappa_{ij}})^{\beta} \tag{4.8e}$$

Procedure:

(a) Initialize the environment, parameters, and colony of ants.

(b) One ant of the colony tries to find a route to the final node, using the variant of the pseudo-generator, Eq. 4.8.

  i. By the **checking previous nodes** interaction, the ant could get stuck in some situations, and not find a route.
  ii. By **reinforcement the number of necrophoresis value** κ_{ij}, the ant should avoid edges with a high amount of necrophoresis value.

(c) If the ant gets stuck, the algorithm takes the last nodes, applies a specific variant *ACOv3* or *ACOv5* and increase the necrophoresis value over the edges. This reinforcement is progressive; in other words, the closest node receives more necrophoresis value κ_{ij} than the furthest nodes, Fig. 4.10, 4.11.

(d) If the ant finds a route, a pheromone reinforcement over the route takes place, using Eq. 4.3.

(e) Repeat steps 2, 3, and 4 until complete an episode. An episode occurs when all the ants have tried to find a route.

(f) After each episode, an pheromone evaporation step occurs if at least one ant of the colony found a route, or if the ants get stuck several times with the same last node. In this step, the Equation 4.5 is used.

(g) The algorithm continues until the most ants can get the same route. The converge percentage ϕ_c defines this stop condition.

4.2.5 Graphic Simulation

The graphic simulation will implement the best variants for the dynamic of the ants. This simulation aims to demonstrate the adaptive behavior of the algorithm. Nevertheless, in this phase, we had to introduce a new heuristic measure to get more appropriate three-dimensional behavior. This measure is the slope equation, which gave us an idea of height between nodes.

Slope Equation

At first implementations, the environment consisted of a graph $G(N, E)$ in two dimensions. Then, the graph added some intrusions and extrusion in some nodes to get a three-dimensional mesh, Fig 4.14. By this change, it is necessary to establish some weights which take in mind the slope or confer the notion of height. These conditions are necessary because it is not the same to walk in a climb than in a fall. Therefore, we proposed a technique to represent three-dimensional behavior using vectors and angles. In this way, let the current position n_i and the next possible node n_j, it is defined two vectors u and v, Fig 4.13. Then, we get the θ angle. However, the angles are in the rank $[-90, 90]$. This interval alters the calculated values of the probability function drastically. For this reason, we use a slope function, Equation 4.9, based on the *Gaussian Function* center in zero that confers a kind of normalization to the angles for avoiding the use of large numbers.

$$s_{ij} = \begin{cases} (1-\omega)e^{\frac{\theta^2}{2c^2}} & if \quad \theta > 0 \\ \\ \omega e^{\frac{\theta^2}{2c^2}} & if \quad \theta \leq 0 \end{cases} \tag{4.9}$$

Equation 4.9 returns three different behavior according to the value of ω parameter. Considering the *Gaussian Equation*, the values ω and $(1-\omega)$ indicate the highest points of the bell. In this way, using the parameter ω, we can adjust the priority of ascents and descents. Therefore, if the environment condition indicates that the initial and final nodes are at the same height, we could use a $\omega = 0.5$, which gives the same priority to ascents and descents, Fig. 4.12. Then, if the initial node is above the final node, we could use a value ω close to 1, which prioritizes the descents, Fig. 4.12. Finally, if the initial node is under the final node, we could use a value close to 0, which prioritizes the ascents. On the other hand, the parameter c controls the width of the Gaussian curve, which prioritizes a larger range angles, Fig. 4.12. This parameter avoids falling in really steep ascents or descents (ravines).

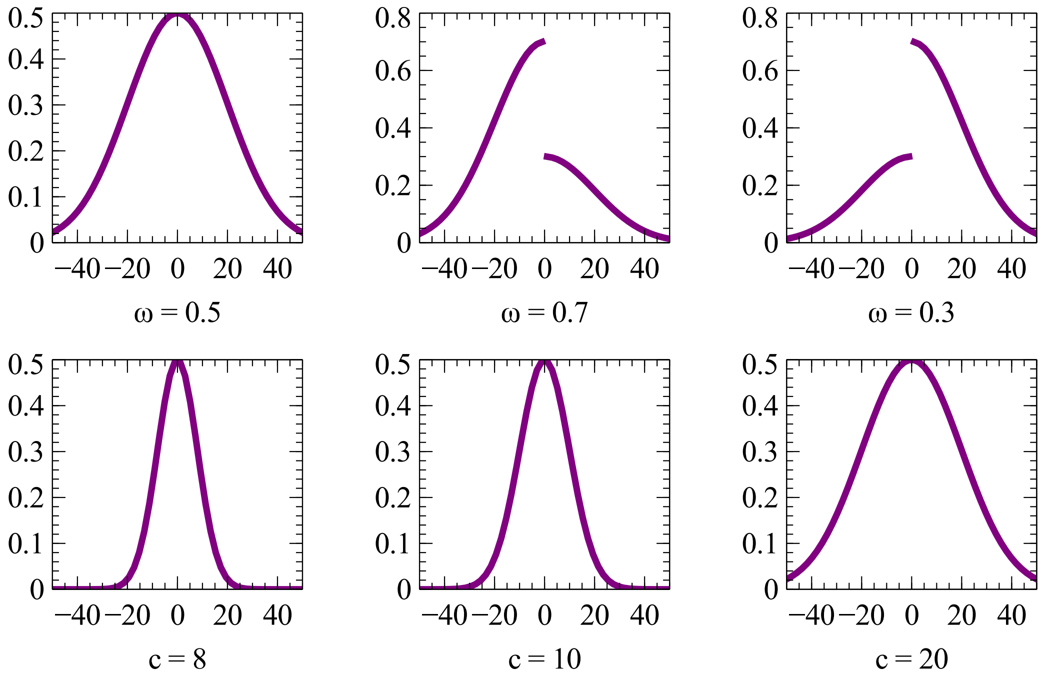


Figure 4.12: Slopes Functions with Different Paraters ω and c

It is important to mention that the slope function s_{ij} return values in the interval $(0, 1]$. Moreover, the slope s_{ij} is different from s_{ji} because, in the first case, it is a climb, and in the second case, it is a drop, Fig. 4.13.

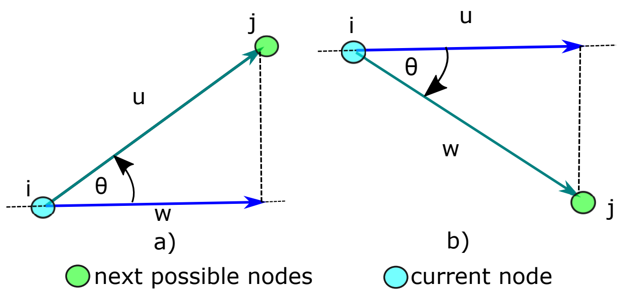


Figure 4.13: 3D Technique. *a) positive slope*, *b) negative slope.*

According to Equation 4.9 and using Random Walk 2, in this graphic simulation, we changed the Movement Choice, Equation 4.2, by Equation 4.10.

$$p_{ij}^{k} = \frac{q_{ij}}{\sum_{l \in N} q_{il}} \tag{4.10a}$$

where q_{ij} is:

$$q_{ij} = \tau_{ij}^{\alpha} \cdot (\nu_{ij} \cdot s_{ij})^{\beta} \tag{4.10b}$$

4.3 Experimental Setup

4.3.1 Data Generation Method

In this section, we defined the data sets, as well as a brief justification for its use.

The final graphic simulation of this work seeks to establish a versatile algorithm that fits a vast number of path planning problems. To this end, we decided to use graphs because many real issues can easily be translated into graphs. Besides, applying minimal changes, the proposed techniques can work with other representation such as cell decomposition, or road maps. The nodes of the graphs proceed from Wavefront .obj files, which can be produced from several software tools. In this project, we use Blender for creating these files by the ANT Landscape Tool [44], which generates random environments with different topographies. The acronyms ANT refers to Another Noise Tool, not to biological ants. In the first experiments, we worked with two-dimensional representations of these graphs. However, one of the objectives is working under three-dimensional conditions. To this end, using the ANT Landscape Tool, we create smooth intrusions and extrusions in some nodes to create three-dimensional meshes, Fig 4.14. These representations help us to model more real scenarios with different topographic conditions.

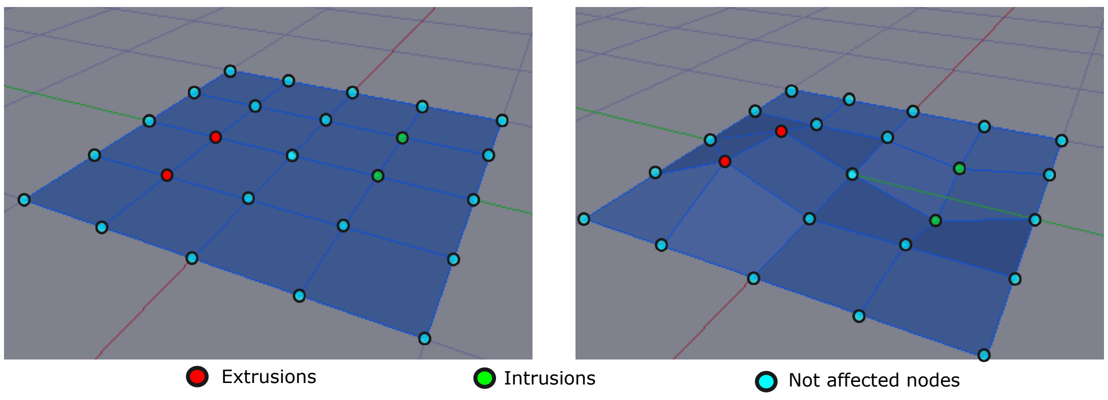
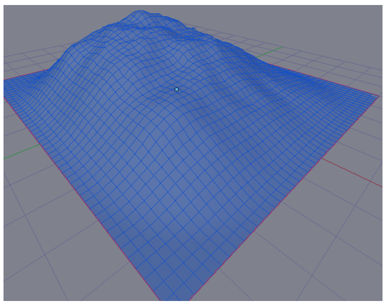
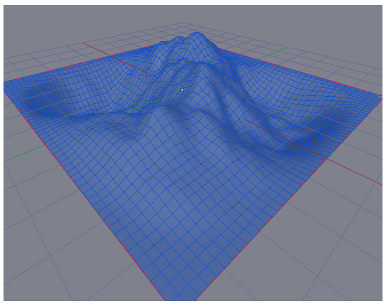
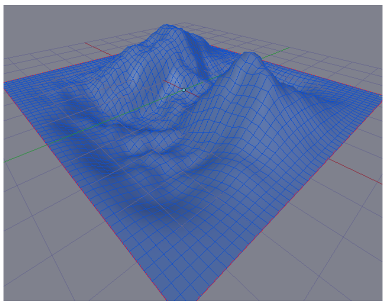


Figure 4.14: Mountain Environment: two-dimensional mesh without Extrusion or Intrusion, and three-dimensional mesh with Extrusion or Intrusion.

The data sets of this algorithm implement four different topographic instances: Mountain, Valley, Doble Valley, and Perlin Noise meshes. These representations of terrains help us to work under similar conditions for a comparison between the ACO algorithm versions, and with other implementations. Moreover, the data sets have different sizes. In other words, the terrains are the same, but with a different number of nodes and edges (the total dimension is not altered). This additional information can produce a better realistic environment because the movements will be smoother and visually more appealing. Moreover, it is essential to establish a benchmark upon which measure the performance of our model. In this way, it was used a complete search algorithm, known as the Dijkstra algorithm, Chapter 3, for finding the best solutions 4.2.


(a) Mountain Environment


(b) Doble Valley Environment


(c) Valley Environment

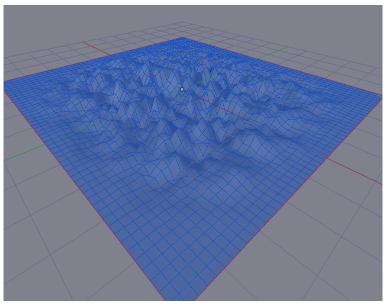

(d) Perlin Noise Environment

Figure 4.15: Different Experimental Environments

The experiments were performed for the data set listed in Table 4.1. Column $|N|$ contains the number of nodes graph, column $|E|$ presents the number of edges belonging to the graph, and column $|E_o|$ presents the number of edges considering the obstacles of Maze Environments. In the following, we will refer to this set of instances as the *experimental set*.

Sample	**Size**	$\lvert N\rvert$	$\lvert E\rvert$	$\lvert E_o\rvert$
1	10	100	180	117
2	12	144	264	174
3	14	196	364	233
4	16	256	480	311
5	18	324	612	402
6	20	400	760	511
7	22	484	924	596
8	24	576	1104	722
9	26	676	1300	800
10	28	784	1512	949
11	30	900	1740	1104
12	32	1024	1984	1276
13	34	1156	2244	1453
14	36	1296	2520	1592
15	38	1444	2812	1784
16	40	1600	3120	1985
17	42	1764	3444	2196
18	44	1936	3784	2388
19	46	2116	4140	2645
20	48	2304	4512	2892
21	50	2500	4900	3113

Table 4.1: Experimental Set for Mountain, Valley, U-terrain and Perlin Noise

The present algorithm also must have some adaptive characteristics; in other words, it must be able to respond to the appearance of obstacles. For this reason, these environments must allow the introduction of obstacles for testing the adaptive condition. In this way, we can insert an obstacle in two ways:

- An edge can be an obstacle, so an ant can not move through it.
- A node can be an obstacle, so an ant can not move to it through any of its possible connections. This obstacle forbids four edges at the same time.

Size	**Best Cost**	**Best Cost With Obstacles**			
		Mountain	Perlin	Valley	Double Valley
10	20	23,78	20,14	20,89	21,78
12	20	24,78	20,42	21,53	22,10
14	20	23,97	20,65	20,89	21,87
16	20	25,85	22,40	23,63	23,58
18	20	24,36	21,69	21,88	21,90
20	20	24,09	20,46	21,82	21,70
22	20	24,48	20,70	21,61	22,83
24	20	25,00	22,30	22,55	22,98
26	20	24,74	22,23	21,95	23,18
28	20	25,73	22,48	22,89	23,90
30	20	26,60	23,22	23,80	23,55
32	20	23,89	21,22	21,85	21,88
34	20	24,55	21,62	21,69	22,43
36	20	24,98	21,93	22,80	23,35
38	20	24,55	22,66	22,64	23,17
40	20	24,32	22,75	21,86	22,81
42	20	25,43	23,37	22,72	23,93
44	20	25,19	23,03	23,04	23,07
46	20	26,38	24,11	23,77	24,14
48	20	24,67	23,16	22,86	23,01
50	20	24,18	23,14	21,69	22,68

Table 4.2: Cost Values of the Experimental Set

In some experiments, it was necessary to establish some predefined environments with obstacles, Table 4.1. These environments have many obstacles in the form of a maze and were defined as **maze environments**. We used a randomized version of the Kruskal algorithm for creating these environments, which allows us to generate random maze configurations. Figure 4.16 shows some examples of the random maze environments generated. The Kruskal algorithm [45] is a method for producing a minimal spanning tree from a weighted graph. The randomized version differs from the original in the searching of the minimal cost edges, instead of that, the proposed version search random cost edges.

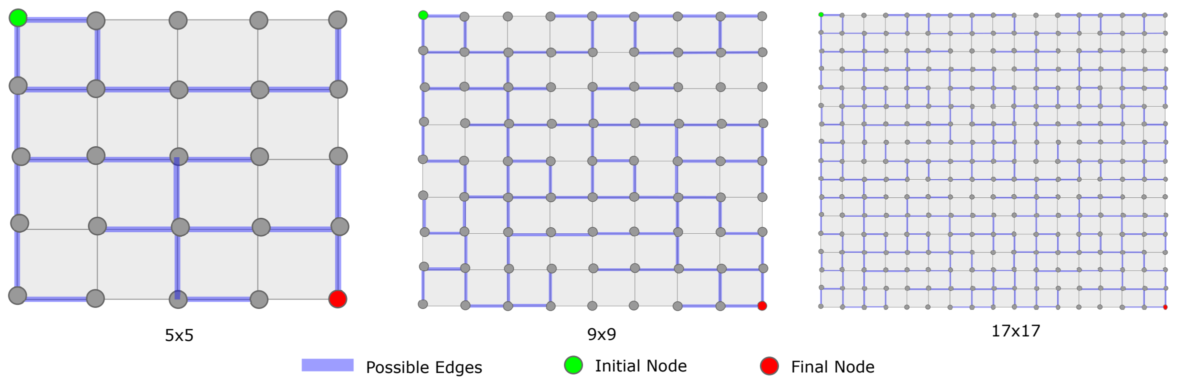


Figure 4.16: Random Maze Environments Examples: 5×5, 9×9 and 17×17

4.3.2 Analysis Method

In this subsection, we defined the ways to study the model proposal. To evaluate and compare the performance of the model proposed in Section 4.2, a series of experiments was planned. These experiments were designed with certain research intentions:

1. Identification of good values for ACO parameters.
2. Evaluation of basic ACO algorithm using the proposed reinforcement, evaporation, and movement choice methods
3. Performance evaluation of global and local interactions.
4. Comparison of the ACO algorithm and other path planning techniques using the best performing ACO variant.
5. Evaluation of the adaptability behavior using a graphical simulation.

To these ends, we defined a set of measures for testing the algorithms. These measures helped us to understand the algorithm performance in terms of duration, precision, learning, exploration, exploitation, and adaptability. The performance measures are the following:

- **Execution Time (ET)** is a measure of duration in units of time. It is the time until getting the convergence. This measure depends on the computer and the programming language. In this way, we used a clock function of C# (Stopwatch Class) for getting the initial and final time. Then, the execution time follows the Equation 4.11.

$$ET = t_f - t_i \tag{4.11}$$

  where ET is the total execution time, t_f is the final time, and t_i is the initial time.

- **Accuracy (AC)** is a measure of precision used to detect what is the level of confidence of our algorithm in percentage. This measure indicates the approximation of predicted solution to the optimal solution. To this end, we used the cost function defined in Equation 2.1. Thus, we measured the optimal solution of the experimental set, Table 4.2. According to these considerations, the accuracy is the Equation 4.12.

$$AC = \frac{C_{opt}}{C_{pred}} \tag{4.12}$$

where C_{opt} is the cost of the best optimal solution of the data set, and C_{pred} is the cost of the predicted solution by the used method.

- **Episodes Number (EP)** is a measure of duration in terms of iterations. It is the number of iterations until getting the convergence. This measure depends on the algorithm design.
- **Stuck Roads Number (SR)** is a measure of duration, exploration, and exploitation. It is the number of times that an ant gets stuck. This measure depends on the algorithm design.
- **Relative Visited Nodes Number (VNr)** is a measure of exploration and exploitation. It is the percentage of nodes visited by the ants according to the total number of nodes in the environment. This measure depends on the algorithm design.
- **Relative Visited Edges Number (VEr)** is a measure of exploration and exploitation. It is the percentage of edges visited by the ants according to the total number of edges in the environment. This measure depends on the algorithm design.
- **Percentage of Environment Learned (PE)** is a measure of learning and precision. It is how many positions have been learned by the ants. This measure considers a static ant that moves only following the maximum number of pheromones. Thus, this ants helps us to measure, from how many positions of the environment, an ant can find the end by following only the pheromones. Figure 4.17 explains this behavior in a better way. In this figure, the expected extra learned positions are in the green zone.

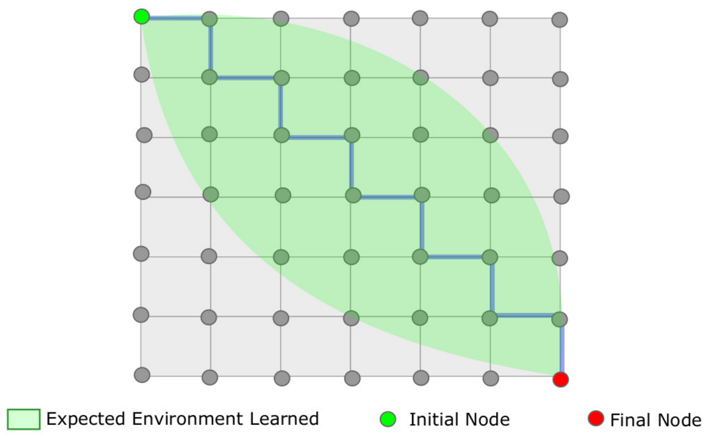


Figure 4.17: Percentage of Environment Learned (PE).

Software and Hardware Considerations

These experiments were executed in a computer with the following specifications:

- Processor: Intel Core(TM) i7-7700HQ CPU @2.80GHz 2.81 GHz
- RAM: 16.0GB
- System type: 64-bit Windows 10 Operating System

The algorithms were programmed in C#, and the graphic simulation used Unity 3D framework.

Biplot and Principal Component Analysis Methods

This work implements several performance measures. For this reason, it is necessary to use an accurate way to analyze these results. The multivariate statistical analysis gave us several tools to deal with this kind of issue. In this work, we decided to use a Biplot method based on Principal Component Analysis (PCA). A Biplot [46] is a two-dimensional representation of multivariate data. This plot shows a point for each **observation** (one execution of the algorithm) , and a point for each **variable** (performance measure). The prefix *bi* refers to two kinds of points representations, not to a bi-dimensional plot [46].

In this sense, for analyzing the result, we expected to obtain results that can be categorized in the following groups:

- Duration: Execution Time (ET), Episodes Number (EP), Stuck Roads (SR).
- Precision: Accuracy (AC), Percentage of Environment Learned (PE).
- Exploration and Exploitation: Relative Visited Nodes (VNr), Relative Visited Edges (VEr).

How to interpret a Biplot graph?

The following explanation is based on the book *Methods of Multivariate Analysis* of Rencher Alvin C, [46]. We start the interpretation using the matrix $\boldsymbol{Y}$, where each n rows correspond to the observations (each execution of the algorithm), and the p columns correspond to the variables (our performance measures).

$$\mathbf{Y} = \begin{pmatrix} \mathbf{y}'_1 \\ \mathbf{y}'_2 \\ \vdots \\ \mathbf{y}'_i \\ \vdots \\ \mathbf{y}'_n \end{pmatrix} = \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1j} & \cdots & y_{1p} \\ y_{21} & y_{22} & \cdots & y_{2j} & \cdots & y_{2p} \\ \vdots & \vdots & & \vdots & & \vdots \\ y_{i1} & y_{i2} & \cdots & y_{ij} & \cdots & y_{ip} \\ \vdots & \vdots & & \vdots & & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nj} & \cdots & y_{np} \end{pmatrix}$$

$$\mathbf{y_i} = \begin{pmatrix} y_{i1} \\ y_{i2} \\ \vdots \\ y_{ij} \\ \vdots \\ y_{ip} \end{pmatrix}, \quad \overline{\mathbf{y}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{y}_i = \begin{pmatrix} \bar{y}_1 \\ \bar{y}_2 \\ \vdots \\ \bar{y}_p \end{pmatrix}$$

We perform a Principal Component Analysis (PCA), and we get

$$\mathbf{Z} = \mathbf{Y_c}\ \mathbf{A} \tag{4.13}$$

$$\mathbf{Z} = \begin{pmatrix} \mathbf{z}_1' \\ \mathbf{z}_2' \\ \vdots \\ \mathbf{z}_n' \end{pmatrix}, \quad \mathbf{Y}_c = \begin{pmatrix} (\mathbf{y}_1 - \overline{\mathbf{y}})' \\ (\mathbf{y}_2 - \overline{\mathbf{y}})' \\ \vdots \\ (\mathbf{y}_n - \overline{\mathbf{y}})' \end{pmatrix}$$

where $\mathbf{A} = \left(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_p\right)$ is the $p \times p$ matrix whose columns are (normalized) eigenvectors of $\mathbf{S}$ (the covariance matrix). For drawing the Biplot, we need to consider the principal components 1 and 2 which have the best representation of $\mathbf{Yc}$. Therefore, as the eigenvectors $\mathbf{a_j}$ of the symmetric matrix $\mathbf{S}$ are mutually orthogonal, then $\mathbf{A} = \left(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_p\right)$ is an orthogonal matrix and $\mathbf{AA}' = \mathbf{I}$. Multiplying 4.13 on the right by $\mathbf{A}'$, we obtain

$$\mathbf{Y_c} = \mathbf{ZA}' \tag{4.14}$$

The best two-dimensional representation of $\mathbf{Y_c}$ is given by taking the first two columns of $\mathbf{Z}$ and the first two columns of $\mathbf{A}$. If the resulting matrices are denoted by $\mathbf{Z_2}$ and $\mathbf{A_2}$, we have

$$\mathbf{Y}_c \cong \mathbf{Z}_2\mathbf{A}_2' = \begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \\ \vdots & \vdots \\ z_{n1} & z_{n2} \end{pmatrix} \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{p1} \\ a_{12} & a_{22} & \cdots & a_{p2} \end{pmatrix} \tag{4.15}$$

The elements of 4.15 are of the form.

$$y_{ij} - \bar{y}_j \cong z_{i1}a_{j1} + z_{i2}a_{j2}, \quad i = 1, 2, \ldots, n; j = 1, 2, \ldots, p$$

Thus each observation is represented as a linear combination. Therefore, we plot the points (z_{i1}, z_{i2}), $i = 1, 2, \ldots, n$ which represent the coordinates for the n observations; and the points (a_{j1}, a_{j2}), $j = 1, 2, \ldots, p$ which correspond to the coordinates for the p variables. To distinguish them and to show relationship of the observations to the variable, the points (a_{j1}, a_{j2}) are connected to the origin with a straight line.

Once the biplot is drawn, the following different conclusions can be interpreted:

- **Principal Components Fit.** The adequacy of the fit of the principal components can be examined with the two eigenvalues of λ_1 (PC1) and λ_2 (PC2) of $\mathbf{S}$. Thus a significant value (close to 100%) of

  $$(Explained)\ PC1 + PC2 = \frac{\lambda_1 + \lambda_2}{\sum_{i=1}^{p} \lambda_i} * 100\%$$

  would indicate that Y_c is well represented by the plot.

- **Level of Variability of each Variable.** The length of lines drawn by each axis point (a_{j1}, a_{j2}) indicates a perception of the variance of the corresponding variable. Thus a long length indicates a high level of variability in the corresponding variable.

- **Correlation between Variables.** The cosine of the angle between the lines drawn to each pair of axis points (a_{j1}, a_{j2}) and (a_{k1}, a_{k2}) shows the correlation between the two corresponding variables. Thus a small angle between two vectors indicates that the two variables are highly correlated, two variables whose vectors form a 90° angle are uncorrelated, and an angle greater than 90° indicates that the variables are negatively correlated.

- **Relevance of a Variable over an Observation.** The values of each variable corresponding to the observation y_i (corrected for means) are related to the perpendicular projection of the point (z_{1i}, z_{2i}) on the p vectors from the origin to the points (a_{j1}, a_{j2}). Thus, the further from the origin a projection falls on a line, the larger the value of the observation on that variable. In the same way, if the perpendicular projection falls into the opposite vector direction, the values of the observation on that variable will be small.

Chapter 5

Results and Discussion

In this chapter, we document our experiments of the model proposal with different ACO configurations. For the analysis of the model, we follow a progressive study. First, we present the ACO basic version or *ACOv0* for having a base to work. Then, using the *ACOv0*, we tested the global interactions, also known as random walks. Next, we evaluated the performance of the local interaction versions. Later, based on previous results, we determined the best ACO version. We used this final version for comparing with other already existed methods. Finally, we presented some qualitative results showing the adaptability of the algorithm in a graphical simulation.

It is important to note that all the results of this chapter show similar performance when works with small sizes meshes. For this reason, the analysis was focused on the results with the large mesh sizes. Furthermore, for each graph first, we describe some results, and then we write a discussion about these results.

5.1 Performance Evaluation of *ACOv0*

The first experiment pretended to be a base for the next experiments. In this way, we need to have an initial idea of the performance of this algorithm. The initial configuration was:

Version	ϕ_c	A	α	β	τ_0	ρ	δ
ACOv0	0.7	15	3	2	0	0.5	2

Table 5.1: Parameters of *ACOv0*

We executed this experiment $n = 30$ times, and the mean, median, and possible outliers of the performance measures were presented.

Figure 5.1 shows the execution time (ET) according to each size with different terrains.

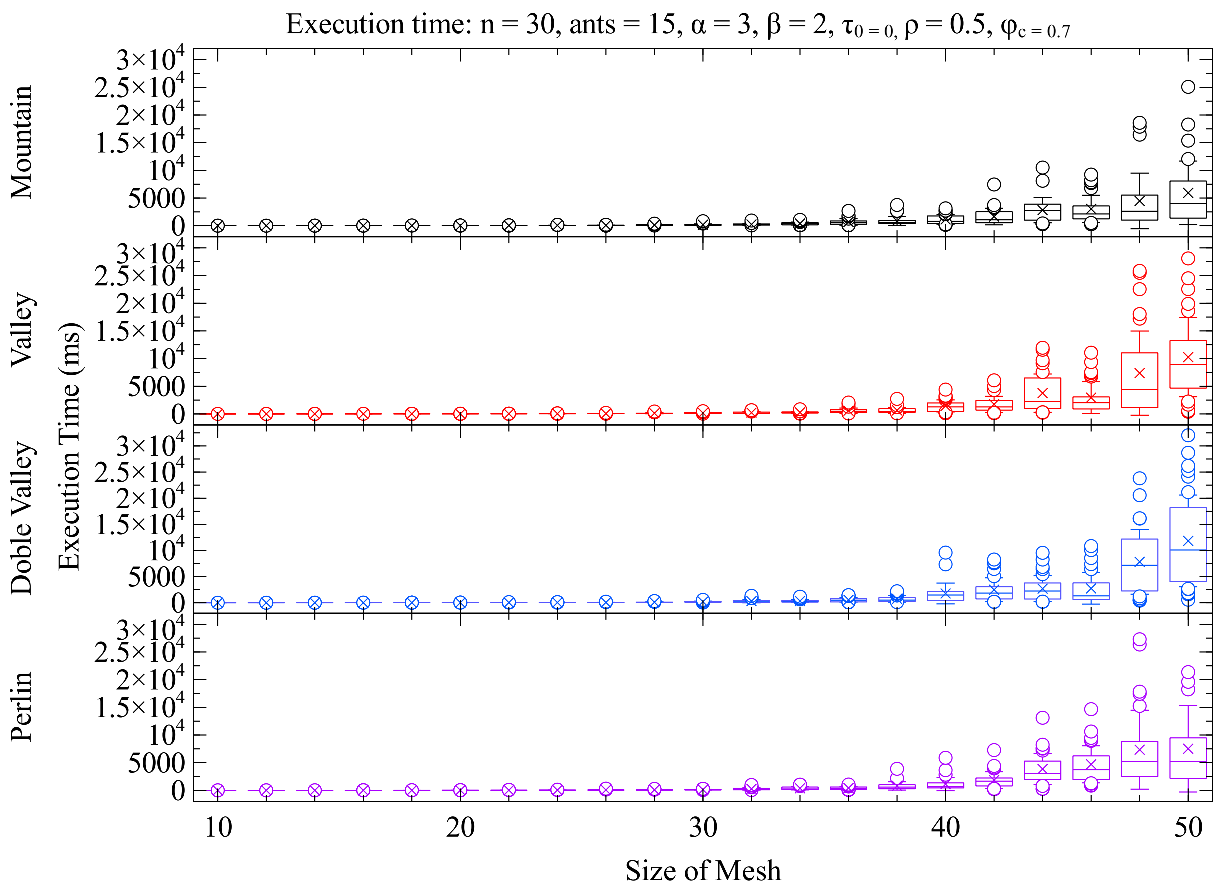


Figure 5.1: Execution Time by Size of Environment - *ACOv0*

According to these results, there are no notable differences between the execution times of each terrain; in other words, they present similar behaviors. The figure shows an increasing behavior in time as the size also increases. Furthermore, we can observe that the amount of outlier measures is higher with a large mesh size. This result can be due to the random behavior of ACO, which can provoke that the algorithm convergences at different times. Thus, as we increase the size, the randomness also increases, and hence the execution time. These outliers also can be due to the current state of the computer, which could generate unpredicted behaviors that alters the time results. In extreme conditions, with large sizes than 50, it would be feasible to execute these experiments in an external computer, not in a home computer.

Other results are the maximum and minimum time of the smallest and largest sizes, respectively. In mean and median, with large sizes, this time does not exceed the $10000ms$ or $10s$ for Double Valley meshes, and $5s$ for all the other terrains. Moreover, the minimum time with small terrains is close to $1s$. We can conclude that the times of large environments are high, considering the small scale of the experimental set. In this sense, we aimed to reduce these values in the next implementations.

Figure 5.2 shows the number of episodes (EP) that the algorithm needs to converge with a specific size.

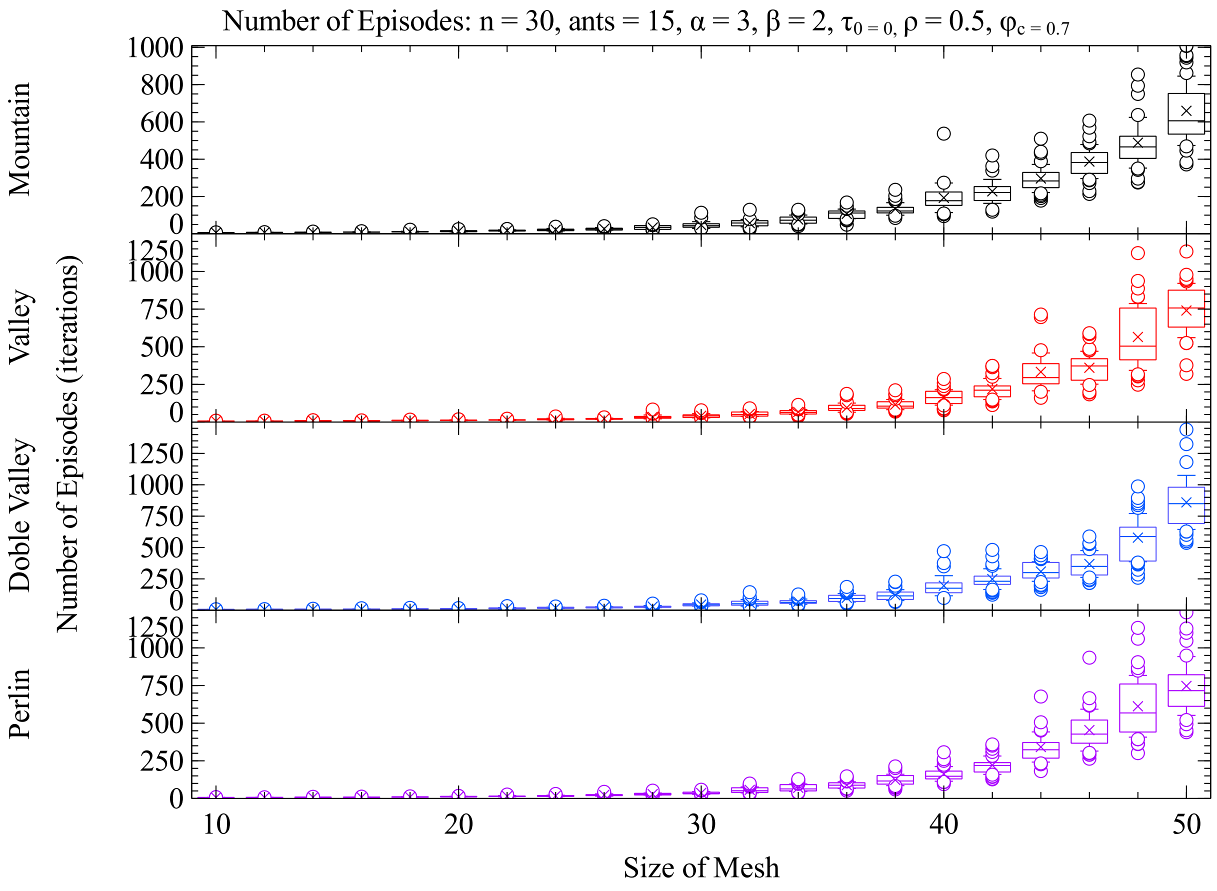


Figure 5.2: Episodes by Size of Environment - *ACOv0*

These results show that the episode number is almost similar in each terrain. The Mountain terrain seems to have increasing behavior smoother than the others, but it is not so significant. There is an evident increment of episodes as the size of mesh increase like the previous execution time results, Fig. 5.1. Moreover, there are several possibles outliers in the mesh with large sizes. We expected that these results resemblance to the execution time of Figure 5.1 because the episodes number (EP) is also a measure of duration. Nevertheless, the computer state does not affect the number of episodes. This measure, therefore, can give a theoretical idea of the global time complexity of the algorithm. In this way, in this algorithm, the results show an average of 600 episodes with large sizes of mesh. This value indicates that the algorithm takes 600 episodes by 15 ants to converge or 9000 iterations in total. In general, these results are very high, considering the implemented size of the mesh. For this reason, in succeeding versions, we aimed to reduce this complexity.

Figure 5.3 shows the accuracy (AC) according each size of mesh environment and environment type.

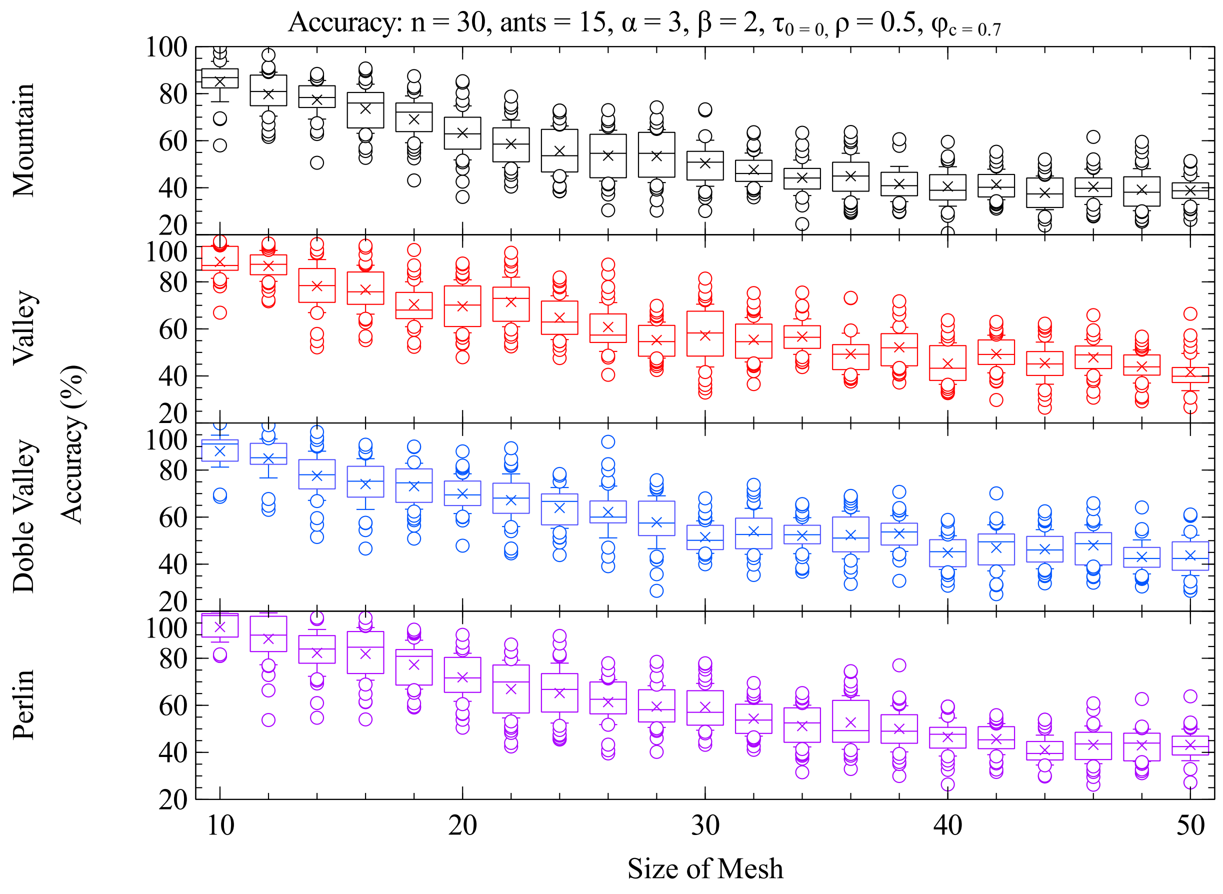


Figure 5.3: Accuracy by Size of Environment - *ACOv0*

These results indicate a high accuracy over the small environments and low accuracy over the larger environments. The smallest mesh shows results around 90%, and, in some cases, the algorithm gets optimal solutions. In general, we expected to no get always optimal solutions because the ACO metaheuristic does not guarantee to find an optimal solution, only a feasible solution. However, using large sizes, the results present an accuracy really low. In fact, in the large environments, the accuracy is less than 50% in most of the cases. For these reasons, in the next implementations, we aimed to improve these results with the addition of global and local interactions.

Furthermore, another result of interest is the behavior with each type of terrain. According to Fig. 5.3, the accuracy is a little better in the environment with Perlin Noise. Nevertheless, the difference is not too marked. In this sense, we could consider that all the methods have similar behaviors.

The following graph shows the percentage environment learned (PE) according to each size of the mesh environment and environment type.

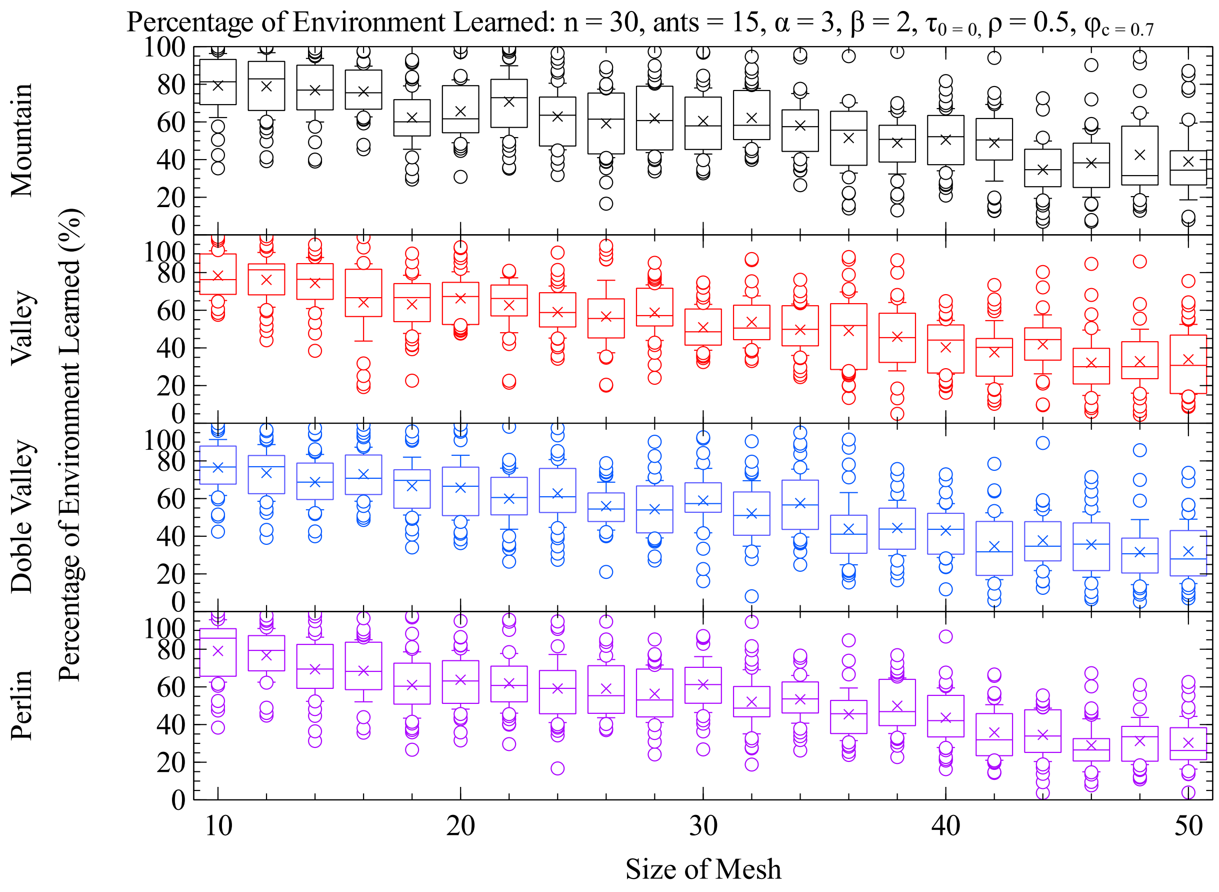


Figure 5.4: Percentage of Environment Learned by Size of Environment - *ACOv0*

The present work aims to find the route from the initial to the final position, that is to say, one to one. However, the nature of this algorithm provokes that the algorithm also learns to move from other initial positions (many to one path planning problem). According to these results, we can observe that this learning is higher in small environments than in large environments. Therefore, the capacity to move from other initial positions is higher in small terrains. Nevertheless, it is essential to think about the quality of these other solutions. For this reason, we also have to analyze the previously studied accuracy, Fig 5.3, which influences the quality of these new solutions. Thus, if this accuracy and the environment learned are good, the ants have a good opportunity to find optimal solutions from several initial positions. In this way, in these results, we can see that the environments with the most accuracy and more percentage environment learned are the small terrains.

On the other hand, the low results for large environments can be by the number of nodes and the exploration capacity of the algorithm. In other words, when the algorithm has more nodes, it is more difficult for the ants to explore all the positions. Thus, the algorithm is more focused on the aim one to one and exploits it. In fact, this result is the expected; the algorithm should exploit the main objective, and the exploration of new positions are alternative results which could be useful, but no necessary.

Figure 5.5 shows the interaction between the different performance variables of the algorithm and the interaction Environment-Size.

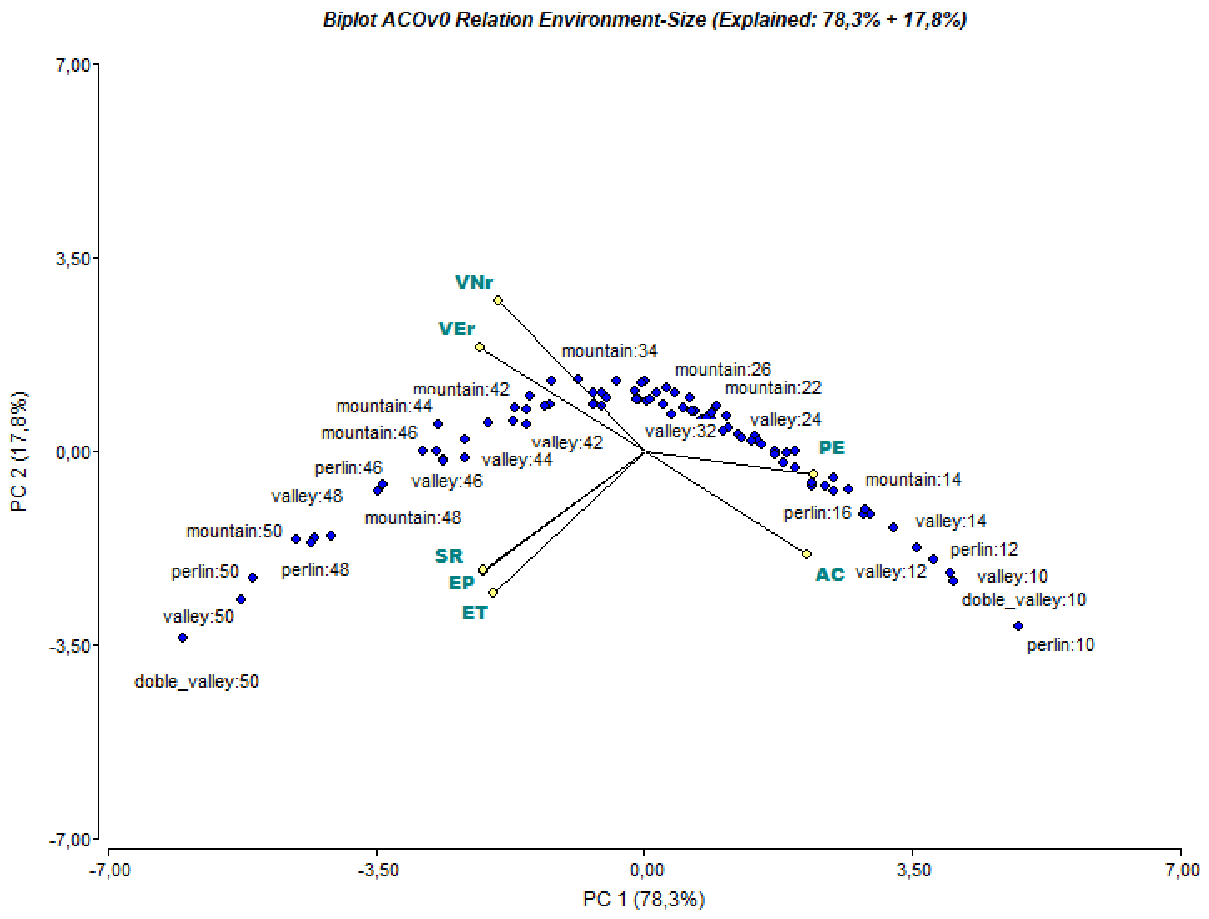


Figure 5.5: Biplot *ACOv0*. Explained: 78,3% + 17,8%

These results help us to confirm some of the previous results. In this graph, we can observe that the meshes with the most significant size are the meshes with more execution time (ET) and episodes (EP). Then, the small meshes are the meshes with more accuracy (AC) and more percentage environment learning (PE). Moreover, we can observe the interactions of other variables that measure the capacity of exploration and exploitation of the algorithm. In this sense, we can observe that the environments with more capacity of exploration are the middle and high size meshes. It is because these environments have high values of relative visited nodes (VNr) and edges (VEr), which indicates that in percentage, they visited more the environment than the small meshes. It is a contradictory result because they have a low percentage of the environment learned (PE). This inconsistency could be due that, although the large meshes can be more explored, the number of pheromones over these edges can disappear quickly.

On the other hand, another result of this graph is the stuck roads (SR). This measure indicates that large meshes have more stuck roads than others. These values are related to the execution time (ET) and episodes number (EP) because the algorithm will take longer to converge if there are many stuck conditions.

5.2 Evaluation of Global Interactions Performance

In the following experiments, we evaluated the performance of two versions of random walk, Section 4.2. Random Walk 1 (RW1) is the Proximity Random Walk, and Random Walk is the Semi-Proximity Random Walk. These versions were tested using several initial configurations, and the best results were found implementing the parameters of Table 5.2.

Version	ϕ_c	A	α	β	τ_0	ρ	δ	N_{rw}	M_{rw}	RN_{rw}	$\Delta\tau_{rw}$	ρ_{rw}
ACOv0	0.7	15	3	2	0	0.5	2	–	–	–	–	–
ACOv0 + RW1	0.7	15	3	2	0	0.5	–	20	2	–	0.2	0.25
ACOv0 + RW2	0.7	15	3	2	0	0.5	–	20	–	10	0.2	0.25

Table 5.2: Parameters of Random Walk Experiments

Each experiment was executed $n = 30$ times, and the mean, median, possible outliers of performance variables and biplot graphs were shown.

Figure 5.6 shows the execution time of the two different random walks, when they find twenty random route, or $N_{rw} = 20$.

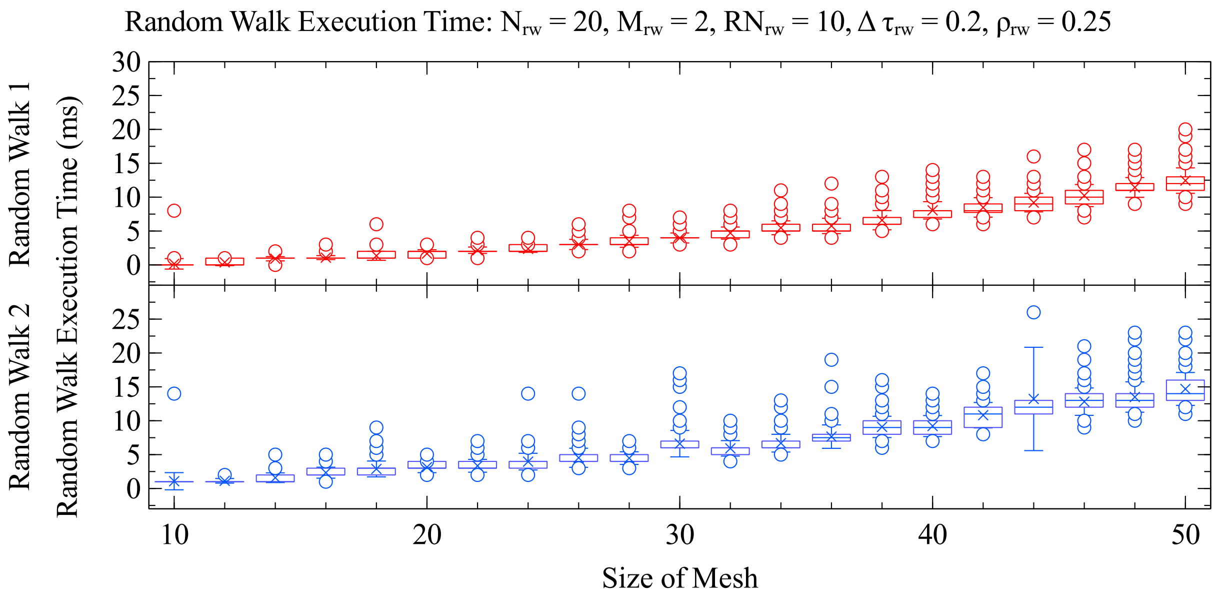


Figure 5.6: Random Walk Comparison Execution Time

According to these results, we can state that the random walks, in general, do not spend much time. The maximum time to find N_{rw} random routes does not exceed $12s$ to Random Walk 1 and $15s$ to Random Walk 2. Moreover, we could indicate that the Random Walk 1 is faster than Random Walk 2; however, the difference is not so marked. It is better to assume that the random walks spend similar execution times. These values are preprocessing times, and they were not considered in the posterior experiments. Some weird results are the time of Random Walk 2 with a size of 44. These results show several possible outliers; it could indicate that in the execution time, the computer could present some unexpected behaviors. In a more laborious study, we should remove these possible outliers and recalculate the times.

According to previous experiments of *ACOv0*, we observe that the results of different terrain types (Valley, Double Valley, Mountain and Perlin) do not have significant variability. For this reason, we decided to enclose all the results in graphs which summarize all their behaviors. In other words, in the following parts, we do not divide the graphs by each terrain type.

The following graph show the relation between size and execution time of the *ACOv0* using the different random walks, and without any random walk.

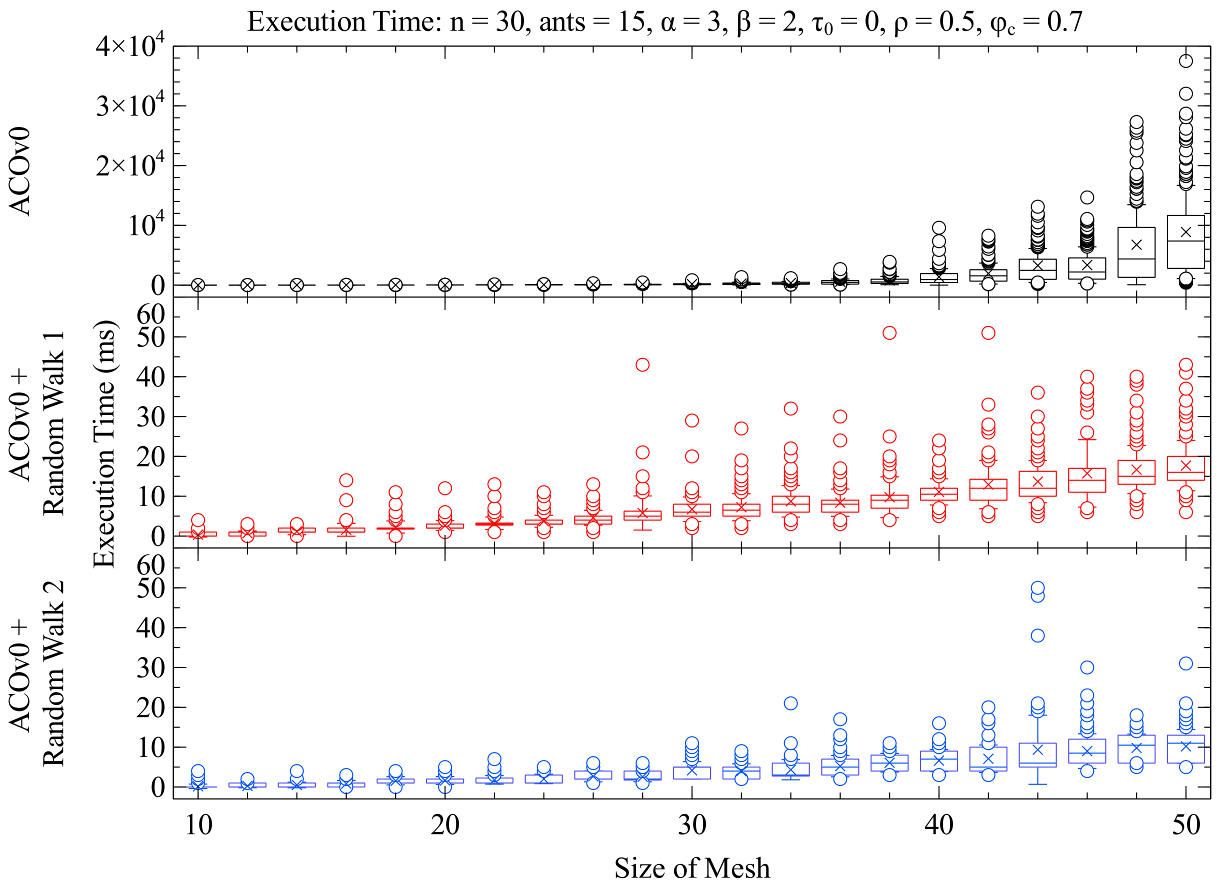


Figure 5.7: Random Walk and ACOv0 Comparison Execution Time

In Fig. 5.7, we can observe a reduction in time by the random walks in comparison with the *ACOv0* alone. This reduction is remarkable; that is to say, the mean and median time, with an environment of size 50, is reduced on a scale of $\frac{2000}{15} \approx 13$ times approximately. Another interesting result is that the Random Walk 2 seems to have better results than the Random Walk 1. It indicates that the initial pheromone values produced by Random Walk 2 are better than Random Walk 1. These results could be due to the accuracy of the random routes found in the preprocessing phase. In the experiments stage, we could observe that the random routes of RW2 get more times the accuracy of 100% than the RW1 routes. In this way, when we add the Random Walk 2 to the *ACOv0*, it seems coherent to get the convergence in less time. We presented this result in more detail, analyzing Fig. 5.9. Furthermore, we can observe that there are many outliers with high variability in the Random Walk 1 experiment than in the Random Walk 2 experiment. These could indicate that the algorithm is not enough stable using Random Walk 1.

Figure 5.8 shows the relation between size and number of episodes of the *ACOv0* using the random walk algorithms, and without any random walk.

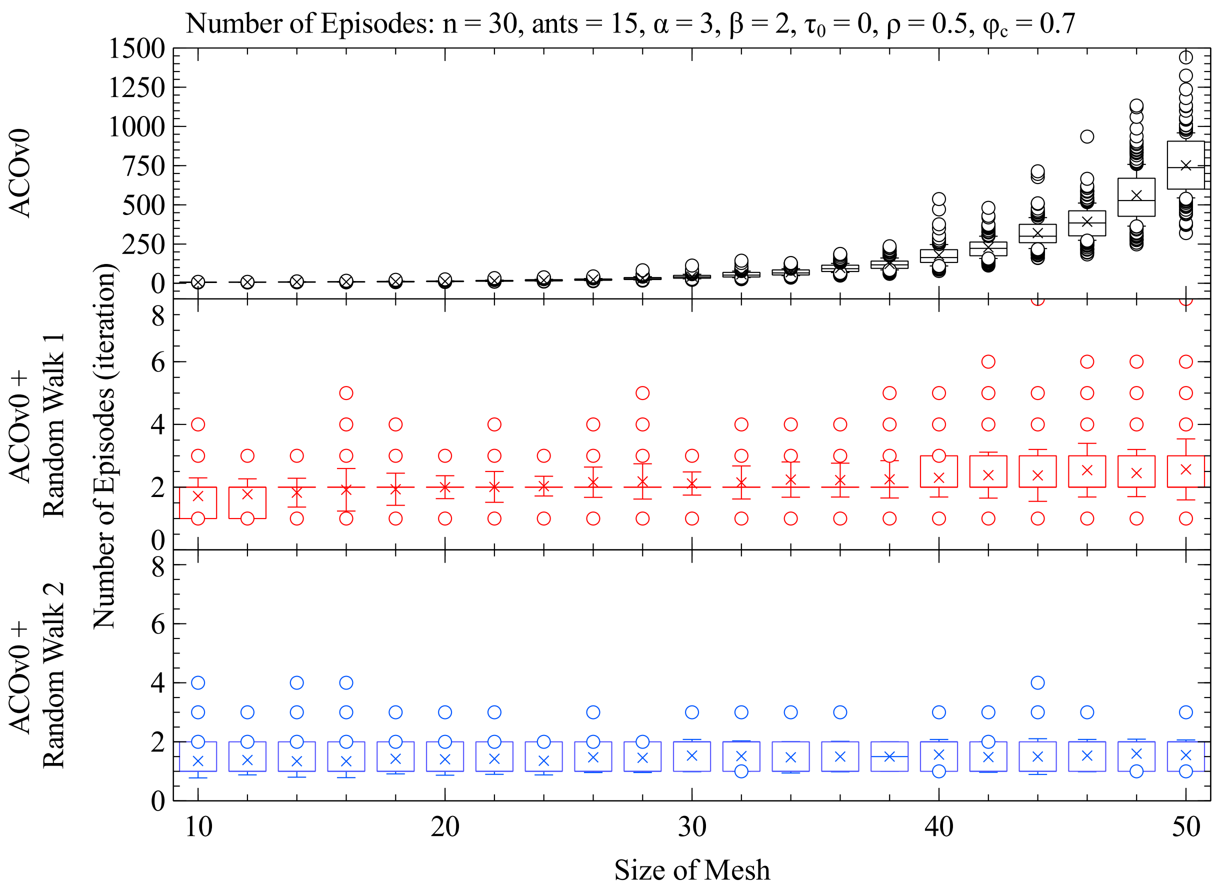


Figure 5.8: Random Walk and ACOv0 Comparison Episodes Number

These results indicate that the use of random walks reduces the number of episodes drastically. This reduction can be translated into a reduction of the computational complexity of the algorithm when this works with large environments. In fact, on average, the algorithm could have an approximated complexity of 2.5×15 for Random Walk 1 and 1.5×15 for Random Walk 2; it is to say, 37.5 iterations using Random Walk 1 and 22.5 iterations using Random Walk 2 approximately. Moreover, the increase in the episodes number is almost constant with all the sizes. In this way, we could expect to get similar complexities with high size environments which were not studied in this work. Finally, we could say that the Random Walk 2 has lesser episodes than Random Walk 1. Again, it could be due to the quality of the random routes found by each method.

Figure 5.9 shows a comparison between the size and accuracy of the *ACOv0* using different variants of random walks, and without any random walk.

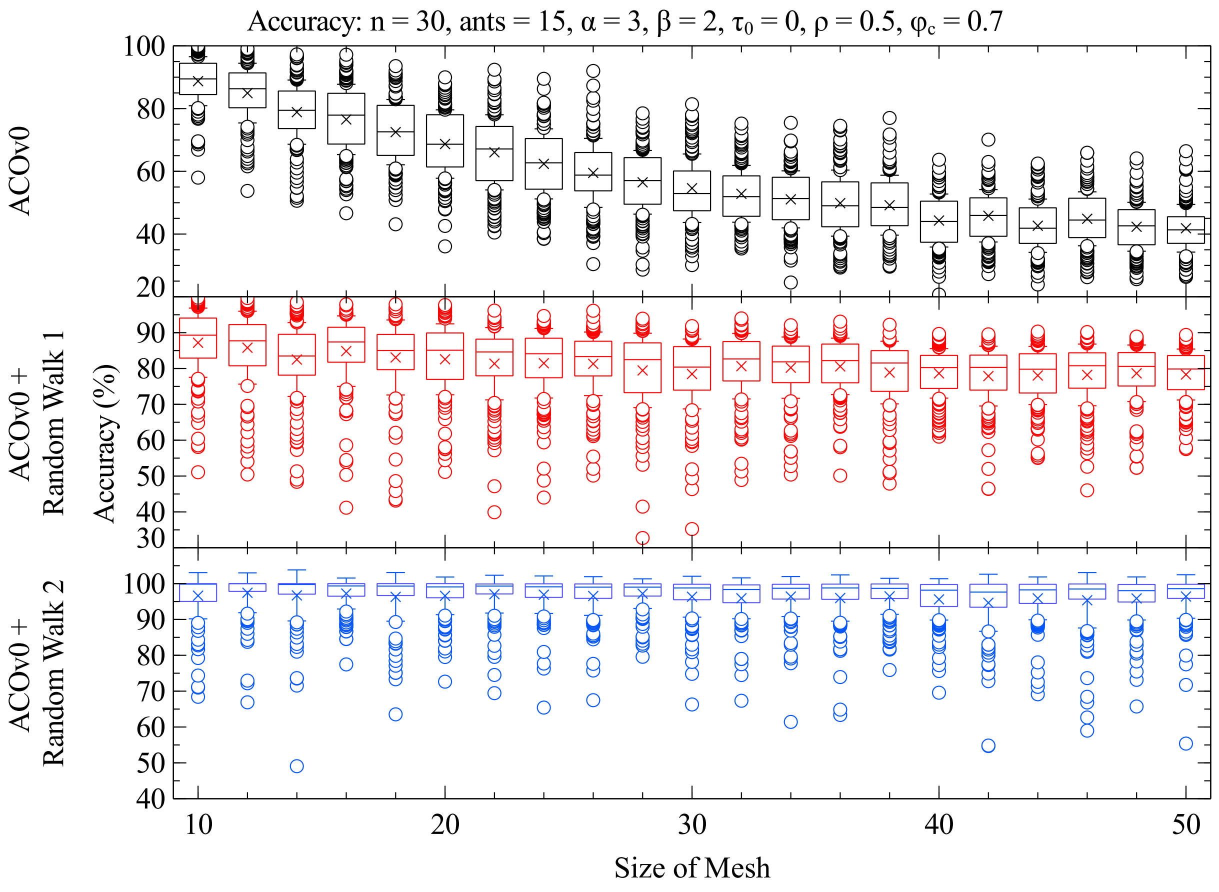


Figure 5.9: Random Walk and ACOv0 Comparison Accuracy

According to these results, the random walks improve the accuracy of the *ACOv0* with all sizes environments. Moreover, if we compare both random walks, we can see better results in Random Walk 2. As the previous results, Fig. 5.7, 5.8, this could be due to the precision of random routes found by the random walk algorithm. In the experimental phase, we found that the random routes of Random Walk 2 were closer to the optimal solution than the routes of Random Walk 1. Therefore, we expected to have better results of *ACOv0* using Random Walk 2. On mean and median, all the accuracies are higher than

80% in Random Walk 1, and greater than 95% in Random Walk 2. The outstanding results are the experiments with large sizes where the accuracy increase in a scale of $\frac{80}{45} \approx 1.78$ times approximately for Random Walk 1 and $\frac{95}{45} \approx 2.11$ times approximately for Random Walk 2.

Moreover, Random Walk 2 presents more constant behavior, which means that if we work with larger sizes, we could get similar results. These results confer an excellent quality of scalability to the algorithm. Despite all these excellent results, the presence of outliers is high. This issue is due to the algorithm randomness. We should expect it because we are working with a metaheuristic technique which does not guarantee to find an optimal solution, just feasible solutions.

Figure 5.10 shows the relation of percentage environment learned and the mesh size of *ACOv0* using different random walk techniques.

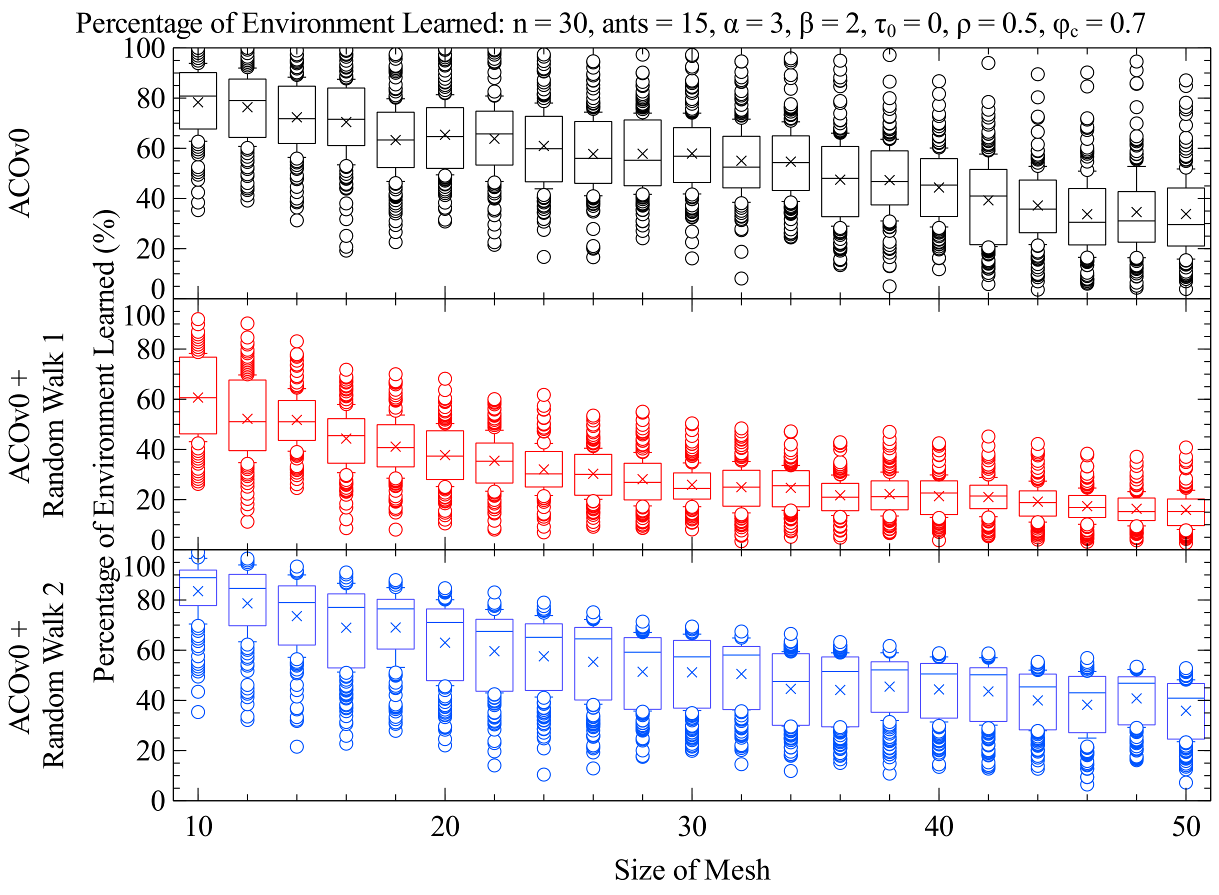


Figure 5.10: Random Walk and ACOv0 Comparison Percentage Environment Learned

According to the graph, the percentage of the environment learned presents better results using Random Walk 2 than using Random Walk 1. In fact, this percentage decreases faster using Random Walk 1, reaching below-average values. In the large sizes, using Random Walk 2, we can appreciate that the percentage is around 40% aproximately, whereas in the normal *ACOv0*, the values are over 30%, and in the RW1 + *ACOv0* the values are around 20%. It is a decrease of percentage in a factor of 0.75 for *ACOv0*, and 0.5 for RW1 + *ACOv0* approximately.

Figure 5.11 shows the interaction between the performance variables and the relation Version-Environment.

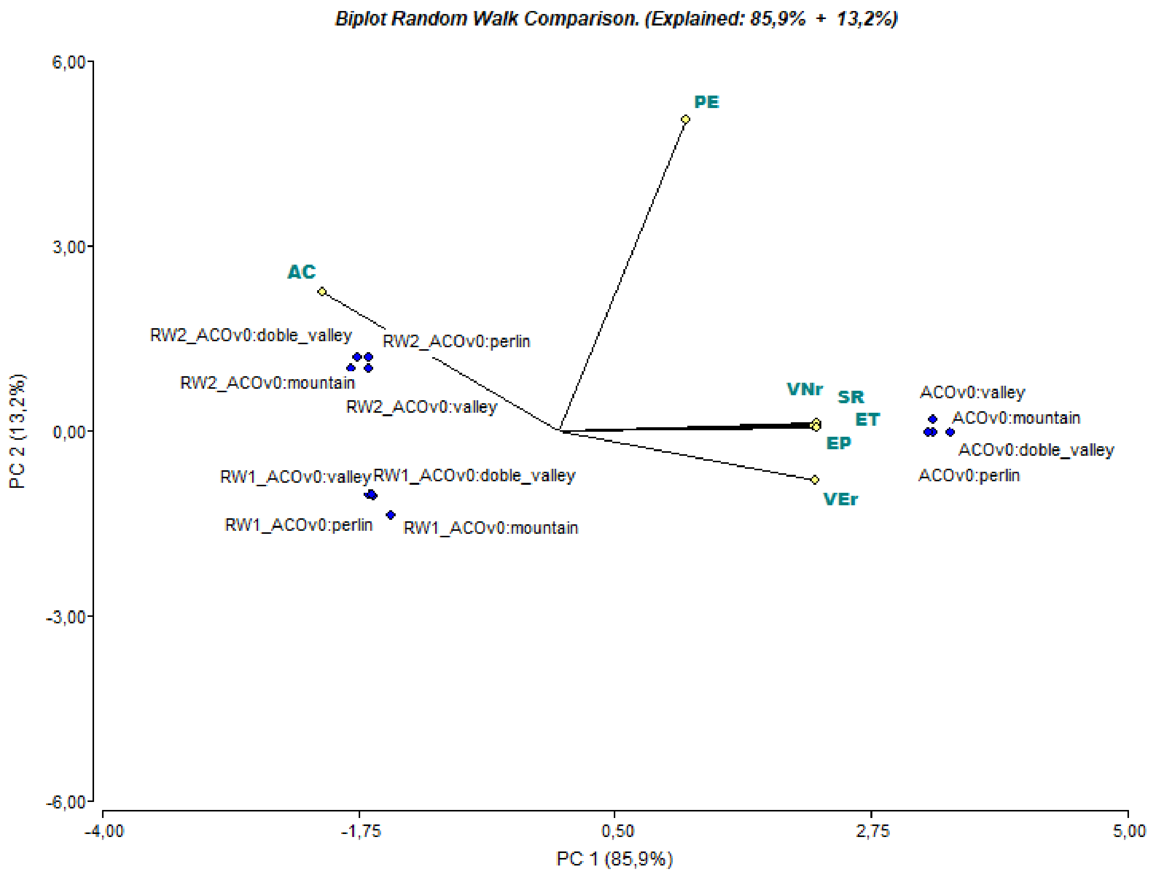


Figure 5.11: Biplot Random Walk Comparison. Explained: 85,9% + 13,2%

Figure 5.11 indicates the algorithm performance in terms of duration, exploration, exploitation, and precision. Regarding the algorithm duration, the axes ET, EP, and SR present high values in the *ACOv0* alone. These results help us to confirm the past results that state that *ACOv0* spend a long time. However, the *ACOv0* alone has the best values of VNr, VEr, and PE, which indicates a high capacity of exploration. This quality, therefore, could be causing the increase in time. On the other hand, we can state that the Random Walk 1 and 2 has a better capacity of exploitation due their low values of VNr and VEr. The accuracy has high values for the method which uses Random Walk 2. We already analyze these results in Fig. 5.14. As in previous results, we can observe that the variable PE has high values to the method *ACOv0* alone, and then to the Random Walk 2. According to these results, we selected the Random Walk 2 for the graphical simulation because it brings the best performance in most of the measures.

5.3 Evaluation of Local Interactions Performance

In the following experiments, we evaluated the performance of some variants of ACO with different local interaction techniques. These versions were tested using several initial configurations, and the best results were found implementing the following parameters:

Version	ϕ_c	A	α	β	τ_0	ρ	δ	RN_k	V_k
ACOv1	0.7	10	3	2	0	0.5	2	–	–
ACOv2	0.7	15	3	2	10	0.3	2	3	–
ACOv3	0.7	15	3	2	0	0.25	2	3	–
ACOv4	0.7	15	3	2	10	0.3	2	–	2
ACOv5	0.7	15	3	2	0	0.25	2	–	2

Table 5.3: Parameters of Local Interations Experiments

Each experiment was executed $n = 30$ times, and the mean, median, possible outliers of performance variables and biplot graphs were shown. Moreover, the aim of these variants is responding to obstacles apparition. To this end, for each size, we pre-calculate a mesh environment using the Kruskal's Algorithm, Fig. 4.16. In this way, all the environment sizes have a different amount of edges, according to the Table. 4.1.

Furthermore, according to experiments of *ACOv0* and Random Walks, we observe that the results of different terrain types do not have significant variability. In this sense, it is not necessary to plot the results of different terrains. Therefore, in the following parts, we presented graphs with summarized information of all the terrain types.

Figure 5.12 shows the relation between the execution time and the mesh size of the several local interactions. According to these results, the fastest versions are the *ACOv2* and *ACOv4*. On average, these versions has execution time close to $0ms$, with values of $100ms$ approximately. This result indicates that the reduction of pheromones decreases the possible stuck conditions faster than the other methods. The difference in time between versions 2 and 4 is small because both are based on the same techniques, just using a different selection of edges. Although we expected that version 3 and 5 would be better than version 1, the results show the opposite behavior. The next best version in the succession is the *ACOv1* which present results around to $1000ms$ approximately with large sizes. The methods *ACOv3* and *ACOv5* bring the worst results which indicates an execution time around $1500ms$ with large sizes. Besides, these versions present outliers with great variability; some values are around $8000ms$ to *ACOv3*, and *ACOv5*. These bad results could be due to the use of the additional heuristic value, necrophoresis value κ_{ij}. In this sense, it seems that this value is increasing the time, instead of reducing; internally, they could spend more time in access and store this value. In mean and median, with large sizes, the *ACOv2* and *ACOv4* is $\frac{1000}{100} = 10$ times faster than *ACOv1*, and $\frac{1500}{100} = 15$ times faster than *ACOv3* and *ACOv5*.

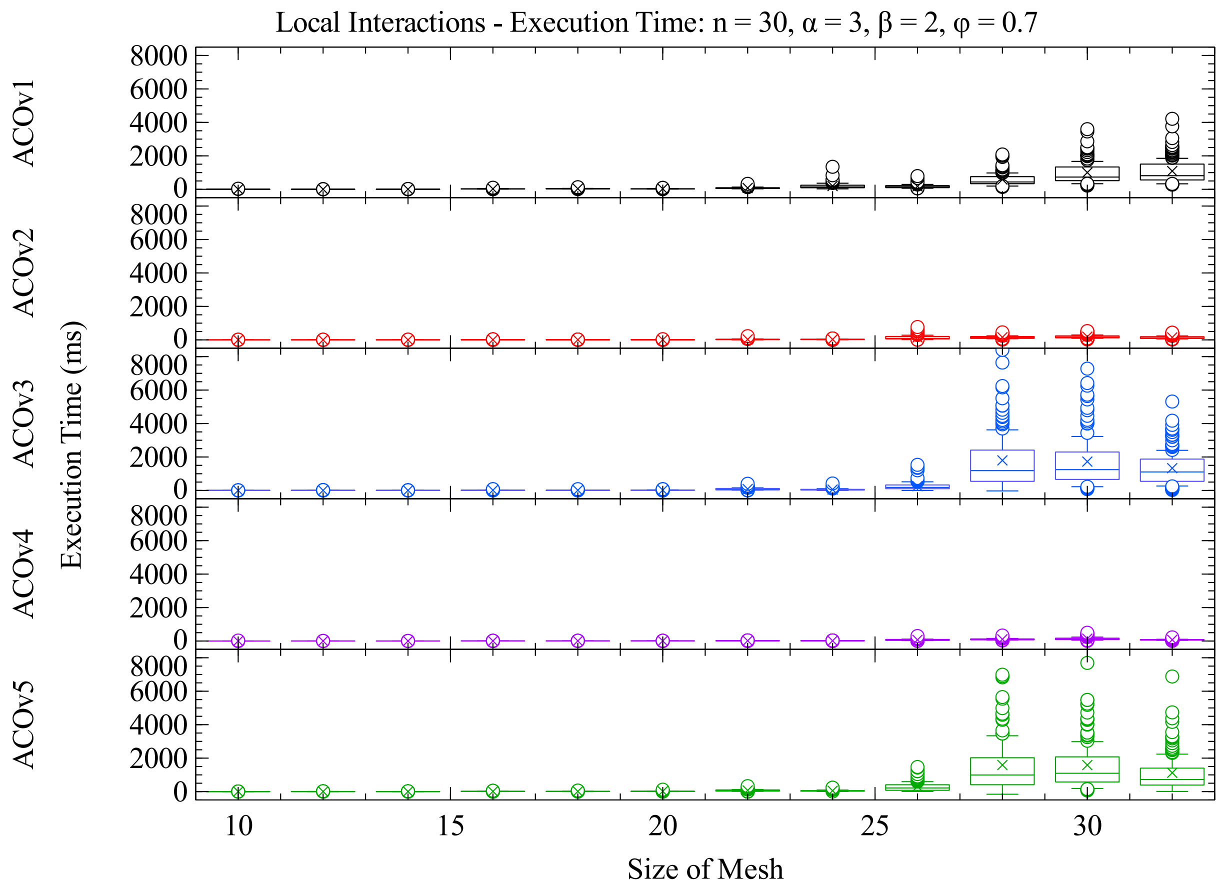


Figure 5.12: Local Interactions Comparison Execution Time

Moreover, in Fig. 5.12, we can observe that there is a smooth decrease between the times of mesh 30 and 32. This leads us to think that versions tend to reduce in time with larger environments. However, it is a wrong conclusion; the reason of this descent is the number and configuration of the edges in the maze environment of Table. 4.1. These results really tell us that it is easier to solve the random maze type of size 32 than the maze of size 30.

Figure 5.13 shows the relation between episodes number and sizes of the mesh of each local interaction version.

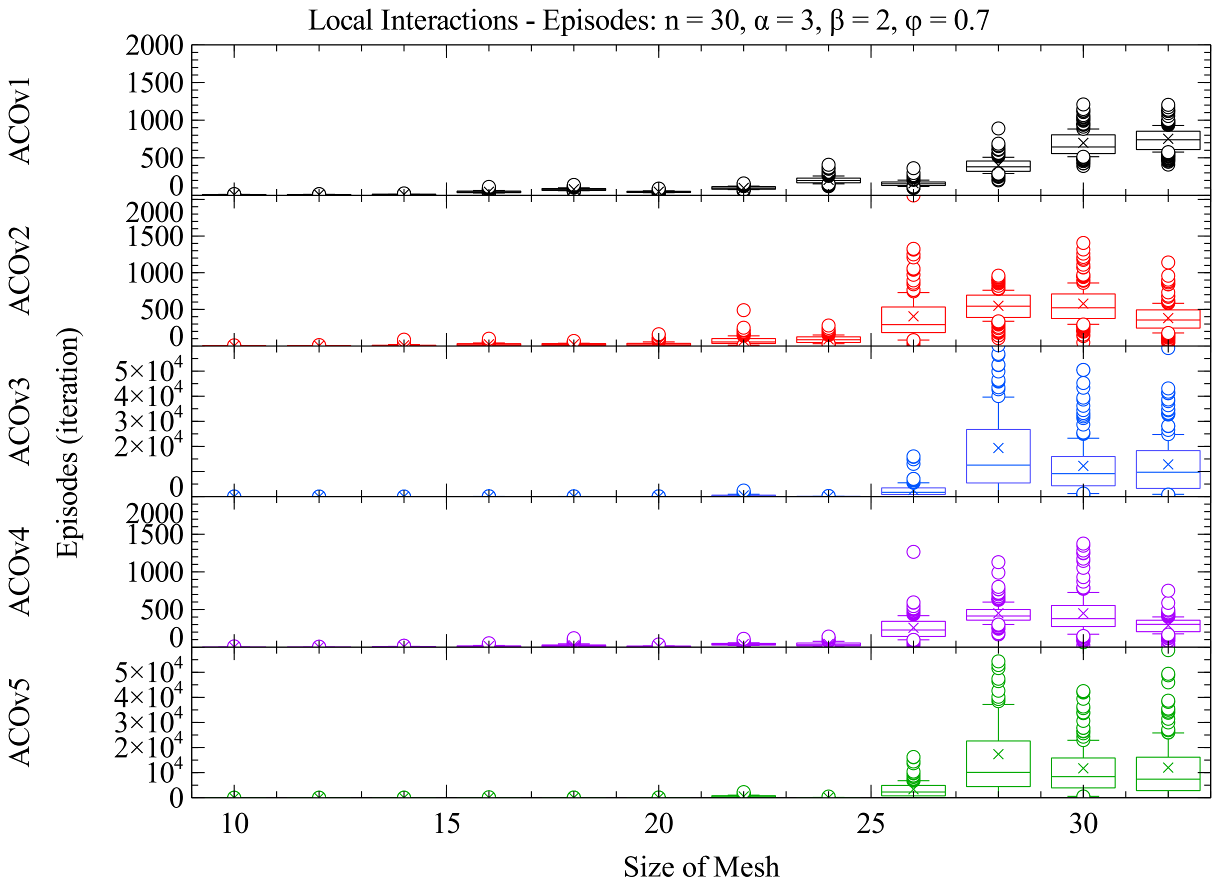


Figure 5.13: Local Interactions Comparison Episodes Number

These results indicate another notion of duration, which is related to the computational complexity of the algorithm. In this graph, the versions with best results are *ACOv2* and *ACOv4* with values around 500 episodes approximately with large mesh sizes. The values of version 4 have slightly better results than version 2; however, for practical aims, both results are considered similar. Later, we have the results of *ACOv1* with values around 600 episodes approximately with large mesh sizes. In this version, we can observe a more evident ascending behavior when we increment the size. This result tells us that using version 1, the preconditions of the maze environment do not guarantee a faster solution to the problem. Finally, we have the results of *ACOv3* and *ACOv5*, which are the worse values around 10000 episodes with large mesh sizes.

This measure gives a more accurate notion of the possible duration of the algorithm in a long term (with larger sizes). In this sense, with large sizes, the *ACOv2* and *ACOv4* are $\frac{600}{500} = 1.2$ times faster than *ACOv1*, and $\frac{10000}{500} = 20$ faster than *ACOv3*, and *ACOv5*.

Fig 5.14 shows the relation between the accuracy and the mesh sizes of the different local interactions.

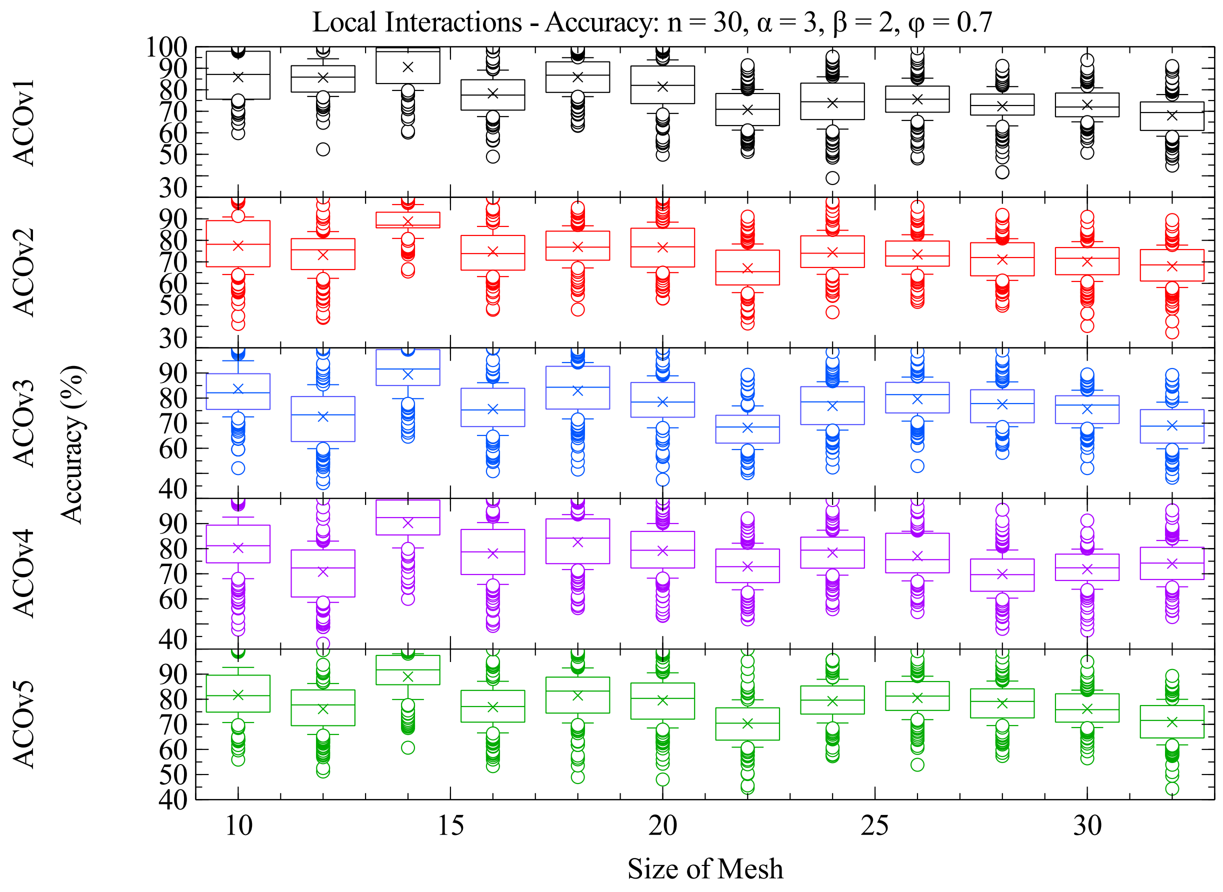


Figure 5.14: Local Interactions Comparison Accuracy

According to these results, the first evident result is the high variability of the solutions with different mesh sizes. There are some low values with small sizes and some high values with large sizes. Besides, the values vary widely even in the same size experiments. For example, using the environment size 24, the algorithm presents results around 40% approximately, which is a deficient value. This variability and low accuracies are an effect of the initial conditions of the environment due to the maze environments. In the beginning, we prepared these environments, adding some obstacles (deleting some edges) through a Kruskal' algorithm, Table. 4.1. For this reason, the number and locations of the remaining edges are different in each case. Despite experimenting with different random mazes, the algorithms should be capable of overcoming these changes and find accurate solutions in all the cases. In this sense, we can state that the proposed local interactions are unstable. Despite this instability, version 1 presents the best results with values up 70% in most of the cases.

Furthermore, the bad results in accuracy counter the great results of duration. In this sense, we can not consider a method that is so fast but return worst results. Therefore, in order to get the Final ACO version, we priorities precision over the duration. Despite these results, we did other analysis to reinforce this conclusion in Fig. 5.15 and 5.16.

Figure 5.15 shows the relation between the percentage of environment learned and mesh sizes.

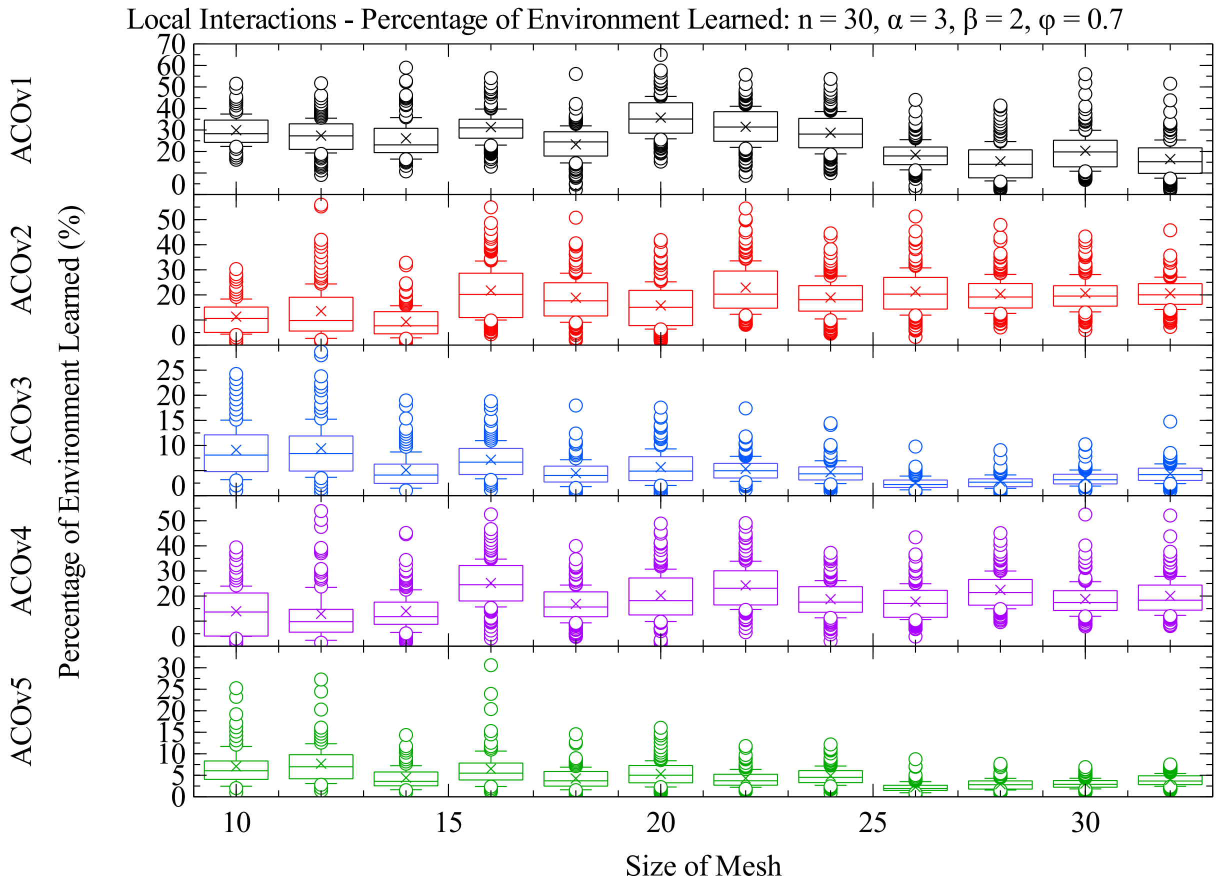


Figure 5.15: Local Interactions Comparison Percentage Environment Learned

The results presented in Fig. 5.15 also show a high variability. We again can observe uncertain behavior in small and large sizes. However, in this case, the results are lower than the accuracy results. In general, the percentage reaches values of 5% in the worst cases, which indicates low exploration capacity. This worse feature could explain the low accuracy of solutions, Fig 5.14. By the bad exploration, the algorithm does not visit many places of the environment, which provokes that the ants only can move in a reduced space. Probably, this space does not contain optimal or close to optimal solutions. In this way, we can not expect to get precise solutions.

Despite these bad results, we could analyze the best version. The version with the most percentage of the environment learned is *ACOv1*, which presents results around 15% approximately. The next best version are *ACOv2* and *ACOv4*, and finally *ACOv3* and *ACOv5*. The *ACOv1* appears to have the best solutions because its associated method allows that the ants tend to visit more places, in comparison with the other techniques.

The biplot of Fig. 5.16, shows the interaction of the variables and the relation Version-Terrain.

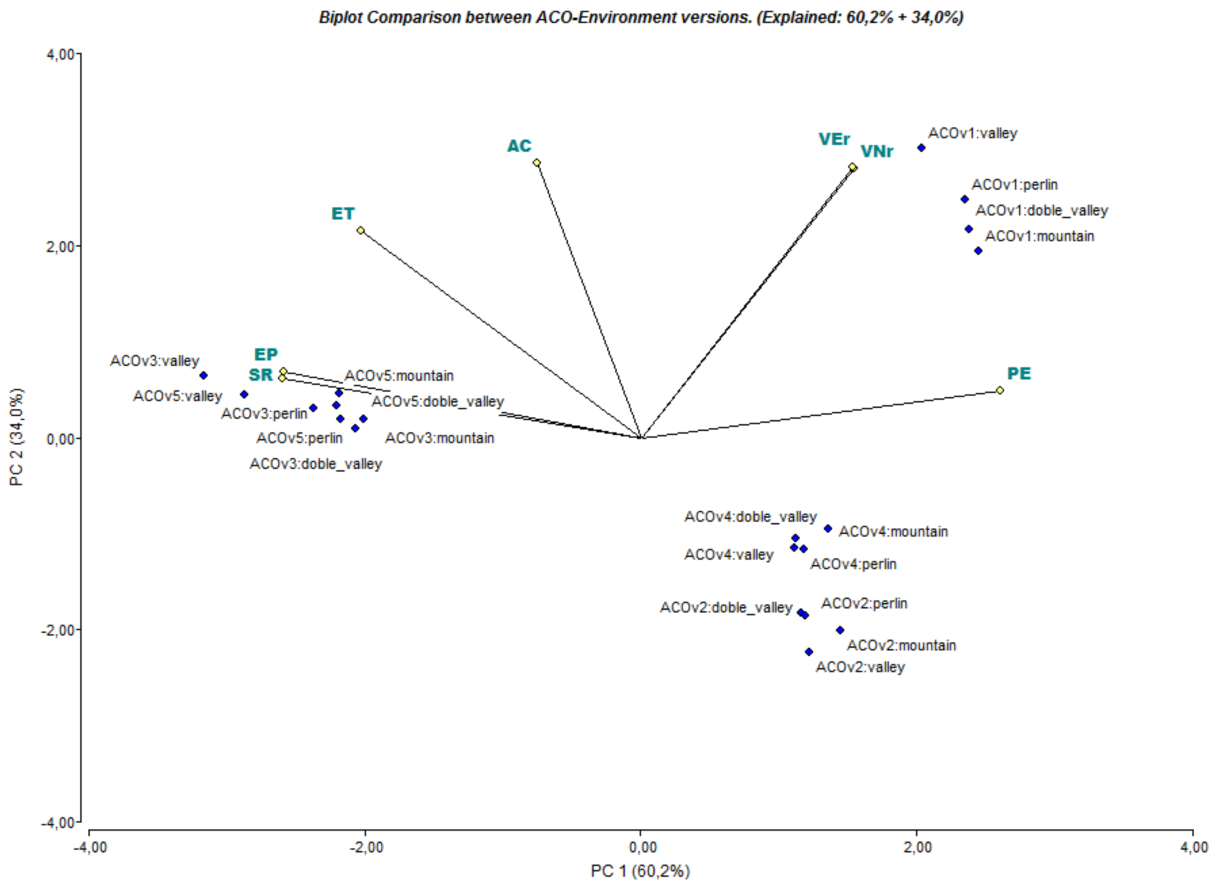


Figure 5.16: Biplot Local Interactions Comparison. Explained: 60,2% + 34,0%

In this graph, we can analyze at the same time, the characteristics of duration, exploration, exploitation, and accuracy. Regarding the duration feature, the results confirm the results of Fig. 5.12 and 5.13. Thus, the method with high values of EP, ET, and SR are *ACOv3* and *ACOv5*. Then, the fast methods are the *ACOv1*, *ACOv2* and *ACOv4*, which presents similar values. In terms of exploration and exploitation, the version *ACOv1* has the best values of VEr, VNr, and PE. This result could be due to the strategy of forbidden nodes. Using this dynamic, the ants must visit more places, and therefore more nodes and edges. Finally, analyzing the accuracy, we found that the worse results are in the methods *ACOv2* and *ACOv4*, whereas the best results are in *ACOv1*. These results could be due to the versions 2 and 4 applies a reduction of pheromones over the principal deposit. This reduction could affect the algorithm convergence and avoid to get proper solutions.

Moreover, another result that we can observe is the association between variables. There is overlapping with the variables VEr and VNr. This result indicates that these measures behave in the same way, and could be replaced by a unique variable. The same happens with the variables SR and EP, which are almost overlapped. However, this overlap could be due to a condition of cause-effect because a high number of stuck roads can cause a high number of episodes.The precision, duration, exploration, and exploitation results go along with the previous results. Nevertheless, by studying Fig. 5.14 and 5.15, we can conclude that, in general, the local interaction proposals are unstable.

5.4 Evaluation of Several Methods

In the following experiments, we evaluated the performance of four different methods for path planning solving. In this part, we decided to use, for the final version, the *Random Walk 2* and *ACOv0* because this combination presents better performance results in terms of duration and precision. Besides, the exploitation is higher that the exploration in this final version. In this sense, we priorities duration, then precision, later exploitation, and exploration finally. Furthermore, this final version can deal successfully with environments with obstacles, as long as the number of obstacles does not increase drastically (partially free environment).

These experiments were tested using several initial configurations, and the best results were found implementing the following parameters in some of the methods:

ACO Final	$\phi_c = 0.7$	$A = 15$	$\alpha = 3$	$\beta = 2$	$\tau_0 = 0$	$\rho = 0.5$
Genetic	$\phi_c = 0.5$	$n_i = 20$	$\rho_s = 0.4$	$\rho_m = 0.8$	$\omega_e = 0.7$	$\omega_n = 0.8$
Dijkstra	`No use parameters`					
A*	$h(v_c) = \nu(v_c, v_f)$ `Preprocessed heuristic - Proximity`					

Table 5.4: Several Methods Initial Parameters or Preprocessed Information

Each experiment was executed $n = 30$ times, and the mean, median, and possible outliers of performance variables were shown. Furthermore, the following experiments consisted of two stages. In the first stage, all the methods were executed in an environment without any obstacle. The route found in this phase was called **Route 1**. In the second stage, some obstacles were added in specific places trying to cover the previous optimal routes. The route found in this phase was called **Route 2**.

Figure 5.17 shows the relation between the accuracy and mesh size of different methods of path planning.

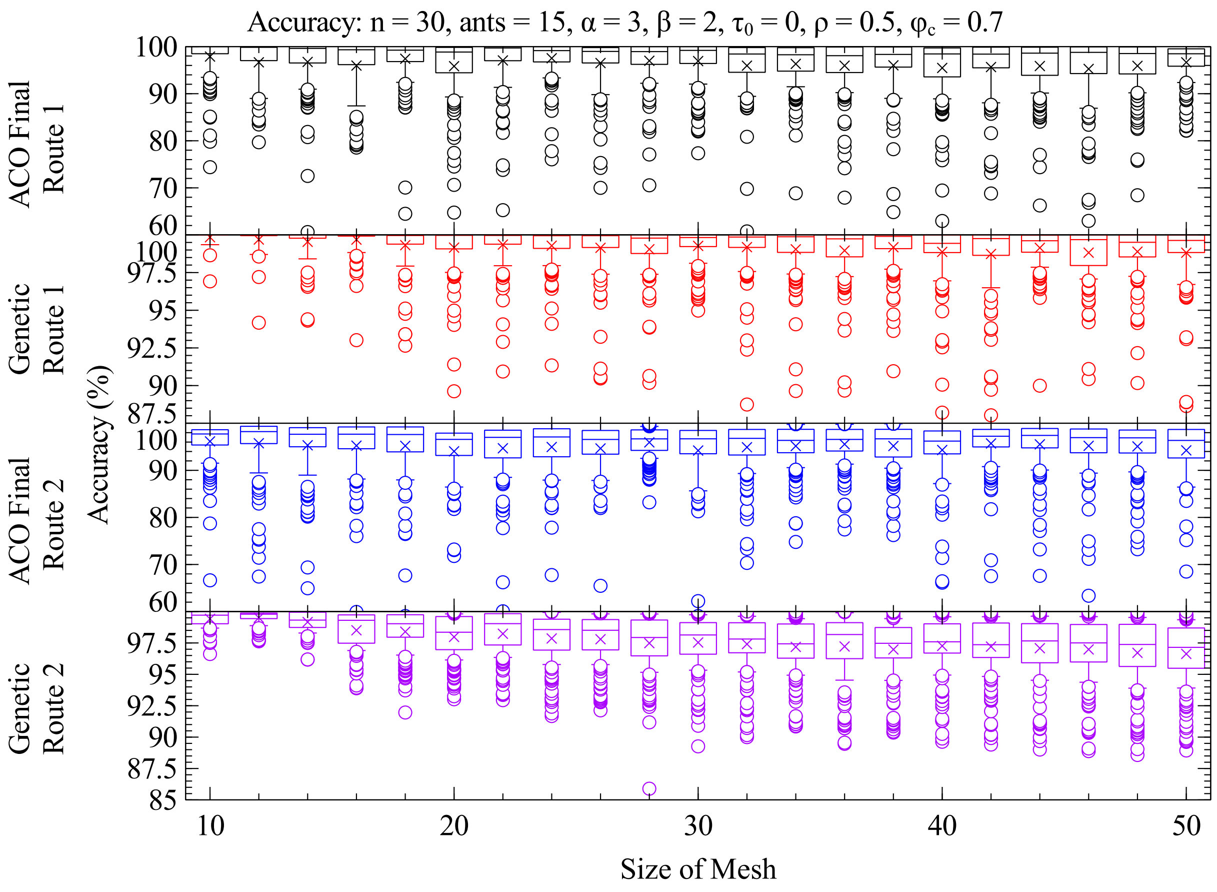


Figure 5.17: Accuracy Comparison of Different Methods of Path Planning

In this part, it is essential to consider that the methods *A** and *Dijkstra* have an accuracy of 100%. For this reason, we do not graph this measure. A result of more interest is the comparison with the Genetic algorithm, which, as well as the ACO algorithm, does not always get the optimal solution. On the one hand, in Figure 5.17, the Genetic algorithm has an accuracy upon 85% in Route 1 and upon 87.5% in Route 2, and a mean and median upon 95% approximately. On the other hand, we can observe a high variability in the ACO solutions with accuracy measurements upon 60% in Route 1 and Route 2, and mean and median upon the 95%. According to these results, we can state that the general accuracy of the Genetic algorithm is better than the ACO algorithm in the Route 1 and Route 2, in terms of variability. However, both methods conserve a mean and median over the 95%, because they get solutions close to the optimal solution several times. Besides, if we analyze the Route 2 of the Genetic Algorithm, we can detect a marked decrease behavior as the size increases. This result could indicate future worst accuracy with some largest sizes. The ACO algorithm seems not to present this marked behavior; its mean and median are almost constant with accuracy over 95% approximately.

In Figure 5.18, we can see the relation between the size of mesh and execution time of the first finding route of several path planning methods, Route 1.

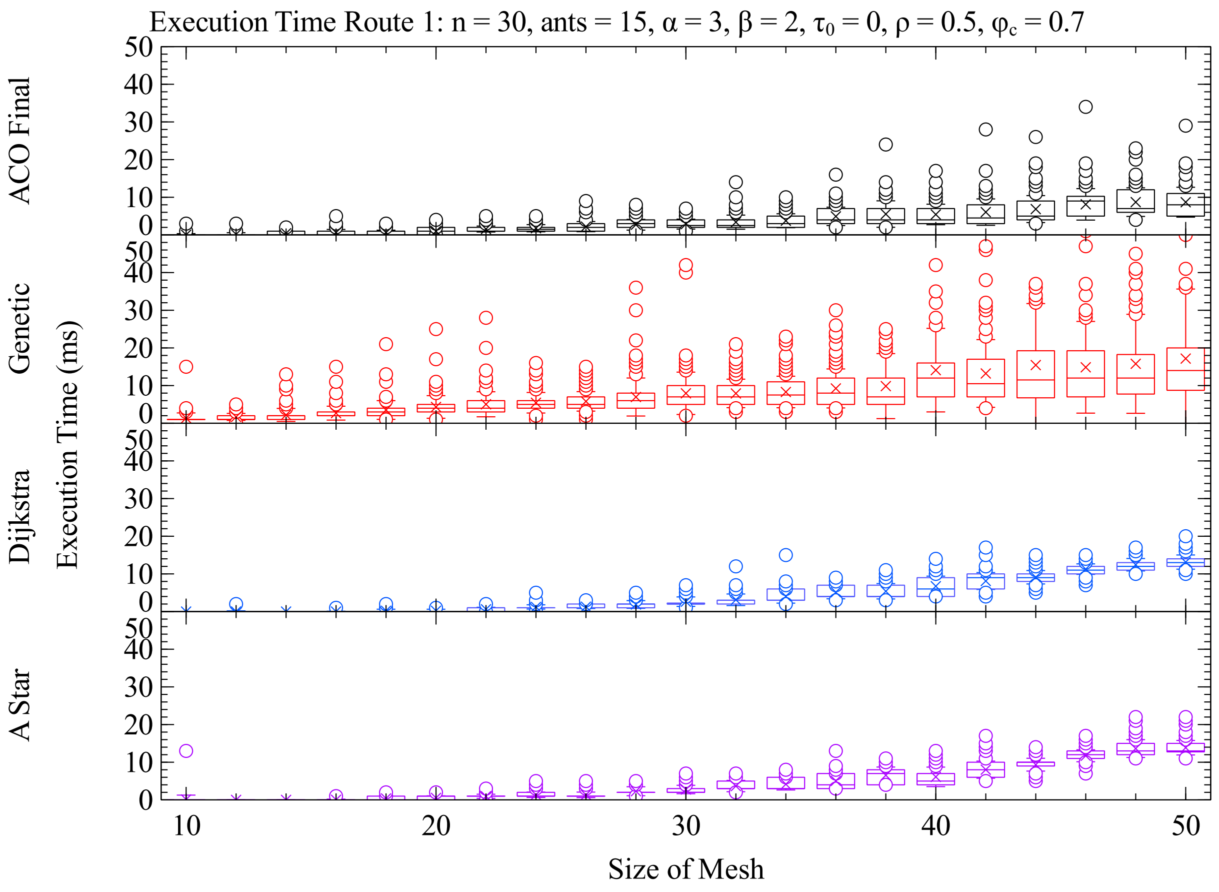


Figure 5.18: Route 1. Execution Time Comparison of Different Methods of Path Planning

According to these results, the execution time of the ACO algorithm is under 7.5 *ms* in mean and median with the largest sizes. Moreover, outliers values do not exceed 30 *ms*. Then, in the Genetic algorithm, we can observe an execution time, in mean and median, around 18 *ms* with the largest sizes. However, these values are more scattered, reaching in some cases values of 60 *ms* or 70 *ms*. The results of the Dijkstra algorithm show a marked increasing behavior, reaching values around 20 *ms*, and in mean and median under 12.5 *ms* with the largest sizes. It is expected behavior due to the nature of the algorithm, which analyzed all the nodes, Chapter 3. As the Dijkstra algorithm, the A* algorithm also presents a marked increasing behavior, and the values of mean, median, and outliers are only a little less than the Dijkstra values. On mean, and median, the values are around 11.5 *ms* with largest sizes, and the outliers get values of 22.5 in some cases. It could indicate that the selected heuristic, proximity Equation 4.1, does not contribute substantially to a faster finding of the solution. Based on these estimated values, we can state that the ACO algorithm has the best performance of execution time. In fact, on mean and median, the ACO solutions are faster in a factor of $\frac{18}{7.5} = 2.4$ than Genetic algorithm, $\frac{12.5}{7.5} = 1.67$ than Dijkstra algorithm, and $\frac{11.5}{7.5} = 1.53$ than the A* algorithm. This outstanding result is an advantage over the other algorithms and compensates the variability of the accuracy results, Fig. 5.17.

Figure 5.19 shows the relation between the size of the mesh and the execution time spent to find a new route with the presence of obstacles in the environment, Route 2.

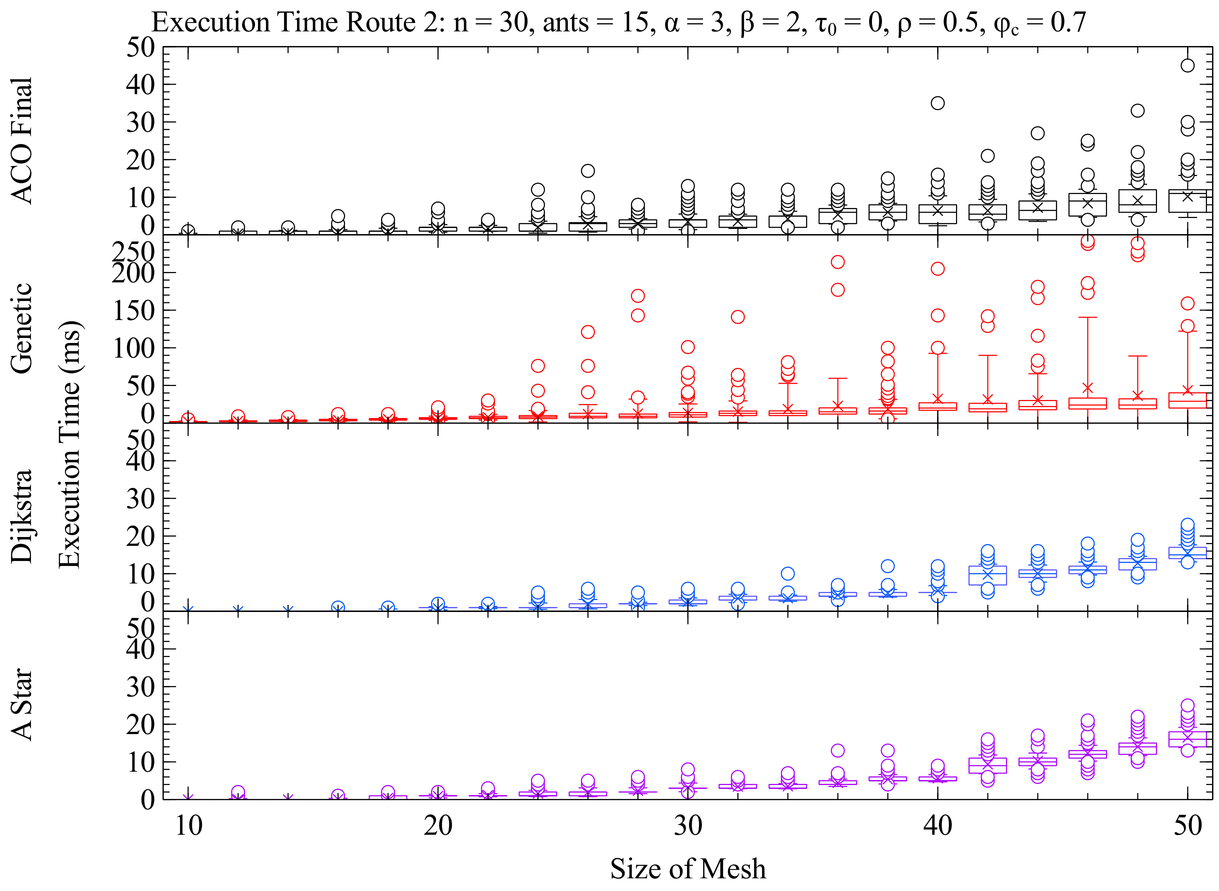


Figure 5.19: Route 2. Rerouting Time Comparison of Different Methods of Path Planning.

According to these results, the ACO algorithm finds a new route in an execution time around 10 *ms* using the largest sizes. Besides, some of its outliers reach values of only around to 30 *ms*. Then, the Genetic algorithm has mean and median execution times around 25 *ms* approximately using the largest mesh sizes and has outliers up to 250 *ms*. The Dijkstra and A* algorithms reach similar values of mean and median around 15 *ms* with the largest sizes and maximum outliers around 25 *ms*. All these results state that the ACO algorithm finds a new route in less time in comparison with the other methods. In this sense, the ACO solutions is faster in a factor of $\frac{25}{10} = 2.5$ than the Genetic algorithm, and $\frac{15}{10} = 1.5$ than the Dijkstra and A* algorithm.

Despite these excellent results, the time of this new execution is not less than the first route found. The ACO algorithm can store information about pheromones and use it to find a new route. Therefore, we expected that these pheromones would speed up the searching process. However, the performance values are almost similar to previous experiments, Fig 5.18. These values could indicate that future results with larger sizes can be similar. Nevertheless, the used mesh sizes return low execution times (milliseconds), which could be an inadequate extrapolation of the behavior with larger sizes. Thus, we can state that it is an unfinished result because we can not show with great certainty that the pheromones speed up the rerouting process or not.

Some conclusions could carry us to interpret that this comparison is not well-founded because we are comparing path planning methods of type one to one (ACO, Genetic) with type one to many (Dijkstra, A*). However, we mentioned in previous experiments, Section 4.2, that the nature of the ACO algorithm allows us to get an alternative result, the Percentage of Environment Learned. This result partially solves the path planning one to many. In that way, we could define this ACO algorithm as a one to one method and quasi one to many method. This definition tells us that it is not unreasonable to perform this comparison.

5.5 Qualitative Evaluation of Graphic Simulation

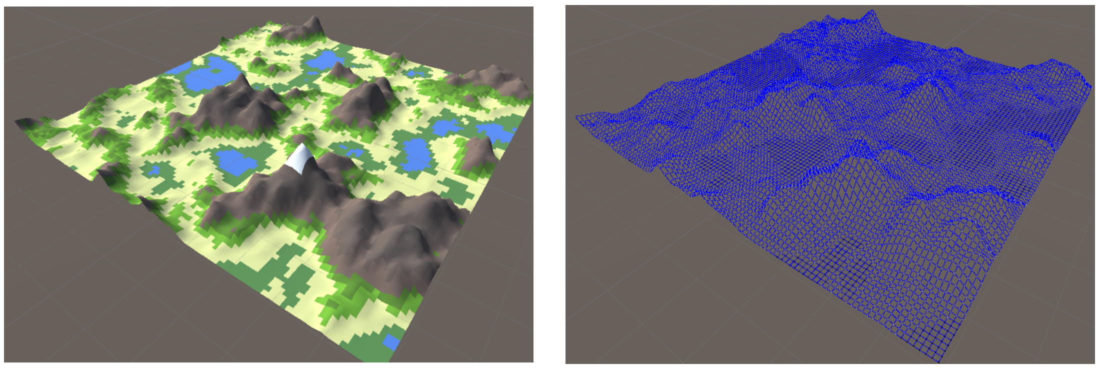
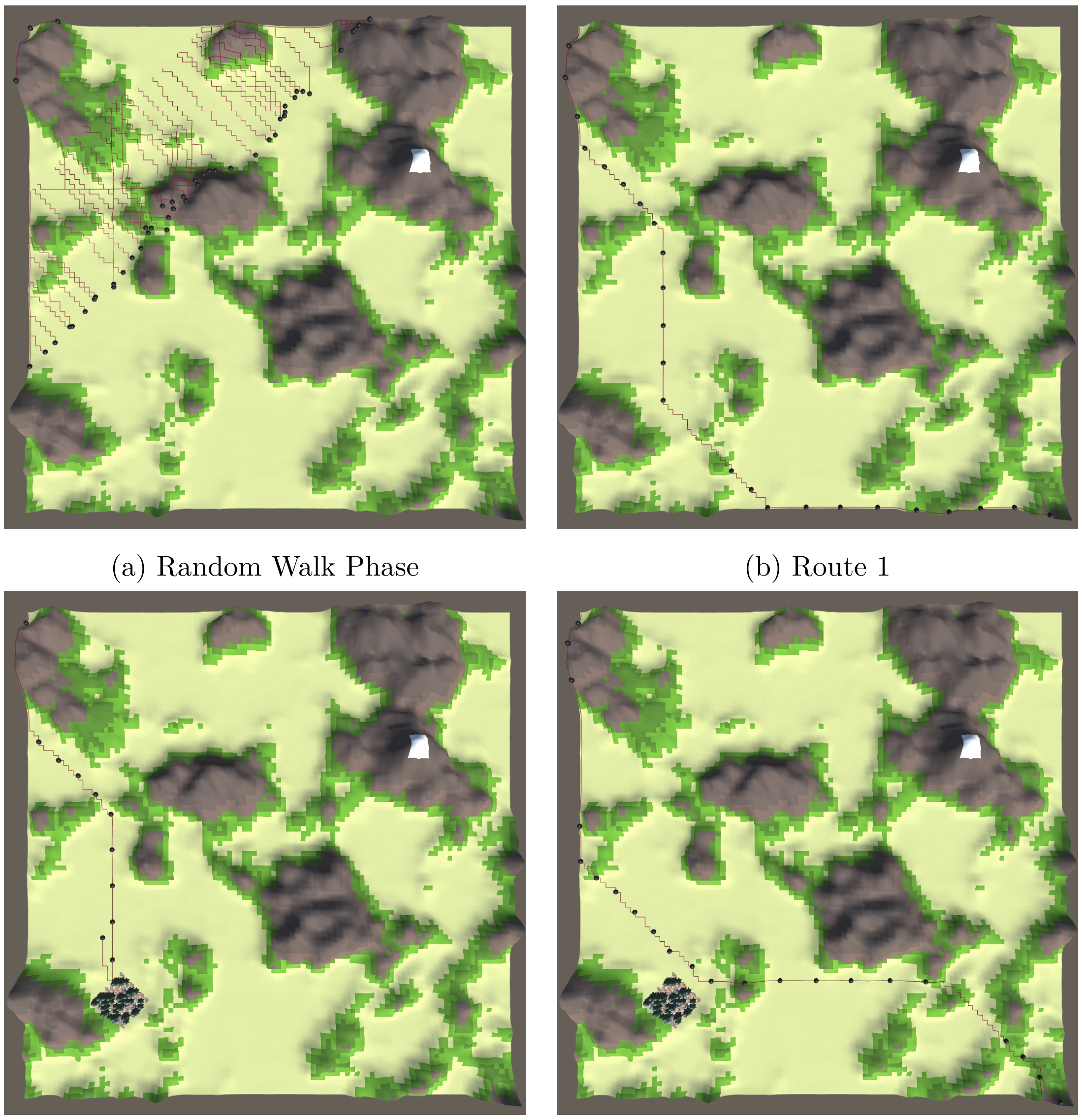

(a) Environment Mesh (b) Graph Mesh

Figure 5.20: Environment Configuration in Graphical Simulation

In this section, we use the best performance variant, Random Walk 2, and ACOv0. In this sense, the graphic simulation implements three-dimensional mesh based on a landscape terrain, Fig. 5.20. We used some concatenation of Perlin noise based on [47] to develop the environment mesh. Figure 5.21 shows the adaptability behavior of the algorithm in an environment without obstacles. In Figure 5.21a, the ants perform the random walk together. Then, in Figure 5.21b, the ants converge to a specific path. Later, in Figure 5.21c, an obstacle is added, and the ants get stuck. Finally, in Figure 5.21d, the ants achieve to find another route. The proposed algorithm reaches this behavior because it can use the information of pheromones for rerouting the ants. In other words, it does not need a new execution; it responses in real-time to the addition of obstacles.



(a) Random Walk Phase (b) Route 1

(c) Obstacle Addition and Killed Ants (d) Rerouting Phase or Route 2.

Figure 5.21: Graphic Simulation Screenshots

5.6 General Results

In this chapter, we incrementally presented the results. All these results showed a great performance when the algorithms work with small mesh sizes. Therefore, most of the analysis focused on the result with larger sizes. In the first experiments, we study the performance of the basic algorithm *ACOv0*. Although the algorithm achieves to converge, the solutions with large sizes present low-performance results, Fig. 5.1, 5.2, 5.3, 5.4.

In order to improve the *ACOv0* results, we introduce two different global interactions, know as *Random Walks*. The random walks substantially improved the performance of the *ACOv0* in terms of duration, and precision. In these experiments, we found that the Random Walk 2 results in better performance values of execution time, episodes, and accuracy. Besides, according to the biplot results, Fig. 5.11, the Random Walks experiments have a high capacity of exploitation; in other words, they have low values of VNr, VEr, and PE. Table 5.5, we can observe the predominance of each variant. In this sense, we subsequently selected the combination RW2 + *ACOv0* for comparing with other path planning methods.

Feature	**Variables**	**High**	**Medium**	**Low**
Duration	ET, EP, SR	ACOv0	RW1 + ACOv0	RW2 + ACOv0
Precision	AC	RW2 + ACOv0	RW1 + ACOv0	ACOv0
	PE	RW2 + ACOv0	ACOv0	RW1 + ACOv0
Exploration	ER, VNr, VEr	ACOv0	RW1 + ACOv0	RW2 + ACOv0
Exploitation	ER, VNr, VEr	RW2 + ACOv0	RW1 + ACOv0	ACOv0

Table 5.5: Best Result of Global Interactions Experiments

In the next experiments, we decided to add obstacles in the form of a maze. Thus, we find that the random walks are not enough to find the solutions because the ants tend to get stuck several times and do not converge to a solution. For these reasons, we implemented some local interactions which control the stuck conditions. In this part, we developed five versions of these local interactions for comparing and get the best. Thus, we found that *ACOv1* produces the best accuracy values. The *ACOv2* and *ACOv4* methods have the best values of execution time and episodes. Besides, according to Fig. 5.16, the *ACOv1* presents a great exploration ability, the *ACOv5* and *ACOv3* last longer times, and the*ACOv2* and *ACOv4* have the worst accuracy. In Table 5.6, we can observe the predominance of each version. Nevertheless, in general, all the local interactions present a bad accuracy and percentage of environment learned with high levels of variability. These results indicate that the local interactions are unstable techniques which can not solve in the same way different maze environments. For these reasons, we avoided to use these methods to the final version.

Feature	**Variables**	**High**	**Medium**	**Low**
Duration	ET, EP, SR	ACOv3 - ACOv5	ACOv1	ACOv2 - ACOv4
Precision	AC	ACOv1	ACOv3 - ACOv5	ACOv2 - ACOv4
	PE	ACOv1	ACOv2 - ACOv4	ACOv3 - ACOv5
Exploration	ER, VNr, VEr	ACOv1	ACOv2 - ACOv4	ACOv3 - ACOv5
Exploitation	ER, VNr, VEr	ACOv3 - ACOv5	ACOv2 - ACOv4	ACOv1

Table 5.6: Best Result of Local Interactions Experiments

Then we compared different methods of path planing with our ACO Final proposal. In these experiments, we decided to work with *Random Walk 2* and *ACOv0*, which presents better performance values. Moreover, this combination can deal with environments with obstacles, as long as they do not increase drastically. We did not use any local interactions because, despite the local interactions achieve to converge, they do not result in accurate performance values. In this way, it is recommendable to improve these techniques in future works. In this comparison, we found that our final proposal reaches an acceptable accuracy over 95% in the first and second routing stages. However, other methods have minor variability in its accuracy. Despite these results, our proposal compensates the variable accuracy with low execution times in the first and second routing stages. In Table 5.7, we can observe the predominance of each method.

Feature	**Variables**	**High**	**Medium High**	**Medium Low**	**Low**
Duration	ET, RT	Dijkstra	A*	Genetic	ACO Final
Precision	AC	Genetic	ACO Final	–	–

Table 5.7: Best Result for Several Methods Comparison

Finally, in the simulation stage, we found some qualitative results, Fig. 5.21, which indicates the adaptability behavior of our ACO algorithm. Besides, these results suggest an idea of the possible applications of this work in real-life based simulations.

Chapter 6

Conclusions

6.1 Conclusion

In this thesis, we proposed a path planning model based on the Ant Colony Optimization algorithm. For designing this model, we implemented a basic ACO algorithm that converged using the proposed data sets. Nevertheless, its performance measures were deficient. For this reason, we included global and local interactions that improve the performance of the basic algorithm. Then, we compared the best variant of the model with other methods for having an idea of its usability. Finally, we performed a graphic simulation based on the final model. It is essential to mention that several preliminary results of this work were presented in the *6th IEEE Latin American Conference on Computational Intelligence LA-CCI* with the paper titled *"Three Dimensional Adaptive Path Planning Simulation Based On Ant Colony Optimization Algorithm"* (Waiting for Publication).

The global interaction experiments showed that the variant: *Random Walk 2* plus *ACOv0* produces the best performance results and can deal with environments with obstacles, as long as they do not increase drastically. Conversely, in local interactions experiments, we found that although the algorithms converge, the solutions had low performance. Therefore, we did not include this interaction in the ACO final version. By comparing our version with other methods, the results suggested that the Final ACO version has an acceptable accuracy with values above 95%. Nevertheless, our proposal presented more variable accuracy results comparing with other methods. This loss of accuracy was compensated with the low execution time, with values around $10ms$ in the first and second stages of routing.

On the other hand, we concluded that the usefulness of the pheromone trails is limited for our problem size since the execution with more nodes and edges generates solutions with more variability. Another conclusion was the generation of an additional result known as the percentage of the environment learned. By the nature of the ACO algorithm, the ants could find other solutions in the environment, solving partially the path planning problem many to one. These solutions were useful results but not necessary because the aim of this project was solving the path planning problem one to one.

Finally, the qualitative results of the graphic simulation showed the adequate adaptive behavior of our proposal. The ants were entirely able to find a new route, using the pheromones, when the obstacles were introduced in execution time.

6.2 Recommendations

In this part, we list some recommendations based on the problems and limitations found in our model proposal.

- Implement an adequate representation of the environment with the use of graphs and data structures. For example, the algorithm could use an adjacency matrix or list to store the heuristic and pheromones values. Besides, instead of graphs, we could use another environment representation based on the application. For example, we can use cells decomposition for robot motion applications [13] [1].
- Establish an incremental style for designing an algorithm based on Ant Colony Optimization. In this way, it is possible to analyze the possible errors and limitations and deal with them as we add new features.
- Prove different heuristic measures for the movement choice technique, Equation 4.2, and establish a useful heuristic based on the problem.
- Establish an adequate frequency of evaporation and reinforcement, as well as the $\Delta\tau$ value according to the problem circumstances.
- In the design stage, we recommend implementing a rank of values for each parameter. Then, we suggest to prove them for selecting the parameters with the best results.
- The ACO algorithm is quite influenced by the correct implementation of its principal features: the movement choice, reinforcement, evaporation, and the initial parameters, as well as the local and global interactions in our proposal. For this reason, we recommend use sizes around $[30, 35]$ in the design stage, neither far too large nor too small. The last interval is because the experiments with small mesh could imply bad predictions in the long term, but the larger sizes spend much time. We should work with a manipulable environment for establishing adequate methods for the main features of the algorithm.
- For the implementation of the several variants of our proposal, we recommend to develop the algorithm with separated functions. This programming technique will allow reusing parts of the code in the other variants.
- In the experiments with maze environments, we recommend searching an easy way to see how it is the composition of the maze for analyzing and detect possible stuck conditions.
- In the methods comparison stage, we recommend using some already implemented algorithm instead of self implementations. These could give us a better idea of the performance of our proposal.

6.3 Future Work

In this part, we propose some future research and implementation, which could be taken into account in future works.

- **High-Performance Computing with more Massive Data Sets.** The performance values calculated using large environments are results of interest because it can give us a prediction of results on a grand scale. In the present work, this scalability quality has been partially studied because we have worked with several mesh sizes, and some variants have shown unfinished results yet. For this reason, we propose to develop more experiments with larger mesh sizes to study the scalability performance of the algorithm. In this way, it would be necessary to implement the algorithm in a programming language that supports the use of high-performance computers such as C, C++, or Python.

- **Parallelization of the Ant Colony Optimization Algorithm.** The present project proposed some variants which implement sequential algorithms where the ants perform a walking one by one. However, the basic idea of the Ant Colony Optimization algorithm is substantially parallel; therefore, the algorithm should be easily paralleled. This parallelization can give us some advantages over other algorithms in terms of execution time. For this reason, another propose is designing some variants of paralleled algorithms that can deal with the path planning problem.

- **Proposal or Review of Methods for Solving Maze Environments.** The present proposal shows some advantages over other algorithms. However, when we tried to analyzed environments with many obstacles, the algorithm produces solutions with low performance, particularly in large maze environments. For this reason, it is necessary to seek other techniques to deal with the maze environments or review the proposed techniques to design a better implementation.

- **Comparison with other Adaptive Methods.** The present project performs a comparison between the ACO, Genetic, Dijkstra, and A* algorithm. This analysis indicates that the ACO algorithm has acceptable performance results. However, its adaptability feature has not been wholly studied because we only do a qualitative representation in the graphic simulation. For this reason, a new comparison with other methods could take place. This comparison should help to detect the level of adaptability of our proposal. We found some steps that follow this research direction in [48].

- **Research a better Implementation of Necrophoresis Value κ_{ij}.** The proposed idea of simulating the *Necrophoresis* natural behavior of the ants is promising. The ideas discussed in this project could benefit from the integration of the idea of Necrophoresis. We recommend designing another technique that can deal with this behavior in a better way.

Bibliography

[1] G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, "Path planning," in *Wheeled Mobile Robotics.* Elsevier, 2017, pp. 161–206. [Online]. Available: https://doi.org/10.1016/b978-0-12-804204-5.00004-4

[2] Yanrong Hu and S. Yang, "A knowledge based genetic algorithm for path planning of a mobile robot," no. April, pp. 4350–4355 Vol.5, 2004.

[3] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,, 2016.

[4] L. Juntao, D. Tingting, L. Yuanyuan, and H. Yan, "Study on robot path collision avoidance planning based on the improved ant colony algorithm," in *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC).* IEEE, Aug. 2016. [Online]. Available: https://doi.org/10.1109/ihmsc.2016.215

[5] A. Tuncer and M. Yildirim, "Dynamic path planning of mobile robots with improved genetic algorithm," *Computers & Electrical Engineering*, vol. 38, no. 6, pp. 1564–1572, Nov. 2012. [Online]. Available: https://doi.org/10.1016/j.compeleceng.2012.06.016

[6] I. Engedy and G. Horvath, "Artificial neural network based mobile robot navigation," in *2009 IEEE International Symposium on Intelligent Signal Processing.* IEEE, Aug. 2009. [Online]. Available: https://doi.org/10.1109/wisp.2009.5286557

[7] K. Makantasis, M. Kontorinaki, and I. Nikolos, "A Deep Reinforcement Learning Based Driving Policy for Autonomous Road Vehicles." pp. 1–17, 2019.

[8] C. Eyckelhof, M. Dorigo, G. Caro Di, M. Snoek, and M. Sampels, "Ant systems for a dynamic tsp - ants caught in a traffic jam," 2002, pp. 88–99, imported from HMI.

[9] M. E. Yuksel, "Agent-based evacuation modeling with multiple exits using NeuroEvolution of augmenting topologies," *Advanced Engineering Informatics*, vol. 35, pp. 30–55, Jan. 2018. [Online]. Available: https://doi.org/10.1016/j.aei.2017.11.003

[10] M. Liu, F. Zhang, Y. Ma, H. R. Pota, and W. Shen, "Evacuation path optimization based on quantum ant colony algorithm," *Advanced Engineering Informatics*, vol. 30, no. 3, pp. 259–267, Aug. 2016. [Online]. Available: https://doi.org/10.1016/j.aei.2016.04.005

[11] C. W. Reynolds, “Steering behaviors for autonomous characters,” in *Game developers conference*, vol. 1999. Citeseer, 1999, pp. 763–782.

[12] W. A. Enezi and C. Verbrugge, “Offline grid-based coverage path planning for guards in games,” 2020.

[13] P. Yap, “Grid-based path-finding,” in *Conference of the Canadian Society for Computational Studies of Intelligence.* Springer, 2002, pp. 44–55.

[14] E. Alba and F. Chicano, “Acohg: Dealing with huge graphs,” in *Proceedings of the 9th annual conference on Genetic and evolutionary computation.* ACM, 2007, pp. 10–17.

[15] S. Hong, “Lazier graph-based path planning for autonomous navigation,” Aug. 21 2018, uS Patent 10,054,447.

[16] B. Dugarjav, S.-G. Lee, D. Kim, J. H. Kim, and N. Y. Chong, “Scan matching online cell decomposition for coverage path planning in an unknown environment,” *International journal of precision engineering and manufacturing*, vol. 14, no. 9, pp. 1551–1558, 2013.

[17] F. Lingelbach, “On probabilistic completeness of probabilistic cell decomposition,” *CoRR*, vol. abs/1507.03727, 2015. [Online]. Available: http://arxiv.org/abs/1507.03727

[18] M. Candeloro, A. M. Lekkas, and A. J. Sørensen, “A voronoi-diagram-based dynamic path-planning system for underactuated marine vessels,” *Control Engineering Practice*, vol. 61, pp. 41–54, 2017.

[19] T. Petković, D. Puljiz, I. Marković, and B. Hein, “Human Intention Estimation based on Hidden Markov Model Motion Validation for Safe Flexible Robotized Warehouses,” *arXiv e-prints*, p. arXiv:1811.08269, Nov 2018.

[20] P. Oettershagen, F. Achermann, B. Müller, D. Schneider, and R. Siegwart, “Towards fully environment-aware uavs: Real-time path planning with online 3d wind field prediction in complex terrain,” *CoRR*, vol. abs/1712.03608, 2017. [Online]. Available: http://arxiv.org/abs/1712.03608

[21] Y.-b. Chen, G.-c. Luo, Y.-s. Mei, J.-q. Yu, and X.-l. Su, “Uav path planning using artificial potential field method updated by optimal control theory,” *International Journal of Systems Science*, vol. 47, no. 6, pp. 1407–1420, 2016.

[22] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

[23] H. Zhai, M. Egerstedt, and H. Zhou, “Path Planning in Unknown Environments Using Optimal Transport Theory,” *arXiv e-prints*, p. arXiv:1909.11235, Sep 2019.

[24] M. Dorigo and G. Di Caro, “Ant colony optimization: a new meta-heuristic,” in *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, vol. 2. IEEE, 1999, pp. 1470–1477.

[25] M. Glabowski, B. Musznicki, P. Nowak, and P. Zwierzykowski, "Shortest path problem solving based on ant colony optimization metaheuristic," *Image Processing & Communications*, vol. 17, no. 1-2, pp. 7–17, 2012.

[26] M. Dorigo, "Optimization, learning and natural algorithms," *PhD Thesis, Politecnico di Milano*, 1992.

[27] D. Angus, "Solving a unique Shortest Path problem using ant colony optimisation," *Communicated by T. Baeck*, no. January, pp. 1–26.

[28] S. Zhang and R. Zhang, "Radio Map Based 3D Path Planning for Cellular-Connected UAV," *arXiv e-prints*, p. arXiv:1912.00021, Nov 2019.

[29] Y. Li, T. Dong, M. Bikdash, and Y.-D. Song, "Path planning for unmanned vehicles using ant colony optimization on a dynamic voronoi diagram." in *IC-AI*, 2005, pp. 716–721.

[30] Y. Rasekhipour, A. Khajepour, S.-K. Chen, and B. Litkouhi, "A potential field-based model predictive path-planning controller for autonomous road vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1255–1267, 2016.

[31] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia, "PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-based Planning," *arXiv e-prints*, p. arXiv:1710.03937, Oct 2017.

[32] D. Marco and T. Stützle, "Ant colony optimization the mit press," *Cambridge, Massachusetts*, 2004.

[33] J. Trejos-Zelaya, L. E. Amaya-Briceño, A. Jiménez-Romero, A. Murillo-Fernández, E. Piza-Volio, and M. Villalobos-Arias, "Clustering Binary Data by Application of Combinatorial Optimization Heuristics," *arXiv e-prints*, p. arXiv:2001.01809, Jan 2019.

[34] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas, "Data mining with an ant colony optimization algorithm," *IEEE transactions on evolutionary computation*, vol. 6, no. 4, pp. 321–332, 2002.

[35] P. Wang, Y. Zhang, and D. Yan, "An improved self-adaptive ant colony algorithm based on genetic strategy for the traveling salesman problem." Author(s), 2018. [Online]. Available: https://doi.org/10.1063/1.5039120

[36] T. Stiitzle and H. Hoos, "The max-min ant system and local search for the traveling salesman problem," in *Proceedings of IEEE international conference on evolutionary computation*, 1997, pp. 309–314.

[37] T. Stützle and H. H. Hoos, "Max–min ant system," *Future generation computer systems*, vol. 16, no. 8, pp. 889–914, 2000.

[38] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[39] A. Levitin, *Introduction To Design And Analysis Of Algorithms, 2/E.* Pearson Education India, 2008.

[40] R. E. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 189–211, Mar. 1990. [Online]. Available: https://doi.org/10.1016/0004-3702(90)90054-4

[41] V. Bulitko and G. Lee, "Learning in Real-Time Search: A Unifying Framework," *arXiv e-prints*, p. arXiv:1110.4076, Sep 2011.

[42] D. Sudholt and C. Thyssen, "Running time analysis of ant colony optimization for shortest path problems," *Journal of Discrete Algorithms*, vol. 10, pp. 165–180, Jan. 2012. [Online]. Available: https://doi.org/10.1016/j.jda.2011.06.002

[43] G. O. López-Riquelme, M. L. Fanjul-Moles *et al.*, "The funeral ways of social insects. social strategies for corpse disposal," *Trends Entomol*, vol. 9, pp. 71–129, 2013.

[44] Ton Roosendaal and Blender Fundation, "Ant landscape manual." [Online]. Available: https://docs.blender.org/manual/fr/dev/addons/add_mesh/ant_landscape.html

[45] J. Buck, *Mazes for programmers: code your own twisty little passages.* Pragmatic Bookshelf, 2015.

[46] A. C. Rencher and W. F. Christensen, *Methods of Multivariate Analysis.* John Wiley & Sons, Inc., Jul. 2012. [Online]. Available: https://doi.org/10.1002/9781118391686

[47] S. Lague. Procedural landmass generation. Youtube. [Online]. Available: https://www.youtube.com/watch?v=wbpMiKiSKm8&list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3

[48] C. Horoba and D. Sudholt, "Ant colony optimization for stochastic shortest path problems," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO 10.* ACM Press, 2010. [Online]. Available: https://doi.org/10.1145/1830483.1830750

Appendices

Appendix 1. Implementation Details

Basic Loop of ACO Algorithm

```
public void ExecuteACO(ref MeshEnvironment env)
{
    //Create a list of ants
    List<FinalAntv0> colony = CreateColony();

    //No init, use the stored values // In some version, we use it
    //env.InitPheromones(tau_0);

    //Variables to control evaporation and convergence
    bool almost_one_route = false;
    bool converge = false;

    //Time Variable
    var watch = System.Diagnostics.Stopwatch.StartNew();

    while (!converge)
    {
        foreach (var ant in colony)
        {
            //Try to find a possible route
            List<int> route = ant.FindRoute(ref env);

            //If the ant finds a route
            if (route.Count != 0)
            {

                //Reinforce the route
                Reinforcement(ref env, ref route);

                //Almost one route per episode
                almost_one_route = true;

                //Check and set if the best cost changed
                CheckAndSetBestCost(ref env, ref route);

            }

        }

        //Reset the visited nodes
        env.ResetNodes();

        if (almost_one_route)
        {
            //Evaporation on the environment
            Evaporation(ref env);
            almost_one_route = false;
        }

        converge = CheckConvergence(ref colony, ref env);

        //Episodes
        episode_counter++;

    }

    //Execution Time
    watch.Stop();
    execution_time += watch.ElapsedMilliseconds;
    colony_ants = colony;

}
```

Reinforcement

```
//------------------------------------------------------------------
public void DeltaTau(ref MeshEnvironment env, ref List<int> route)
{
    //Calculate the Delta Tau according to the best cost so far.
    if (best_cost != Double.MaxValue)
    {
        Double current_cost = ExtraTools.GetCost(ref route, ref env);
        delta_tau = best_cost / current_cost;
    }

}
//------------------------------------------------------------------
public void Reinforcement(ref MeshEnvironment env, ref List<int> route
    )
{
    for (int i = 0; i < route.Count - 1; i++)
    {
        //Iteration by each pair of nodes
        int node1_idx = route[i];
```

```
            int node2_idx = route[i + 1];

            //Select the associated edge
            int index = env.world[node1_idx].neighboors.IndexOf(node2_idx)
                ;
            int edge_idx = env.world[node1_idx].edges[index];

            //Change the delta tau according to the current cost and the
                best cost so far
            DeltaTau(ref env, ref route);

            //Reinforce the route, delta_tau is a global variable
            env.edges[edge_idx].pheromone_amount += delta_tau;

        }
}
```

Evaporation

```
//---------------------------------------------------------------------
//Evaporation Function

public void Evaporation(ref MeshEnvironment env)
{
    //Loop around all the edges list. Reduce the pheromone according
        the p value.
    for (int i = 0; i < env.edges.Count; i++)
    {
        env.edges[i].pheromone_amount = env.edges[i].pheromone_amount
            * (1 - evaporation_factor);
    }
}
```

Find Route Process of an Ant

```
//---------------------------------------------------------------------
public List<int> FindRoute(ref MeshEnvironment env)
{
    //Variable to store the route
    List<int> route = new List<int>();
    route.Add(env.start_node);

    //Variable to detect a feasible route
    bool found_route = false;

    int current_node = env.start_node;
    while (!found_route)
    {
        //Select the node according the movement choice technique
        int next_node = SelectNextNode(ref env, current_node);

        //If the movement is possible
        if (next_node != -1)
        {
            //Add the new node and change the current node
            route.Add(next_node);
            current_node = next_node;
        }
        else
        {
            //Otherwise the ant gets stuck
            break;
        }

        //If we find the target node, we break the loop
        found_route = current_node == env.final_node;
    }

    if (found_route)
    {
        //Store some critical information in the current ant
        current_route = route;
        current_cost = ExtraTools.GetCost(ref current_route, ref env);
        counter_route++;
    }
    else
    {
        //If the ant gets stuck, the route is not feasible.

        //In this place we can put the different local interactions
           which only works
        //with the last node of this bad route.
        // ----- LOCAL INTERACTION FUNCTION SPACE ------
        route = new List<int>();
    }

    return route;
}
```

Compute Coefficient

```
//-------------------------------------------------------------------
public Double ComputeCoefficient(Double pheromone, Double distance,
    Double proximity)
{
    Double coefficient = 0;
    if (pheromone == tau_0 || counter_route < 3)
    {
        coefficient = Math.Pow(proximity, alpha) * Math.Pow(1 /
            distance, beta);
    }
    else
    {
        coefficient = Math.Pow(pheromone, alpha) * Math.Pow(proximity,
            beta);
    }

    return coefficient;
}
```

Appendix 2. Graphical Animation with Static and Dynamic Obstacles

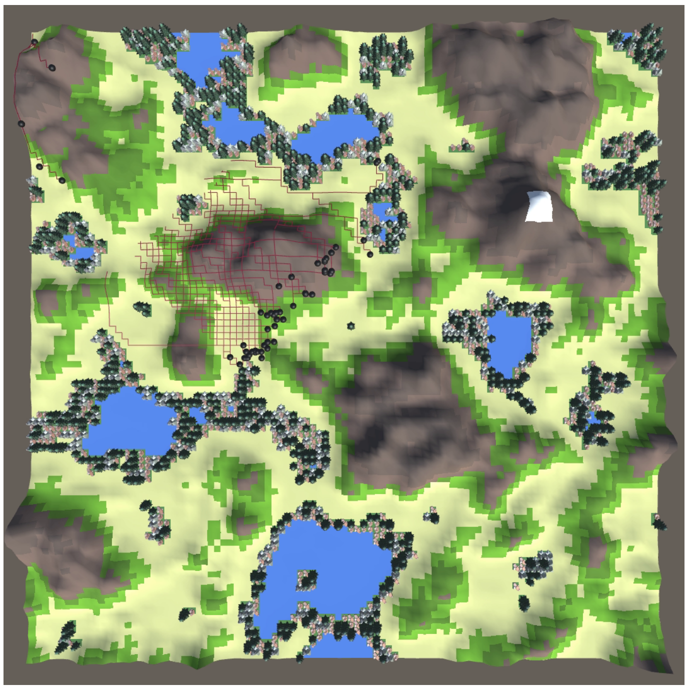


Figure 1: Random Walk Phase. Environment with Obstacles

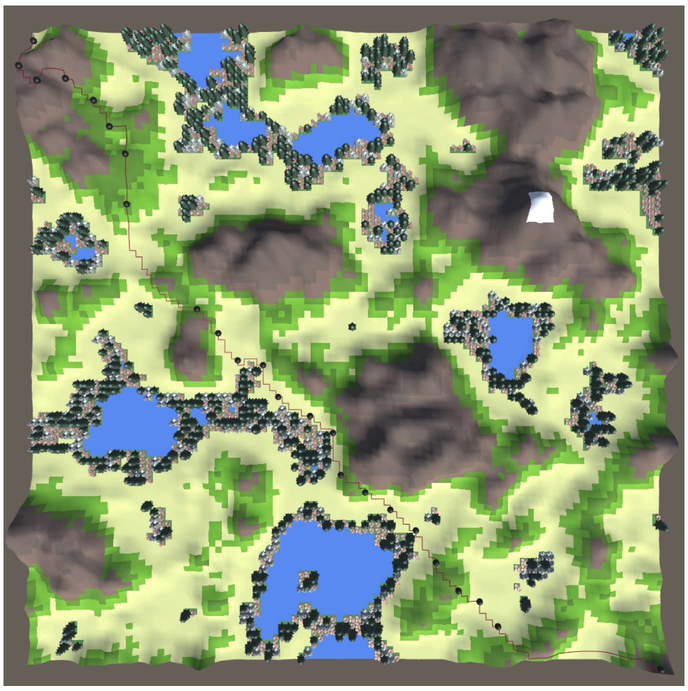


Figure 2: Route 1. Environment with Obstacles

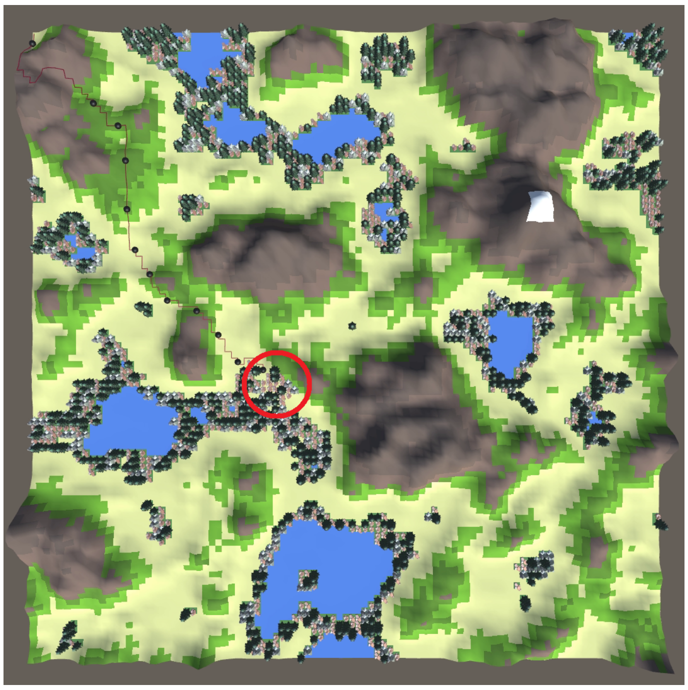


Figure 3: Obstacle Addition and Killed Ants. Environment with Obstacles

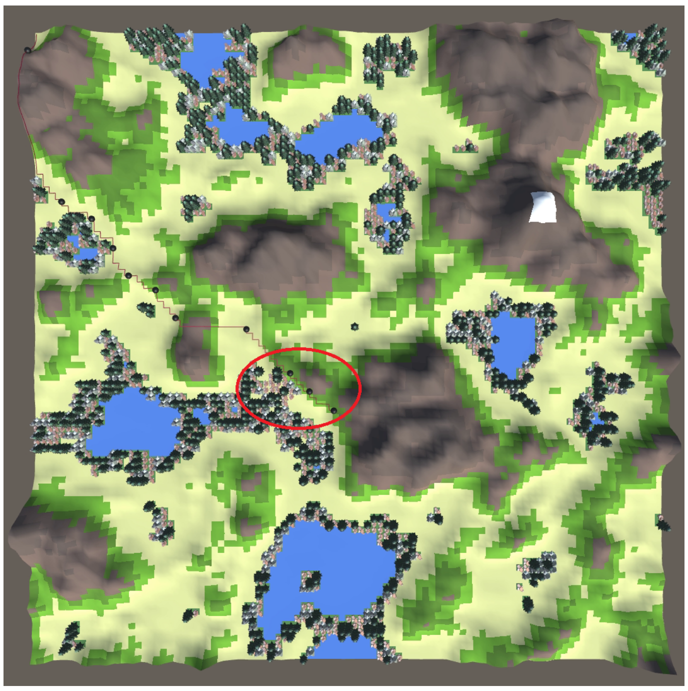
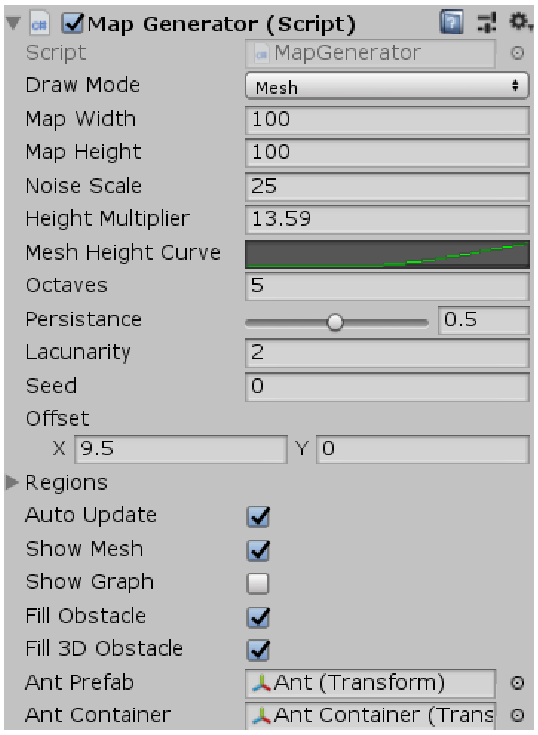


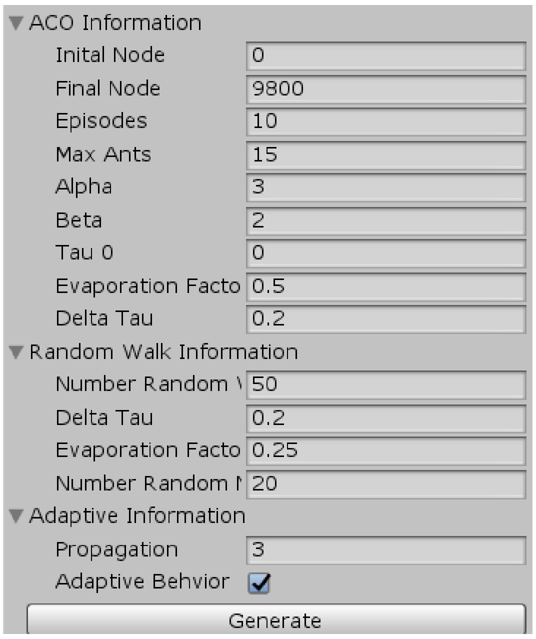
Figure 4: Rerouting Phase or Route 2. Environment with Obstacles



Map Generator (Script)

Script	MapGenerator
Draw Mode	Mesh
Map Width	100
Map Height	100
Noise Scale	25
Height Multiplier	13.59
Mesh Height Curve	
Octaves	5
Persistance	0.5
Lacunarity	2
Seed	0
Offset	X 9.5 Y 0
Regions	
Auto Update	✓
Show Mesh	✓
Show Graph	☐
Fill Obstacle	✓
Fill 3D Obstacle	✓
Ant Prefab	Ant (Transform)
Ant Container	Ant Container (Trans

(a) Menu Configuration 1



ACO Information	
Inital Node	0
Final Node	9800
Episodes	10
Max Ants	15
Alpha	3
Beta	2
Tau 0	0
Evaporation Facto	0.5
Delta Tau	0.2
Random Walk Information	
Number Random \	50
Delta Tau	0.2
Evaporation Facto	0.25
Number Random I	20
Adaptive Information	
Propagation	3
Adaptive Behvior	✓

(b) Menu Configuration 2

Figure 5: Menu