PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY OF MOHAMED BOUDIAF-M'SILA

FACULTY OF MATHEMATICS
AND COMPUTER SCIENCE
COMPUTER SCIENCE
DEPARTMENT N°: …

Domain: Mathematics and Computer Science

Branch: Computer Science

Option: Artificial Intelligence

A Dissertation

Submitted in partial fulfillment of the requirements for the degree of Master in Artificial intelligence

By

ZEGAAR Souheila

SAADOUNE Nour Chames Elassil

Title of the thesis

Path planning for mobile robots (A comparative study)

Under the supervision of

Dr. BENTRCIA Rahima

Composition of the jury

Dr.BENTRCIA Rahima	University of M'sila	Supervisor
Dr.BOUZAROURA Ahlem	University of M'sila	Reporter
Mr.BOUGHERARA Seddik	University of M'sila	Examiner

Academic Year: 2022/2023

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY OF MOHAMED BOUDIAF-M'SILA

FACULTY OF MATHEMATICS
AND COMPUTER SCIENCE
COMPUTER SCIENCE
DEPARTMENT N°:…

Domain:Mathematics and Computer Science

Branch: Computer Science

Option:Artificial Intelligence

A Dissertation

Submitted in partial fulfillment of the requirements for the degree of Master in Artificial intelligence

By

ZEGAAR Souheila

SAADOUNE Nour Chames Elassil

Title of the thesis

Path planning for mobile robots (A comparative study)

Under the supervision of

Dr. BENTRCIA Rahima

Composition of the jury

Dr.BENTRCIA Rahima	University of M'sila	Supervisor
Dr. BOUZAROURA Ahlem	University of M'sila	Reporter
Mr.BOUGHERARA Seddik	University of M'sila	Examiner

Academic Year: 2022/2023

DEDICATION

To my guiding stars, Mom and Dad, I am so grateful for everything you have done for me. You have helped me become the person I am today. I am a better person because of you.

Mom, you have sacrificed so much for me, always putting my needs and dreams before your own. Your selflessness and dedication have been a constant source of inspiration.

Dad, you have been my role model, my mentor, and my biggest source of strength, Your wisdom and your patience. This thesis is a tribute to you, the extraordinary man who has shaped my life in countless ways. Your unwavering love, guidance, and support have been the driving force behind my journey. I am forever grateful for your presence in my life.

To my dearest sisters and brothers, I dedicate this work to you. You have always been my best friends, my confidantes, and my biggest supporters. You have seen me at my best and my worst, and you have always loved me unconditionally.

To my dearest friends, your friendship and unwavering support have been a constant source of comfort and encouragement. Your presence in my life has brought joy, laughter, and countless cherished memories. Thank you for your unwavering belief in me and for always pushing me to reach for the stars. Your friendship is a true blessing.

Thank you once again for everything.

Sincerely,

Souheila

اهداء

الحمد لله وكفى والصلاة والسلام على الحبيب المصطفى وأهله ومن وفى اما بعد

الحمد لله الذي وفقنا لتثمين هذه الخطوة من مسيرتنا الدراسية بمذكرتنا هذه

اهدي ثمرة جهودي المتواضعة

إلى من أفضلها على نفسي ولم لا، فلقد ضحت من أجلي، ولم تدخر جهداً في سبيل إسعادي على الدوام أمي الحبيبة سعيدة. الى من كلت أنامله ليقدم لحظة سعادة وليمهد لي طريق العلم، صاحب الوجه الطيب والأفعال الحسنة، فلم يبخل علي طيلة حياته، والدي العزيز شعبان.

الى حبيبة قلبي اختي وصديقتي عمتي خليدة.

الى من ساندني ووقف معي في أصعب اللحظات، وامن بي الى الجذع الثابت الذي اتكئ عليه رفيق الدرب زوجي العزيز عبد الرؤوف.

الى اولادي الاعزاء اسر ورهف، اللذان استمتعا في تمزيق الاوراق والبكاء الدائم اثناء انجاز هذه المذكرة.

الى امي وابي الثانيين شعبان ونورة.

الى اخوتي الاعزاء بهجة الروح وفرحة القلب، حسام سراج أشرف رائد حفظكم الله.

الي من شاركتني الحلم واحتضنتني في التعب صديقتي سهيلة.

الي كل الاصدقاء الذين شاركوني رحلة الدراسة،

الى كل من كان لهم أثر في حياتي ،

الى كل من أحبهم قلبي ونسيهم قلمي.

سعدون نور

ACKNOWLEDGMENTS

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

In the Name of Allah, the Most Merciful, the Most Compassionate, Alhamdulillah, all praise belongs to Almighty Allah, the Lord of the worlds, and prayers and peace be upon Muhammad, His servant and messenger.

We want to sincerely thank ***Ms. Bentrcia Rahima****, our advisor, for her essential advice, support, and mentorship during my academic career. She was kind enough to answer my inquiries, no matter how simple or difficult they were, and we appreciate that. We consider ourselves really fortunate to have had her as our counselor because her knowledge and experience have been beneficial to us.*

We are grateful to each jury member for consenting to screen this application on our behalf. Also, thanks to all of our instructors who helped with our education and to everyone who came from near or far to support us in fulfilling this thesis.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
BFS	Breadth-First Search
DFS	Depth-First Search
EWNS	East West North South"
FL	Fuzzy Logic
GA	Genetic Algorithm
GUI	Graphical User Interface
HGA	Hybrid Genetic Algorithm
PSO	Particle Swarm Optimization
NN	Neural Network
RRT	Rapidly-exploring Random Trees
VFH	Vector Field Histogram

List of Equations

GENERAL INTRODUCTION

Technology has evolved continuously and rapidly, and one of the most notable developments in recent years is the developments of artificial intelligence, which represents a revolutionary advance in computing and which depends on the development of computer systems that can learn and think similarly to humans. Artificial intelligence technology aims to develop computer systems and software that enable multiple and diverse tasks and the ability to self-learn, creative thinking and right decision-making with high accuracy and high speed. This is done through the use of advanced techniques and tools such as artificial neural networks, machine learning, natural language processing, and others. Industrial intelligence technology is used in many fields such as banks, industry, medicine, education, etc., and helps to improve processes, save time and effort, and reduce errors, breakdowns and problems that may occur as a result of humans, by analyzing data and using artificial intelligence to make decisions and guide actions. In this way, technology helps to achieve many advantages and benefits for society and humanity in general, and contributes to the improvement of life and the development of many of the world's vital sectors. This area is expected to evolve further and deeper in the future, providing more precise and effective solutions to many of the problems facing society and the world at large, and investment in this area is increasing exponentially, with investment expected to reach billions of dollars in the coming years.

A robot is a machine designed to carry out a range of tasks automatically or with some degree of autonomy. Robots are commonly used in manufacturing and industrial settings, but are increasingly being developed for a variety of other purposes, including healthcare, agriculture, and exploration.

Mobile robots can be classified into different categories based on their mobility capabilities. For example, wheeled robots are designed to move on flat surfaces, while legged robots can traverse rough terrain. Other types of mobile robots include aerial robots (such as drones), aquatic robots (such as underwater vehicles), and even humanoid robots.

A robot is a machine designed to carry out a range of tasks automatically or with some degree of autonomy. Robots are commonly used in manufacturing and industrial settings, but are increasingly being developed for a variety of other purposes, including healthcare, agriculture, and exploration.

Using robots has become more widespread in recent years, with advancements in technology enabling the creation of increasingly sophisticated machines. Robots can be programmed to

perform tasks that would be difficult or dangerous for humans, and can also work for longer periods of time without requiring rest. In manufacturing, robots can perform tasks such as welding, painting, and assembly. In healthcare, robots are being developed to assist with surgeries and to provide care for patients. In agriculture, robots can help with tasks such as planting, harvesting, and monitoring crops. And in exploration, robots can explore environments that are too dangerous for humans to enter, such as the depths of the ocean or outer space.

While robots offer many benefits, including increased efficiency and safety, there are also concerns about their impact on employment and the economy. As technology continues to advance, it is likely that robots will play an increasingly important role in a range of industries and sectors.

Path planning is the process of finding a path between a start and a goal location in a given environment. It is a fundamental problem in robotics, autonomous vehicles, and other fields where motion planning is required. The goal of path planning is to determine a path that satisfies certain constraints, such as obstacle avoidance, time constraints, and energy efficiency.

These factors contribute to the increasing importance of route planning and trajectory planning algorithms in robotics. While trajectory planning algorithms take a given geometric path and add temporal information to it, path planning algorithms create a geometric path from an initial to a final point passing via pre-defined via-points, either in the joint space or in the operational space of the robot. Robotics relies heavily on trajectory planning algorithms because determining the times of passage at the via-points affects both the kinematic and kinetic aspects of motion. In other words, the robot is exposed to inertial forces (and torques) that depend on accelerations along the trajectory, whereas the values of the jerk (i.e., the derivative of the acceleration) primarily dictate the vibrations of its mechanical structure. Typically, path planning algorithms are categorized based on the techniques utilized to create the geometric path [1].

The navigation of mobile robots is a difficult problem since a variety of obstacles must be detected and a collision-free path must be chosen. Path planning is the process of identifying a collision-free path, which means that a robot should plan a reliable path between the source and the target without clashing with the dynamic and static impediments encountered in its complicated or unpredictable environment. This was accomplished in real time by constructing an algorithm employing a variety of strategies. Many approaches to planning algorithms have been proposed. The main goal of these planning algorithms was to get the robot to choose the shortest course possible. Every proposed algorithm aimed at giving the robot with intelligence that meets this requirement. Path planning is therefore critical in the case of mobile robots.

Chapter 1

Introduction

1.1 Overview

People have always wished to create intelligent machines that could execute tasks. These machines are now known as robots, after the Czech word robota, which means slavery or drudgery. Many people's imaginations have been sparked by robots, which have appeared in mythology, literature, and popular films. Robby the Robot, R2D2 and C3P0, Golem, Pushpack, Wanky and Fanny, Gundam, and Lt. Cmdr. Data are among popular robotic characters. Robots, like their literary counterparts, can take many shapes, and building them necessitates solving numerous issues in engineering, computer science, cognitive science, language, and so on. Robots must navigate our world, regardless of their shape or the work they must execute [2].

Mobile robots are becoming an increasingly important part of modern life, with applications in areas such as logistics and healthcare. As such, it is essential that they are able to navigate their environment efficiently and accurately. Path planning algorithms play a crucial role in this process, enabling the robot to plan its route from start to finish. In this paper, we present a comparative study of various path planning algorithms for mobile robots. We discuss the strengths and weaknesses of each algorithm and provide insights into how they can be used in different scenarios. Ultimately, our goal is to help readers understand the various approaches that can be taken when designing a path planning system for a mobile robot.

The field of robotics has seen significant advancements in recent years, particularly in the development of mobile robots. Mobile robots are becoming increasingly common in various industries, including manufacturing, healthcare, and agriculture. One of the critical challenges in the development of mobile robots is their ability to navigate autonomously in their environment, known as path planning. Path planning is the process of determining the optimal path for a robot to navigate from its current location to a specific destination while avoiding obstacles and ensuring the robot's safety.

In this section, we introduce the concepts related to the research topic and highlight the importance of the research and its applications in our lives. Path planning for mobile robots is a crucial task in robotic.

The general issue of path planning for mobile robots is defined as finding a path that a robot (with a given geometry) must follow in the described environment, to achieve a particular stance and direction Goal point, given a Starting position and orientation Start point(Figure1-1 [3]). The path is constrained to avoid obstacle collision and optimize the performance of the robot (For example, finding the path with the minimum distance, or find the way to minimize energy consumed by the robot).

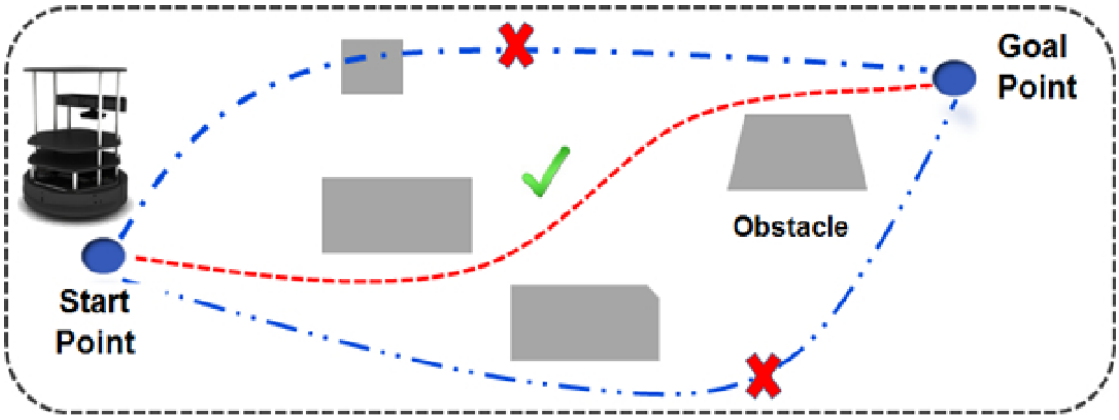


Figure 0-1 Path planning from Start point to Goal point.

1.2 Challenges in Path Planning Mobile Robots

Path planning for mobile robots is a challenging task due to several reasons.The environment may be complex and dynamic, with obstacles that can move or appear/disappear over time,the robot may have limited sensing and perception capabilities, which can affect its ability to detect obstacles or plan a safe path.

The robot may have limited computational resources, which can limit its ability to plan a path in real-time, the robot may have to operate in uncertain or unknown environments, which can make path planning even more challenging. Addressing these challenges requires developing new algorithms and techniques that can handle these complexities.

1.3 Problem Statement

Despite the numerous path planning algorithms available for mobile robots, each algorithm has its limitations, making it challenging to determine which algorithm is best suited for a particular application. Therefore, a comparative study of the different path planning algorithms

is necessary to identify their strengths and weaknesses and determine which algorithm is best suited for specific applications.

The problem we address in this research is to compare five path planning algorithms for mobile robots and identify the most suitable algorithm that can provide efficient and accurate results. We propose a solution by conducting a comparative study of different algorithms and evaluating their performance based on several metrics. The aim is to provide a comprehensive analysis of each algorithm and to provide insights into the best practices for path planning for mobile robots.

1.4 Aims of the Project

The primary objective of this research is to compare the performance of various path planning algorithms for mobile robots. The research aims to achieve the following objectives:

- ✓ To provide an overview of the current state-of-the-art path planning algorithms for mobile robots.
- ✓ To evaluate the performance of five path planning algorithms concerning their ability to find optimal paths.
- ✓ To determine which path planning algorithm is best suited for specific applications.
- ✓ To provide recommendations for future research in the field of path planning for mobile robots.

1.5 Scope and Limitations

This research focuses on a comparative study of various path planning algorithms for mobile robots. The study will evaluate the performance of the algorithms in terms of their ability to find optimal paths. The research will not focus on the design and implementation of mobile robots or the hardware used to develop these robots.

1.6 Thesis Organization

In this section, we provide an outline of the thesis and its organization. The thesis is organized as follows:

- Chapter 1 Introduction

- Chapter 2: Literature Review
- Chapter 3: Methodology
- Chapter 4: Results and Discussion
- Conclusion and Future Work

Chapter one provides an introduction to the research, including an overview of mobile robots, path planning, the problem statement, research objectives, scope, and limitations.

Chapter two, we review the existing literature on path planning algorithms for mobile robots, including their advantages and disadvantages, and identify the research gaps that our study aims to address.

Chapter three, we describe the methodology we used to conduct the comparative study, including the simulation environment, the evaluation metrics, and the algorithms we compared.

In Chapter four, we present the results of our study and discuss the performance of each algorithm based on the evaluation metrics.

Finally, we summarize our findings, conclude the thesis, and briefly discuss future directions on the research topic.

1.7 Thesis Contribution

In this section, we outline the scientific contributions of our research. Our study provides a comprehensive comparison of different path planning algorithms for mobile robots and identifies the most suitable algorithm that can provide efficient and accurate results. Furthermore, our study highlights the strengths and weaknesses of each algorithm and provides insights into the best practices for path planning for mobile robots. Our findings can be used as a guide for researchers and practitioners working in the field of robotics.

In our research, we have conducted a comprehensive and detailed comparison of well-known path planning algorithms, namely A*, BFS, DFS, Dijkstra's algorithm, and the Genetic algorithm (GA). It is important to note that this specific comparison, which combines classical and learning approaches, appears to be unique and novel in the existing literature.

Chapter 2

Background and Literature Review

2.1 Background

Due to its versatility and capacity for independent operation, mobile robots are becoming more and more common in the field of robotics. Mobile robot navigation relies heavily on the path planning algorithm, which gives them the ability to move from one place to another while avoiding obstacles and getting there as quickly as feasible.

This literature review provides an overview of the existing literature on path planning for mobile robots, highlighting the key concepts, algorithms, and techniques used in the field.

Path planning is the process of finding an optimal path from a starting point to a goal point while avoiding obstacles along the way. Mobile robots use sensors such as cameras, lidar, and sonar to perceive their environment and generate a map of the surrounding area. This map is then used to plan a safe and efficient path for the robot to follow.

There are several types of path planning algorithms, including potential field, A* search, Dijkstra's algorithm, and genetic algorithms. Each algorithm has its strengths and weaknesses, and the choice of algorithm depends on the specific requirements of the application.

2.2 Classification of Path planning

Path planning is heavily researched by researchers due to its usefulness in robot navigation. Understanding path planning classification is critical since it leads to the solution of the path planning problem. Robot route planning can be classed into several types based on factors such as obstacle type, environment type, planning type, space type, and time [4]. Path planning algorithms are classified into two types based on the availability of environmental information: off-line and on-line. Offline path planning of robots in contexts where fixed barriers and moving obstacle trajectories are known in advance is also known as global path planning. When complete information about the environment is not accessible in advance, the mobile robot gathers information as it goes through the environment using sensors. This is referred to as

online or local path planning. Off-line and on-line path planning algorithms can be divided into two types: classic approaches and evolutionary approaches [5] .

Path planning is classified into two basic categories: global path planning and local path planning.. Global path planning necessitates a completely understood environment and static terrain. The algorithm creates a complete path from the source point to the goal point before the robot begins its move in this sort of path planning. Local path planning, on the other hand, assumes that the environment is fully unknown to the mobile robot and that the algorithm is capable of devising a new path to reach the goal point [6]. We chose known static environment. As shown in Figure 2-1 [4] and stated in the following points:

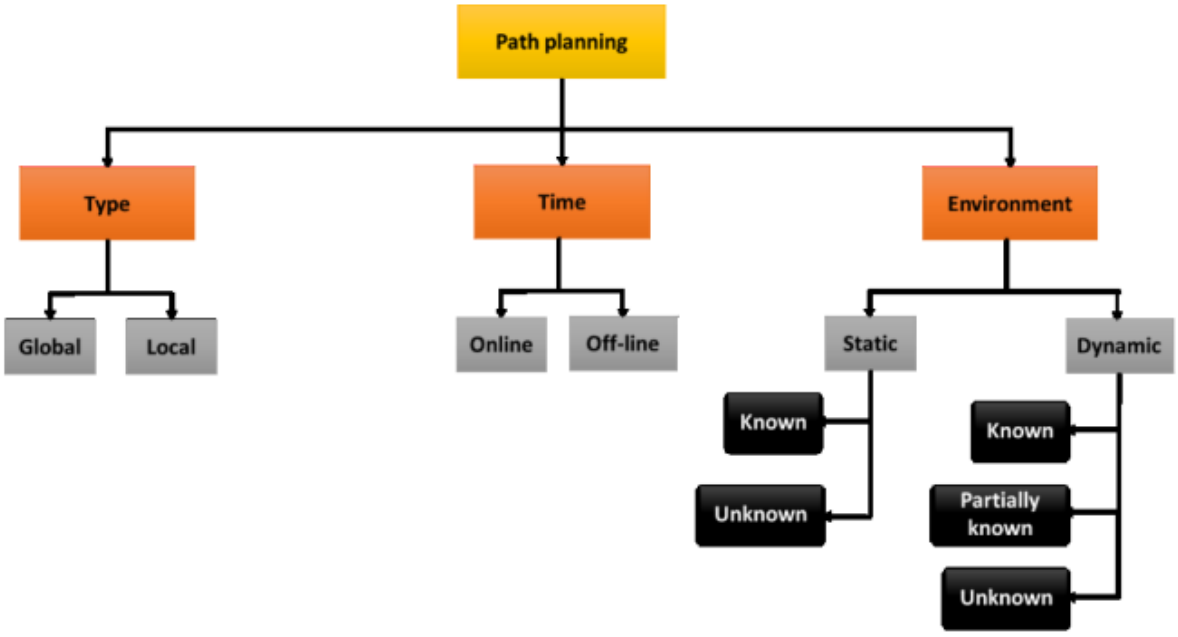


Figure 0-1 path planning Classification

2.3 Techniques and approaches used in path planning

Classical algorithms, Artificial Intelligence Techniques, and hybrid approaches are three different types of techniques used in computer science to solve complex problems.

While classical algorithms rely on pre-defined rules and mathematical models, machine learning-based methods use data and statistical analysis to make predictions and decisions. Hybrid approaches combine the strengths of both techniques to achieve better results.

2.3.1 Classical algorithm

Classical algorithms are step-by-step procedures designed to solve specific problems in a predictable and deterministic way.

Examples of classical algorithms include sorting algorithms, search algorithms, and graph algorithms. These algorithms have been around for decades and are widely used in various applications such as databases, operating systems, and networking.

For example, sorting algorithms are used to sort large amounts of data in databases, search algorithms are used to find specific files in operating systems, and graph algorithms are used to optimize network routing.

- The artificial potential field is a virtual force method proposed by Khatib [7] . The artificial potential field approach has the advantage of having strong real-time performance and being convenient for robot bottom control. The typical artificial potential field approach, however, has the following drawbacks: Inaccessibility of the aim and local minimum points.

- In [8]**,** Syed Abdullah *et al* proposed in this work Dijkstra's algorithm, a multi-layer dictionary is suggested to improve the storage structure. The approach was formerly used to optimize a single parameter (such as travel time, distance, and fuel) for movement between two locations. Although the path determined by the traditional Dijkstra's algorithm is the shortest, it might not be the most practical**.**

- BFS was first introduced by E.F. Moore in the 1940s .In [9], M.Bala *et al* They conducted a study about this algorithm,the experimental data demonstrates that the BFS algorithm can successfully and optimally implement path planning in a dynamic context. Reach the desired location while dodging the obstacles. However, it has been noted that the outcomes differ when using different heuristic functions.

2.3.2 Artificial Intelligence Techniques

Artificial Intelligence (AI) techniques such as Reinforcement Learning and Deep Learning have shown promising results in path planning for mobile robots.
Reinforcement Learning trains the robot to learn from its experience and optimize its path planning strategy. Deep Learning uses neural networks to learn the mapping between the input sensor data and the output control actions.

- Karaboga in [10] developed the artificial bee colony (ABC) technique in 2005, swarm-based intelligent foraging behavior of honeybees activities for searching their food. When compared to existing swarm-based algorithms, the new swarm algorithm is relatively simple and very adaptable. It's also quite robust, at least for the test problems in this paper. Based on the simulation findings, the suggested approach can be used to solve unimodal and multimodal numerical optimization problems.

- The optimum path for a wheeled mobile robot in a static environment is described in this paper [11] through a comparison of two optimization algorithms. PSO was first used to choose the optimal path for a mobile robot, and chaotic PSO was subsequently used to improve the outcomes of the path planning algorithm. According to the simulation results, the path discovered by Chaotic PSO is shorter than the path of PSO with less iterations.

- In [12] ZHANG used the artificial potential field method to begin using in conjunction with reinforcement learning, aiming to set the potential at each point of the robot's movement in the environmental space; however, this approach frequently runs into the issue of local optimality.

- Li *et al* in [13] proposed an improved ACO, which improved the convergence speed by adaptively changing the wave coefficient and updating the state transition rules.
- Hao *et al*. proposed an adaptive genetic algorithm based on collision detection [14], which solves the problem of low quality of genetic algorithm paths and low convergence iterations by optimizing genetic operators and adding collision detection methods.

- In this paper [15] Song *et al* proposed a HGA-ACO algorithm combined with the improved GA and the improved ACO ,powerful methods for robot path planning, which achieved good results in simulation and improved the overall performance of path planning.

- The fuzzy logic algorithm simulates the driving experience of the driver by combining physiological perception and action. The advantage of the fuzzy logic algorithm is its robustness, which eliminates the need for sophisticated mathematical models and avoids the problems of mobile robots in other algorithms that are highly dependent on the environment. The drawback is that establishing fuzzy rules is more complex [16]

2.3.3 Hybrid approaches

Hybrid approaches to path planning combine multiple classic approaches to overcome their limitations and improve performance.

- A novel hybrid approach combining the NN with the FL was developed in [17] For mobile robot navigation. The response time for NFOC on the simulation robot program was significantly reduced when compared to the response time for other fuzzy controllers.
- In Sasi Kumar *et al* [18] proposed A* based Potential field approach. Even in highly cluttered environments, an algorithm that performs better than the potential field method in terms of planning time and provides a more optimum path than the A* and more towards that of the potential field algorithm while overcoming the major problem of trapping at local minima may be used.
- In this paper [19] to resolve autonomous navigation in an unknown 2D static environment, a new technique based on sensor-based algorithms is introduced. Ahmed and colleagues compared the suggested approach against the Bug2 and VFH techniques. The simulation results demonstrated that the proposed (M-Bug) algorithm reduces the distance and time consumed by mobile robots and outperforms other approaches in terms of length and time.
- In Jie Qi *et al* [20] suggested method, which takes into account both path length and path smoothness, can quickly select the optimal node out of multiple alternatives. The MOD-RRT* can produce a better beginning path when compared to other static

planning methods. The usefulness of this approach in real-world applications is further demonstrated by actual experience.

There are still many different kinds of algorithms to look for the best course, and not all of them have been described or thoroughly examined in earlier studies. According on previous study different types of algorithms may do the same task in less or more time, space, effort, etc. Because each form of algorithm has a unique specialty, advantages, and disadvantages, Chapter four compares and thoroughly discusses the outputs of each type of algorithm.

2.4 Conclusion

These were some of the most common algorithms used in robots. All these algorithms are very complex with a combination of physics, linear algebra and statistics used to map actions and movement.

However, as technology progresses, robot algorithms will evolve to become more complex. Robots will be able to complete more tasks and think more about themselves.

Path planning is essential for mobile robots navigating in an unfamiliar environment. To address this difficulty, several methods and techniques have been created.

The path planning approach chosen depends on the application's specific needs, such as the complexity of the environment, the processing resources available, and the desired amount of autonomy.

Chapter 3

Methodology

3.1 Simulation environment

We selected a simulation environment that is representative of the real-world scenario in which the algorithms will be applied. The environment should contain obstacles and other features that are relevant to the problem at hand.

3.1.1 The Maze Problem

The main concept of maze solving algorithms is to discover the shortest path between two points while avoiding obstacles and dead ends.

A grid map, a graph, or a series of geometric forms can all be used to illustrate the maze. The agent's goal is to discover the best way through the maze while avoiding obstacles and minimizing traversal costs such as time or energy [21].

3.1.2 Drawing Mazes

There are four types of maze nodes that can be drawn:

- Wall: A node that cannot be traversed.
- Path: A node that can be traversed.
- Start: The node from which the algorithm will start solving.
- End: The node that must be reached by the algorithm.

3.1.3 Pyamaze

The pyamaze module was designed to facilitate the production of random mazes and the efficient application of various search algorithms.

The fundamental goal behind this module, pyamaze, is to make it easier to create configurable random mazes and work on them, such as using the search algorithm. You do not need to program the GUI or use Object-Oriented Programming when using this module because the module will offer you with help. This module makes use of the Tkinter GUI framework, which is built into Python, and you do not need to install any frameworks to use it.

Install the Package: On command prompt run the following command:

```
pip install pyamaze
```

3.1.4 GenerateaMaze

To generate a maze, first create the maze object and then use the CreateMaze function. The final statement will use the function run to conduct the simulation.

```
from pyamaze import maze
m=maze ()
m.CreateMaze ()
m.run ()
```

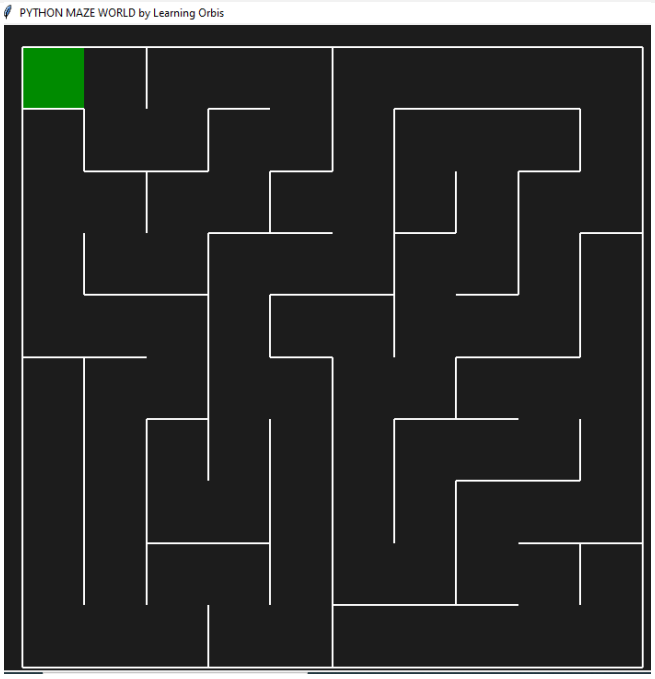


Figure 0-1 A random 10x10 Maze Generated

The maze's goal is the top-left cell, but we may change it to any cell we want. Furthermore, a Perfect labyrinth is constructed by default, which implies that all cells in the labyrinth are accessible and there is only one path from any cell to the destination cell. As a result, any cell can be viewed as the starting cell and hence is not marked. The last cell, i.e. the last row and column cell, is assigned as the start cell internally. As a result, the overall aim will be to discover a path from the bottom-right cell to the top-left cell.

We can change the size of the maze while creating that. For example, a 5x5 maze can be generated as:

```
from pyamaze import maze
m=maze(5,5)
m.CreateMaze()
```

```
m.run()
```

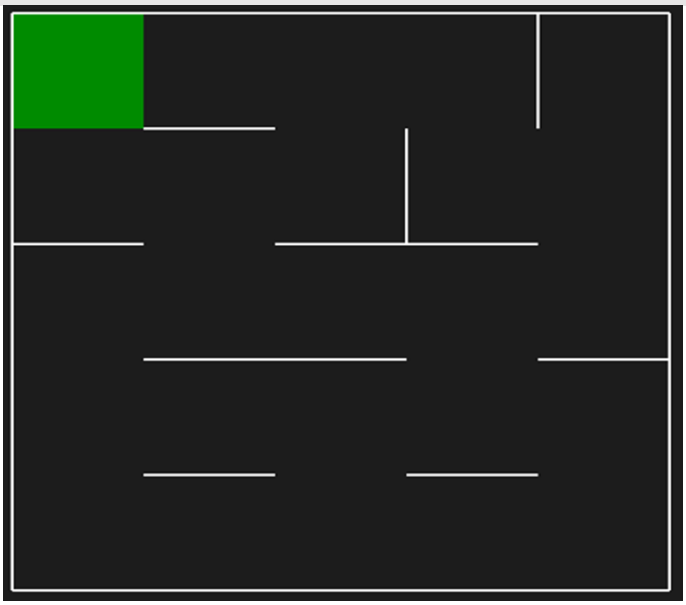


Figure 0-2 A Random 5x5 Maze

The row number is the first input to the CreateMaze function, and the column number is the second. You should use the function as m to generate a 15x20 maze.MakeMaze (15,20).

It is necessary to understand the maze parameters in order to use them in the application. To begin, each maze cell has two indices, one for the row and one for the column. The indices of the previously generated 5x5 maze are displayed here:

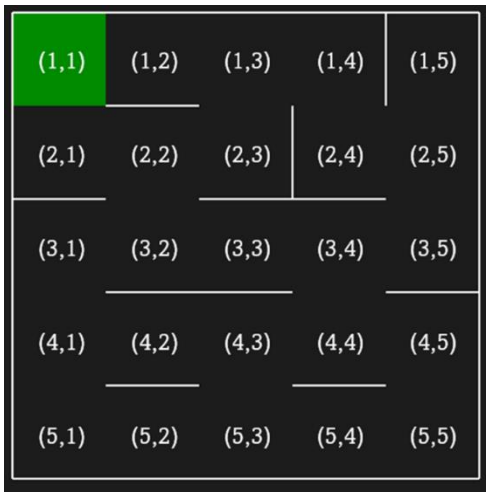


Figure 0-3 Indices of the Maze Cells

3.1.5 Multiple Paths Maze :

By default, the generated maze is Perfect Maze, which means there is only one path from any cell to the goal cell. However loopPercent set to 100 means the maze generation algorithm will maximize the number of multiple paths, such as:

```
m.CreateMaze(loopPercent=100)
```

The generated maze is shown here:

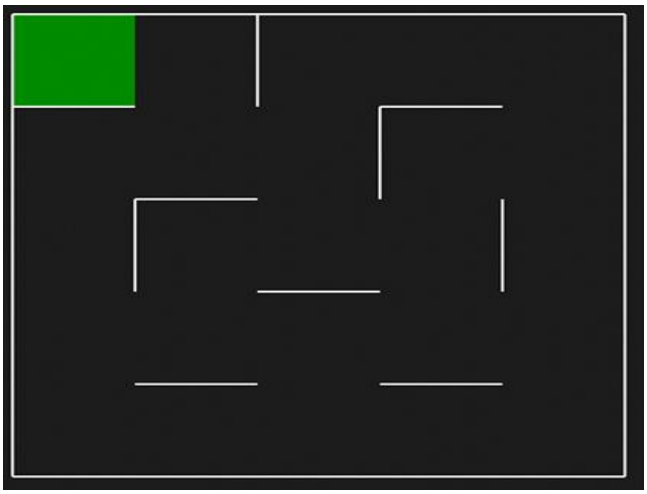


Figure 0-4 Maze with Multiple Paths (Loops)

3.1.6 Placing Agents inside the Maze

We can place an agent (or agents) inside the Maze. An agent can be a physical agent, such as a robot, or it can simply be employed to highlight or point out a cell in the maze.

The agent class is included in the pyamaze module for this purpose. After importing the agent class, we can create the agent object, and the first input argument should be the parent Maze. The agent will be placed on the Maze's start cell (the last cell) by default.

- **Path from start to goal:** The maze generating method in the pyamaze module (Recursive Backtracker) not only generates a random maze, but it also knows the path from start to destination. This information is available as a dictionary in the Maze's attribute path. The path's key is a cell, and the value is another cell that represents the travel from key cell to value cell in order to reach the goal.
- **Move the agent on a path:** After building a Maze and an agent (or agents) within the Maze, we can direct the agent to follow a specified path. The agent should be moved along the maze's ***path*** characteristic. In the maze class, we have a

method called tracePath that takes one dictionary as an input argument. The dictionary's key is the agent, and the value is the path we want that agent to take. The tracePath method will move the agent along the path.

```
from pyamaze import maze,agent
m=maze(20,20)
m.CreateMaze(loopPercent=50)
a=agent(m,filled=True,footprints=True)
m.tracePath({a:m.path})
m.run()
```

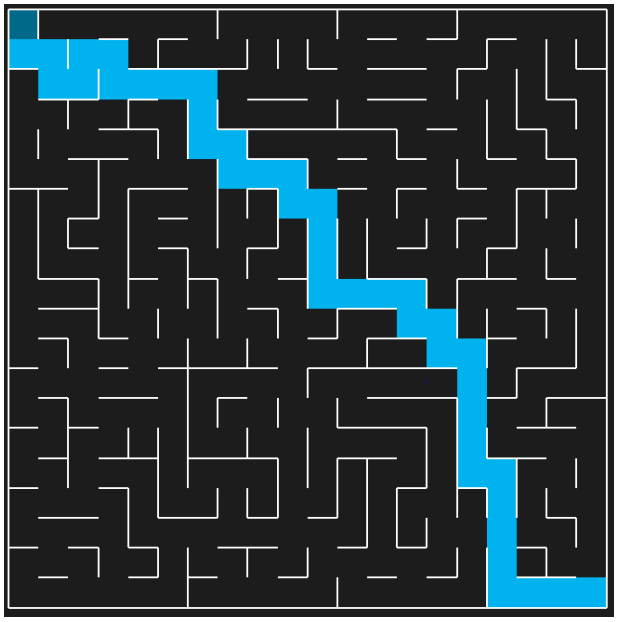


Figure 0-5 simulate the agent moving on the path

3.1.7 The various optional elements of the Agent class

- **Location of the agent:** The agent's default location is the Maze's start cell, which is at the bottom-right corner. You can adjust the cell's placement by changing the x and y values.
- The agent object has two attributes, x and y, which can be accessed and set once the agent has been created. Furthermore, the agent's attribute location is set to the tuple (x,y), which offers the entire x and y information as a single argument. You can also adjust it to a different value to change the agent's position.
- **Goal of the agent:** The agent's default aim is the Maze's goal, which means the agent's target is to attain the Maze's goal. If you want to change the agent's aim, you can do so by changing the argument goal while establishing the agent. The agent's goal should be assigned a two-valued tuple.

- **Size of the Agent:** By default, the size of the agent is smaller than the cell dimensions. You can set the argument ***filled*** to ***True*** and the agent will fill the whole cell.
- **Shape of the agent:** By default, the agent is of square shape and there is a second option of shape **arrow** that you can set to the ***shape*** argument and the agent will be arrow-head shaped. This will differentiate the front and other sides of the agent. The argument ***filled*** has no effect if the shape is set to an arrow.
- **See the footprints:** When you implement a search algorithm and the agent moves through the maze, it may be necessary to display the entire path trail. To accomplish this, set the optional argument footprints to True, and anytime the agent moves, an impression of footprints will be imposed on the prior location. Footprints is simply the agent's shape in a different color tone. Look at the output for this code:

```
from pyamaze import maze,COLOR,agent
m=maze(5,5)
m.CreateMaze()
a=agent(m,shape='arrow',footprints=True)
a.position=(5,4)
a.position=(5,3)
a.position=(5,2)
m.run()
```

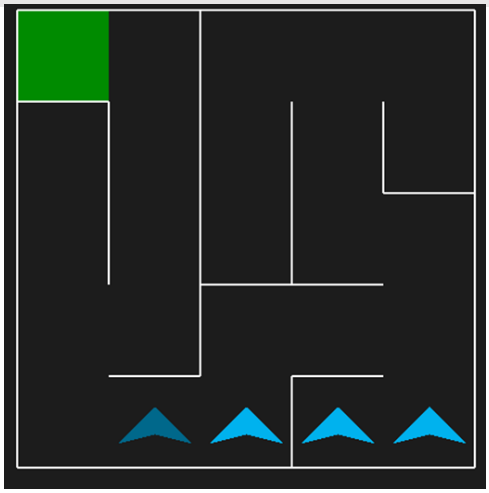


Figure 0-6 shape of the agent

There are three ways we can specify a path for the agent to follow:

- **Path as a Dictionary:** The path can be a dictionary, as illustrated above, with key-value pairs reflecting the movement from key-cell to value-cell.

- **Path as List:** There is also a possibility to provide the path as a List of cells. Then the agent will follow the path starting from the first cell inside the list to the last cell.
- **Path as String:** We can also provide the path as a string of movement directions (EWNS), e.g. 'EENWWSES' is a string of 8 steps for the agent to follow.
- **Kill the agent:** It is possible to kill the agent after it completes the path. By setting the argument *kill* to *True* and the agent will be killed after 300 milliseconds after completing the path.
- **Movement Speed:** We can control the movement speed of the agent using the argument *delay* having the default value of 300 milliseconds. It is a time delay between the movement steps of the agent.
- **Mark some cells:** For different demonstrations, it might be needed to mark a few cells. For that, there is an option of *showMarked* that can be set to *True* and any cell present inside the list of maze *markCell* will be marked if the agent passes through that cell.
- **Multiple Agents on different Paths:** There is also the possibility to move multiple agents on their own paths. For that, we can provide more agent-path information inside the input dictionary to the *tracePath* method. There will be the movement against all agent-path pairs provided in the dictionary.

We can also use the tracePath function several times. In that instance, all agents-paths provided the first time will finish their paths, followed by the other agent-path pairs provided the second time in tracePath, and so on.

We can go to the GitHub link and copy the module code as a Python file called pyamaze.py [22].

3.2 Evaluation metrics

Analytical comparisons of motion planning algorithms are done here using the general criteria we summarized. These criteria include

3.2.1 Time complexity

An algorithm's time complexity can roughly reflect its time cost. The big O notation is widely used to calculate an algorithm's time complexity. The magnitude of the complexity of the algorithm's main phases requires more consideration than the coefficients in front of the magnitude [23].

3.2.2 Path length

The total length of the path generated by the algorithm from the start point to the end point.

3.2.3 Completion Time (Running time (s))

The amount of time required to generate a feasible path from the start point to the end point.

3.2.4 Search path

A search path refers to the sequence of nodes or vertices traversed during a search algorithm's exploration of a graph or network. It represents the route followed by the algorithm to reach a particular goal or find a target node within the graph.

3.2.5 Robustness

refers to the ability of a system, algorithm, or process to function effectively and reliably under various conditions or in the presence of uncertainties, disturbances, or unexpected inputs.

Robustness, in this case, refers to the ability of the maze generation algorithm to handle different values of "loopPercent" effectively and consistently produce mazes with the desired characteristics. A robust algorithm would be able to generate mazes with varying levels of loopiness while still maintaining valid and solvable maze structures.

3.3 Algorithms compared

We selected a set of algorithms that are relevant to the problem and represent a range of approaches.

3.3.1 Dijkstra's algorithm

In the Djikstra algorithm, a graph G is considered with n nodes via edges e by computer scientist Edsger W. Dijkstra in 1959 [24].
Edge $e_{n1,n2}$ has the cost from node n1 to node n2. Here, two lists are maintained: visited list V and unvisited list U. s is the start node, g is the goal node. U is a min-priority queue [25].

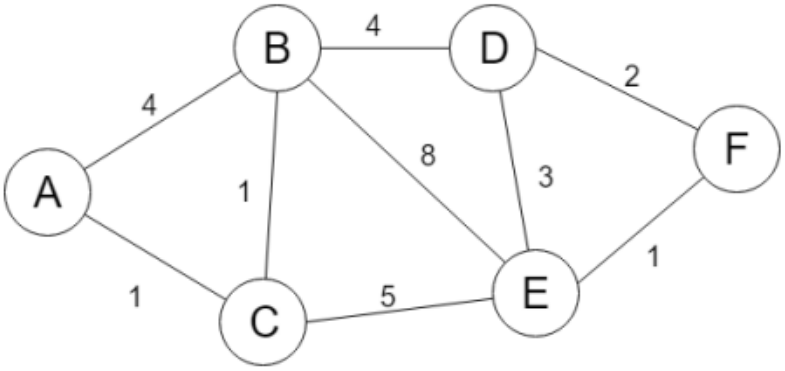


Figure 0-7 Graph G with costs on edges.

First, s is placed in U with $e_{s,} = 0$; no cost is incurred. The other nodes are added to U with $e_{s,n1\ldots k} = \infty$, where k is the number of nodes in the graph. From U, the node which has the minimum cost is chosen. In this case, the start node is picked, and costs from the start s to its neighbors x are updated. The rule for updating the cost for x is shown in (1), where $e_{current}$ refers to x cost in U and $x - 1$ is the current node being considered. After updating the costs to each neighbor, the selected node (start node) is removed from U and placed in V . The same steps are repeated until g is added to V or if the minimum cost in U is ∞, i.e. no path to g found [25]. The pseudo code [26]

$$e_{s,xupdated} = \min(e_{s,xupdated}, e_{s,x-1} + e_{x-1,x}) \quad (1)$$

Algorithm Dijkstra's

$Require: G, E, vstart, vgoal$
1: $for\ all\ vgoal\ \in\ vgoal\ do$
2: $insert(vgoal, 0, open - list)$
3: $while\ the\ open - list\ is\ not\ empty\ do$
4: $vj\ =\ get - best(open - list)$
5: $if\ vj\ \in\ vstart\ then$
6: $return\ SUCCESS$
7: $for\ all\ vi\ such\ that\ ei,j\ \in\ E\ do$
8: $if\ vi\ is\ unexpanded\ or\ ci\ >\ di,j\ +\ cj\ then$
9: $ci\ =\ di,j\ +\ cj$
10: $update(vi\ , ci\ , open - list)$
11: $set\ back - pointer\ from\ vi\ to\ vj$
12: $return\ FAILURE$

3.3.2 A star

The A* Algorithm is one of the most effective and widely used strategies for path finding and graph traversal. This approach is used in many games and web-based maps to effectively locate the shortest path. It is, in essence, a best-first search algorithm.

- A* Algorithm works as
- It maintains a tree of paths originating at the start node.
- It extends those paths one edge at a time.It continues until its termination criterion is satisfied.

A* Algorithm extends the path that minimizes the following function:

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

Here,

- 'n' is the last node on the path
- g(n) is the cost of the path from start node to node 'n'
- h(n) is a heuristic function that estimates cost of the cheapest path from node 'n' to the goal node [27].

Algorithm A*

Require: G, E, vstart, vgoal

1: $for\ all\ vi\ \in\ vgoal\ do$

2: $insert(vi\ , hstart, i, open - list)$

3: $while\ the\ open - list\ is\ not\ empty\ do$

4: $vj\ =\ get - best(open - list)$

5: $if\ vj\ \in\ vstart\ then$

6: $return\ SUCCESS$

7: $for\ all\ vi\ such\ that\ ei,j\ \in\ E\ do$

8: $if\ vi\ is\ unexpanded\ or\ di,goal\ >\ di,j\ +\ dj,goal\ then$

9: $di,goal\ =\ di,j\ +\ dj,goal$

10: $ci\ =\ hstart,i\ +\ di,goal$

11: $update(vi\ , ci\ , open - list)$

12: $set\ back - pointer\ from\ vi\ to\ vj$

13: $return\ FAILURE$

3.3.3 Breadth-First Search

BFS by E.F. Moore in the 1940s, Given a digraph $G\ =\ (V, E)$ and source nodes, the breadth-first search method searches the connected edges of S in G to find all nodes that can be reached from S. The distance between S and all these accessible nodes (i.e., the minimum cost sum) is calculated. The search algorithm generates a breadth-first tree with roots S and all the reachable nodes of S. For any node υ reachable from S, the path from S to V in the breadth first tree corresponds to the shortest path from S to V in digraph G, i.e., the lowest cost expected path. Since the data structure of the lane-level map is sparse, this paper uses an adjacency table to store the road data. For breadth-first search, several additional data structures are used to store the relevant nodes. All sub-nodes resulting from the expansion of the node are added to a first-in, first out queue. Neighbor nodes that have not been checked are placed in the open-list queue. Open-list O is used to store nodes connected to node u found by the adjacency table as the priority queue for subsequent searches. Each node $u\ \in\ V$ that completes the search is placed in the close-

list queue. Close-list C can also be thought of as a breadth-first tree of completed searches. The travel time cost between the source node s and node u calculated by the BFS algorithm exists in the variable $d\ [u]$. The search process of the BFS algorithm is shown in Figure 3-11. First, the source node s is taken as the root node in C. Then, the root node is used as the parent node u in the adjacency table, and the result is added to O. After that, node υ is gradually extracted from O, and the cost of the path segment corresponding to node (u, υ) is added to the path sequence. In this process, υ is constantly added to queue Q as the parent node of the next layer search. The path search and cost calculation steps are repeated until the search reaches the target point or the search of all the nodes in the digraph is completed.

Algorithm Breadth-First Search

Require: G, E, vstart, vgoal

1: $for\ all\ vgoal\ \in\ vgoal\ do$
2: $push - back(vgoal, open - list)$
3: $while\ the\ open - list\ is\ not\ empty\ do$
4: $vj\ =\ pop - top(open - list)$
5: $if\ vj\ \in\ vstart\ then$
6: $return\ SUCCESS$
7: $for\ all\ vi\ such\ that\ ei,j\ \in\ E\ do$
8: $if\ vi\ is\ unexpanded\ then$
9: $push - back(vi, open - list)$
10: $set\ back - pointer\ from\ vi\ to\ vj$
11: $return\ FAILURE$

The functions *push-top ()* and *pop-top ()* place a node on the top of the stack and remove a node from the top of the stack, respectively. Given a finite graph, depth-first search is complete. However, in a countably infinite graph, depth-first search is not complete and may not even find a solution when one exists. In practice, depth-first search is used when v_{start} contains many nodes, and all of them are expected to be distant from $v_{\cdot goal}$

3.3.4 Depth-First Search

In the 1960s, Charles A. Coulomb developed DFS. Depth-first search employs the method of searching "deeper" in the graph whenever possible, as the name implies. Depth-first search looks for edges that branch off from the most recently discovered vertex that still contains undiscovered edges. After exploring all of's edges, the search "backtracks" to explore edges leaving the vertex from whence was identified. This technique is repeated until we have found all of the vertices that can be reached from the initial source vertex. If there are any uncovered vertices, depth-first search chooses one of them as a new source and restarts the search from there. This method is repeated until the algorithm has discovered every vertex.

When depth-first search discovers a vertex during a scan of a previously discovered vertex u's adjacency list, it records this occurrence by setting's predecessor attribute:υ.π to u. Unlike breadth-first search, whose predecessor subgraph forms a tree, the predecessor subgraph produced by a depth-first search may be composed of several trees, because the search may repeat from multiple sources [28].

Algorithm Depth-First Search

$Require: G, E, vstart, vgoal$

1: $for\ all\ vgoal\ \in\ vgoal\ do$

2: $push - top(vgoal, open - list)$

3: $while\ the\ open - list\ is\ not\ empty\ do$

4: $vj\ =\ pop - top(open - list)$

5: $if\ vj\ \in\ vstart\ then$

6: $return\ SUCCESS$

7: $for\ all\ vi\ such\ that\ ei, j\ \in\ E\ do$

8: $if\ vi\ is\ unexpanded\ then$

9: $push - top(vi, open - list)$

10: $set\ back - pointer\ from\ vi\ to\ vj$

11: $return\ FAILURE$

3.3.5 Genetic Algorithms

John Holland created the first genetic algorithms (GAs) in 1960.. GAs are Darwinian evolution-inspired algorithms that include an initiation mechanism, a fitness function to evaluate each chromosome, natural selection, crossover, and mutation operators .The GAs start by randomly producing an initial population that represents the alternative solutions (chromosomes) to the problem to be improved. An adaptation function then evaluates each chromosome to determine the quality of each proposed solution. Following that, genetic operators are employed to develop the new progeny; selection is used to pick the parents who will be subjected to reproduction based on their adaption values. Later crossover is applied to products new offspring by recombining data from the two parents selected in the previous step.Mutation is employed to ensure population diversity by modifying the genetic structure of some individuals based on a mutation rate. This evolutionary cycle is repeated until the stopping condition is met, which can be the number of generations previously fixed, or the process can be stopped if the population does not involve itself quickly enough. The steps of the standard GAs [29] are detailed in the following :

Genetic Algorithm

Pseudo-code of the standard genetic algorithm

1: $\boldsymbol{Input}: N: Population\ size; Pc: Crossover\ rate; P: Mutation\ rate.$

2: $\boldsymbol{Output}: Best\ Chromosome.$

3: $t \leftarrow 0$

4: $Initialize\ arbitrarily\ the\ initial\ population\ P(t)$

5: $\boldsymbol{while}\ (not\ termination\ condition)\ do$

6: $Evaluate\ P(t)\ using\ a\ fitness\ function$

7: $Select\ P(t)\ from\ P(t-1)$

8: $Recombine\ P(t)$

9: $Mutate\ P(t)$

10: $Evaluate\ P(t)$

11: $Replace\ P(t-1)\ by\ P(t)$

12: $t \leftarrow t+1$

13: $\boldsymbol{end}$

The steps of the standard GAs [4] :

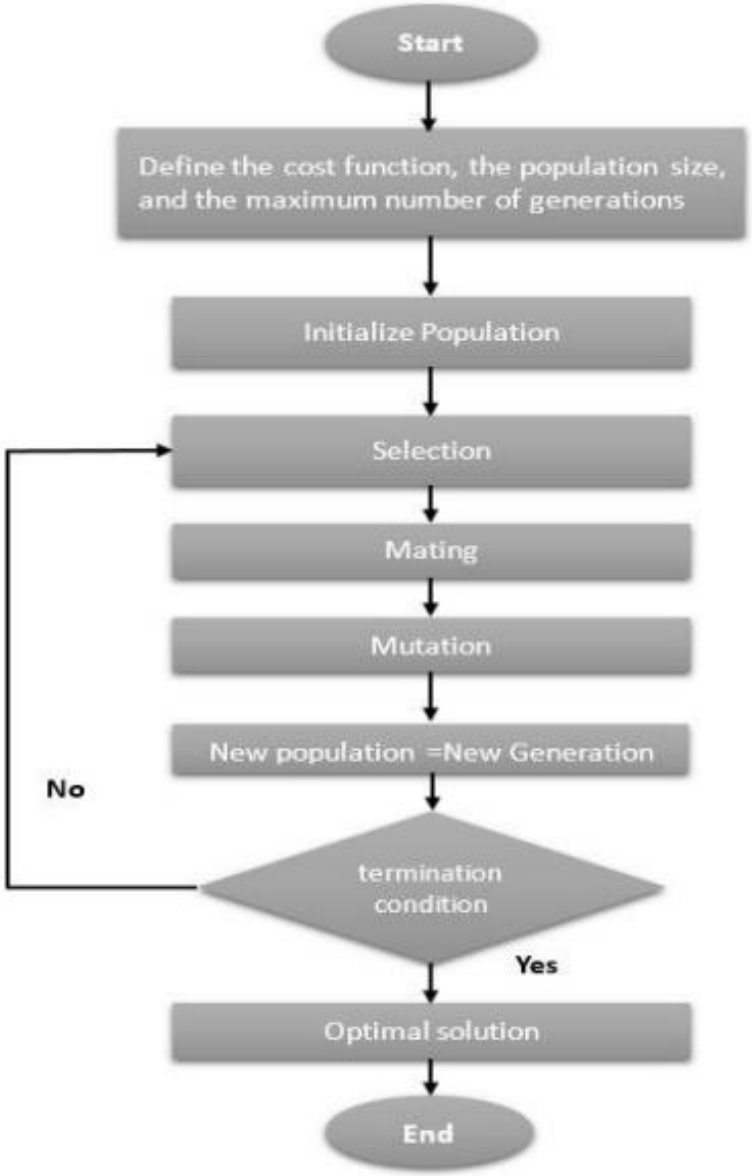


Figure 0-8 The flowchart of the Genetic algorithm

A GA is executed the following five components:

- A suitable genetic representation for individual (chromosomes).
- A method to generate the initial population.
- A fitness function to evaluate the quality of each potential solution.
- Genetic operators that modify the genetic composition of parents to produce a new offspring.
- The choice of the values of the various GA parameters (population size, cross over rate, mutation rate, stopping criteria... etc.

Chapter 4

Results and Discussion

In this chapter, performance simulation experiments are considered. Maze environments and single-point robots that can move without kinematic constraints. In the environment we mentioned earlier (Chapter 3) some indicators change; the number of obstacles and loop percent (multiple path).

The experiments are designed to measure the performance of the following basic search algorithms: Dijkstra's algorithm, A* algorithm, BFS algorithm, DFS algorithm and the genetic algorithm.

This study uses a typical environment to test performance under different environment styles. In this work, six different environments were used experimentally. The figures below show six environments with 30×30 cells.

4.1 Tools used

4.1.1 Hardware : Laptop

Processor	Intel® Core™ i5-6200U CPU @ 2.30GHz
Installed RAM	4.00GB
System Type	Windows 10 Pro Home 64-bit
Graphics	Intel® HD Graphics 520

4.1.2 Software :

Python-3.11.3: Visual Studio is an integrated development environment (IDE) from Microsoft that is used to develop software for Microsoft Windows, macOS, and Linux. It is a powerful tool that provides a wide range of features for writing, debugging, and testing code.

Visual Studio Code 2023: Visual Studio is an integrated development environment (IDE) from Microsoft that is used to develop software for Microsoft Windows, macOS, and Linux. It is a powerful tool that provides a wide range of features for writing, debugging, and testing code.

4.2 SIMULATION RESULTS

This section describes the results of the implementation and outlines the throughput of the agent by sampling all algorithms for a given task. Our algorithm has been tested in 6 scenarios. Each approach was run 10 times on each environment, the experiments were carried out (10*6*4) 240 times.

4.2.1 The loopPercent parameter

In a maze generator controls the number of loops in the generated maze. A higher loopPercent will result in a maze with more loops, while a lower loopPercent will result in a maze with fewer loops.

Loops in a maze can make it more interesting to explore because they introduce multiple paths to the goal. Instead of having just one direct route, the presence of loops allows for different choices and decision points within the maze.

A loopPercent value of 100 indicates that the maze generation algorithm will prioritize creating as many loops as possible, resulting in a maze with multiple paths and a higher level of complexity. The generated maze will have dead ends, loops, and branching paths, offering different routes to reach the goal cell.

Here are some examples of mazes with different loopPercent values:

LoopPercent=15: will have a moderate number of loops. There will still be a single path from any cell to the goal cell, but there will be some dead ends and backtracking required.

LoopPercent=50: This will generate a maze with a moderate number of loops. There will still be a single path from any cell to the goal cell, but there will be some dead ends and backtracking required.

LoopPercent=100: This will generate a maze with a maximum number of loops. There will be many dead ends and backtracking required to find the exit.

The loopPercent parameter can be a useful tool for controlling the difficulty of a maze. A higher loopPercent will result in a more challenging maze, while a lower loopPercent will result in an easier maze.

The experiments are conducted in six different environments, each with 30×30 cells. The environments differ in the number of obstacles and the loop percent. The results of the experiments are shown in Table 4-1.

Environment	Number of obstacles	LoopPercent
1	0	15%
2	0	50%
3	0	100%
4	24 (random)	15%
5	24 (random)	50%
6	24 (random)	100%

Table 0-1 The six scenarios of the generated maze (environment)

4.2.2 Scenario Number 1

Simple maze (30×30) : environment without obstacle and loopPercent =15%

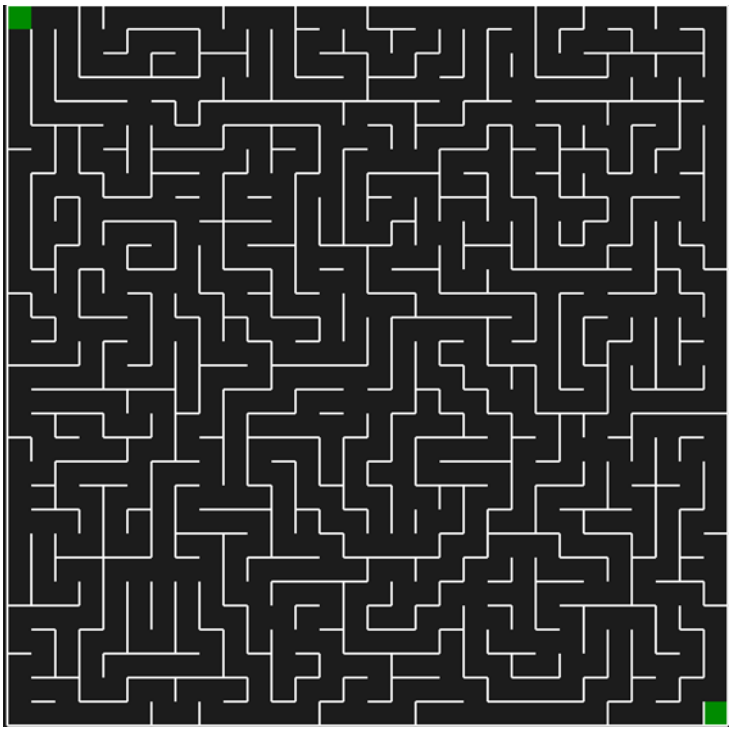


Figure 0-1 Environment 1

The simulation results of all methods were given in Figure 4-2 to the same point and in the same environment, where the starting point was taken as (30, 30) and the final point as (1,1).

In the six scenarios, the same start and goal points were used to demonstrate the shortest path outcome across all algorithms.

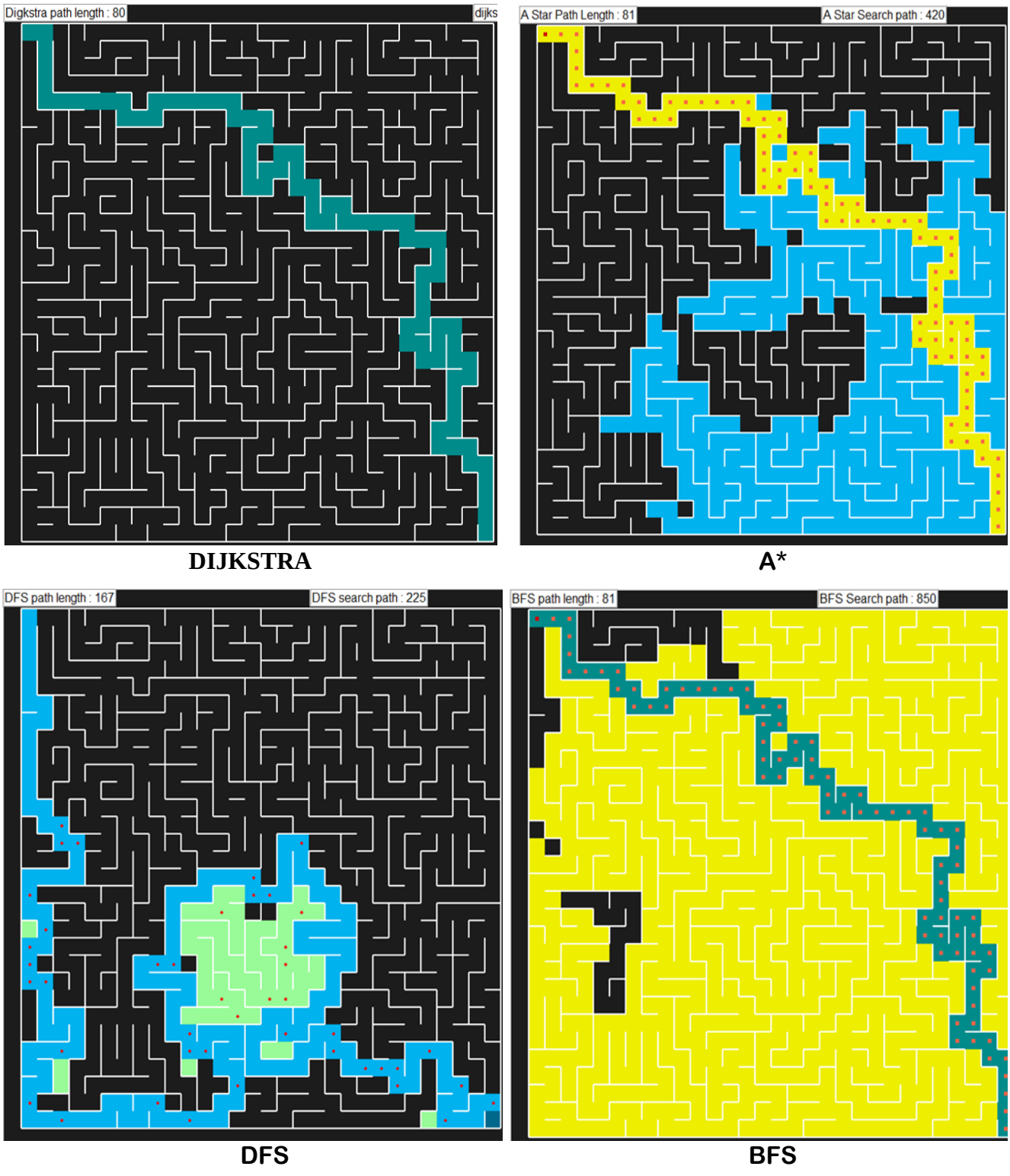


Figure 0-2 Best path found by the four algorithms in environment 1

- Blue color : path length
- yellow color : search path
- Red color : obstacles

4.2.3 Comparison of Scenario Number 1

PATH		PATH LENGTH				SEARCH PATH		
START	GOAL	BFS	DFS	Dijkstra	A*	BFS	DFS	A*
1.1	30,30	81	249	80	81	895	717	780
30,30	1.1	81	167	80	81	850	226	420
30.1	1,30	85	175	84	85	880	800	561
1.1	1,30	48	184	47	48	551	816	124
30,30	1,30	54	292	53	54	481	564	139
1.4	30.25	93	207	92	93	892	510	685
14.14	20.2	50	205	49	50	860	327	362
18.1	18.28	43	232	65	66	353	569	346
2.5	24.25	81	229	80	81	802	573	520
23.8	8.24	66	172	65	66	864	776	525

RUN TIME(S)			
BFS	DFS	Dijkstra	A*
0.002005061	0.002011593	0.605722499	1.336963251
40.6681511	4.344854274	58.58835439	15.35871476
13.96170183	24.73246096	42.84185757	4.908237874
13.26942748	22.97365103	44.76641811	4.645087762
13.34225021	26.67625859	45.00261432	4.828699252
0.312630069	63.15037309	5.784698825	2.282187001
19.20221173	4.07284373	46.92656237	6.386962901
0.361310427	18.34550692	6.74357265	2.063380567
0.936337194	19.43480541	10.65947152	2.493425236
10.08007557	23.41934712	44.68996078	6.375108406

Table 0-2 Simulation data for Scenario 1

Table 4-2 illustrates the obtained results of the compared search algorithms. The best algorithm in path length criterion depends on the values listed in the table. According to the table, Dijkstra's algorithm seems to provide the shortest path in most cases. There are clear differences in the search path among the listed algorithms. The values of the search path range from low numbers to high numbers. In this criterion, it seems that BFS explores a large number of nodes and may not be practical in some cases, while DFS

explores fewer nodes but may cause delays in reaching the goal, the A* algorithm consistently outperforms the other algorithms in terms of both path length and run time. The following Figure 4-3, Figure 4-4**,** and Figure 4-5 demonstrate these differences:

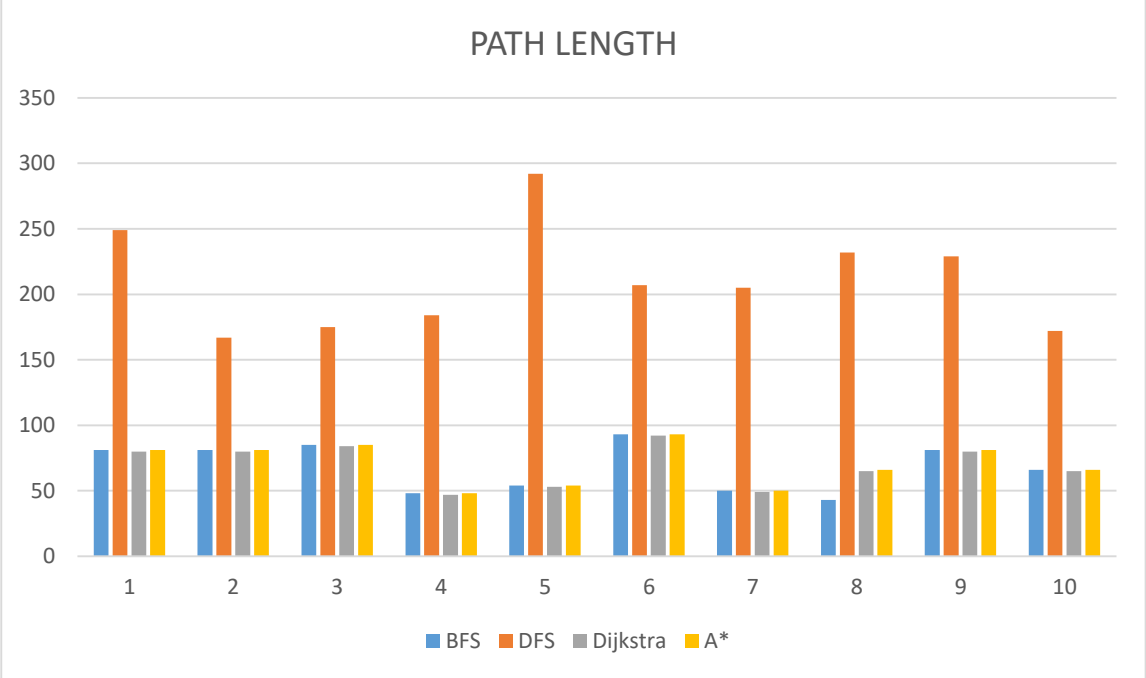


Figure 0-3 Path length of Scenario Number 1

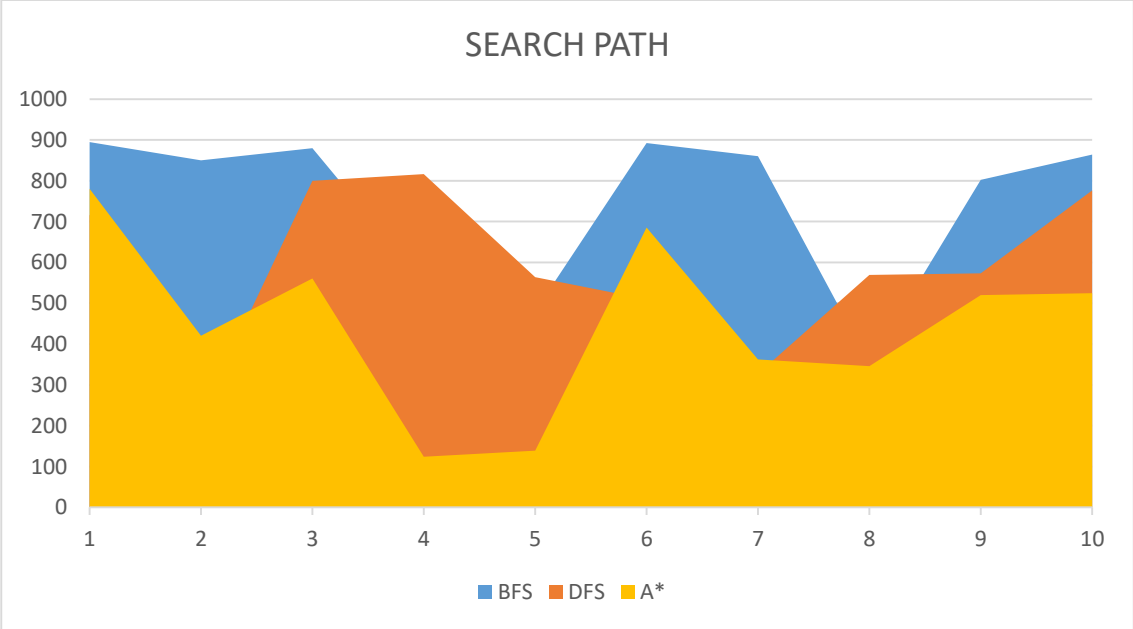


Figure 0-4 Search path of Scenario Number 1

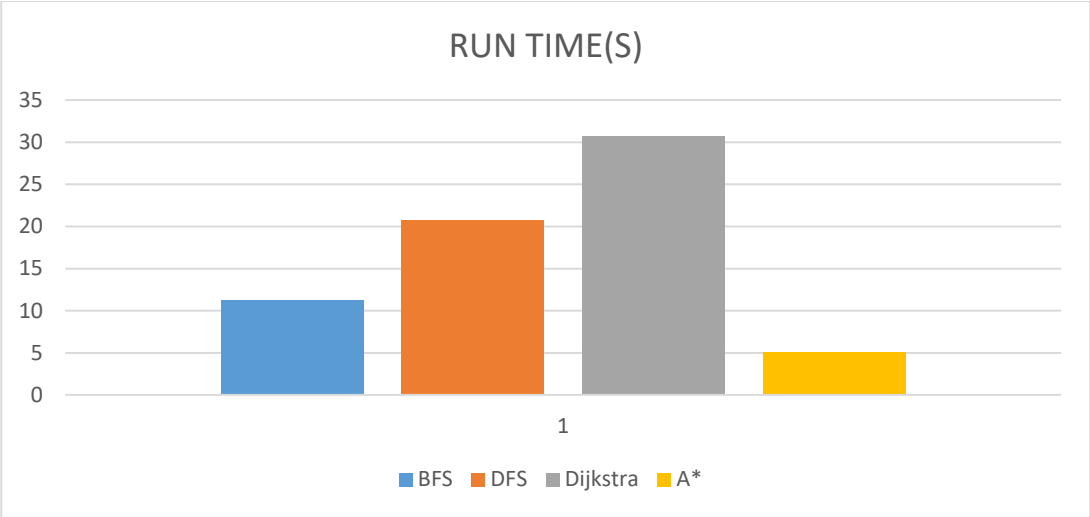


Figure 0-5 Average of Run Time (S) in Scenario Number 1

4.2.4 Scenario Number 2

Simple maze (30×30): environment without obstacle and loopPercent =50%

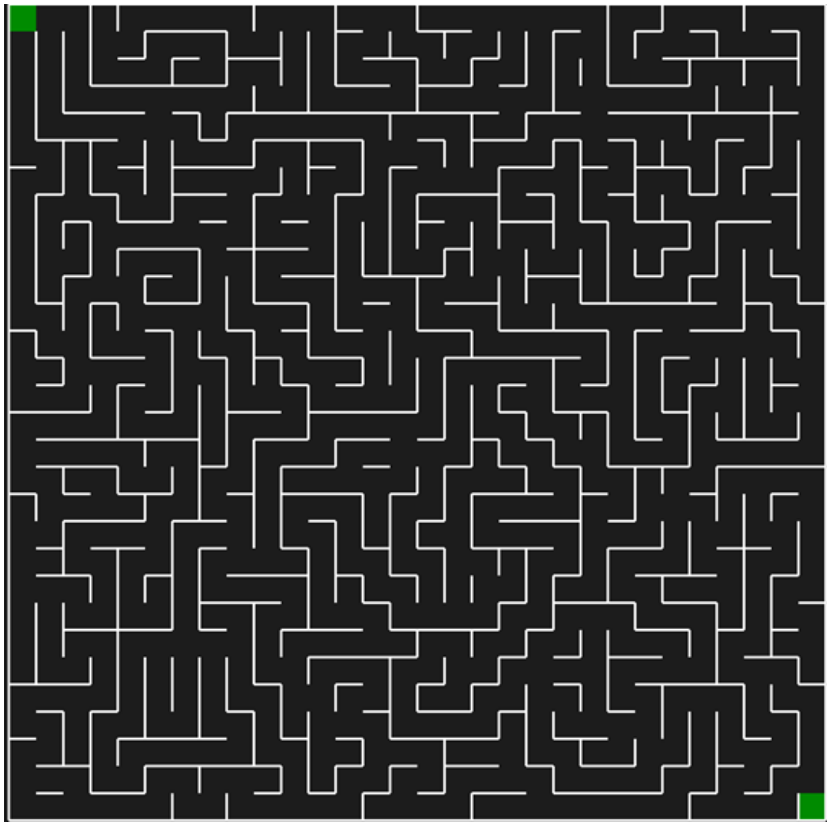
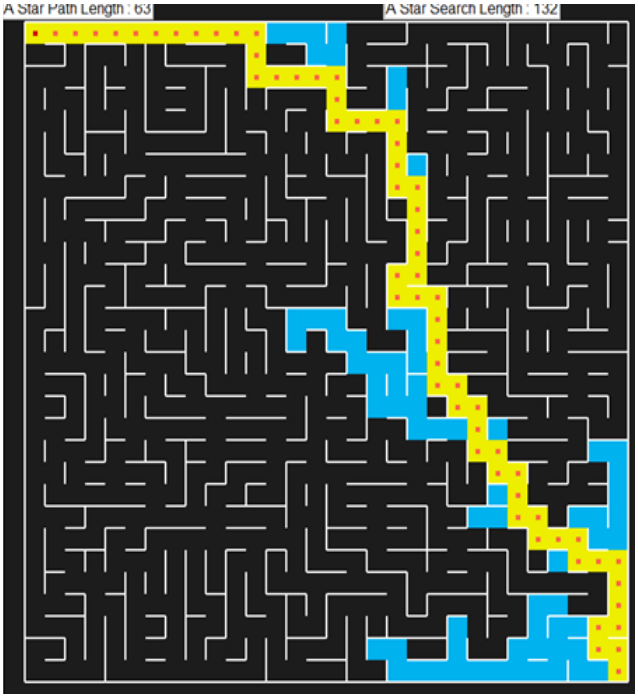
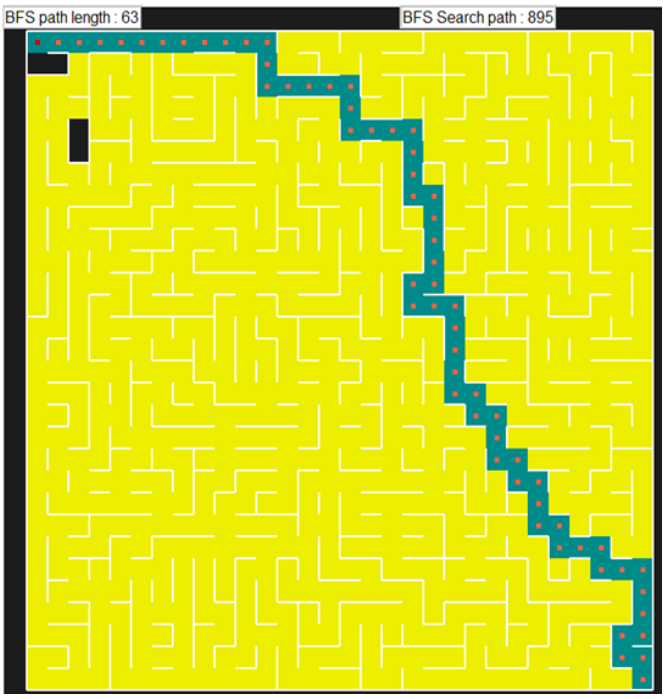


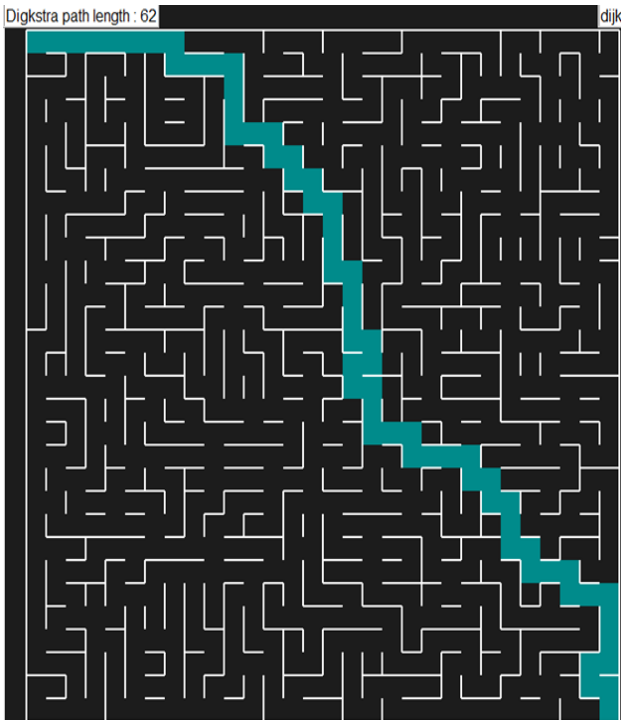
Figure 0-6 Environment 2



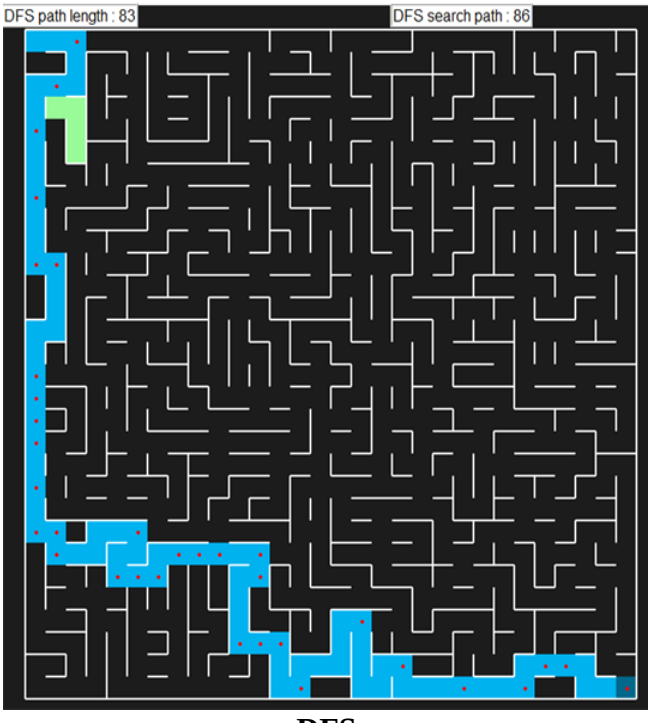
A*



BFS



Dijkstra



DFS

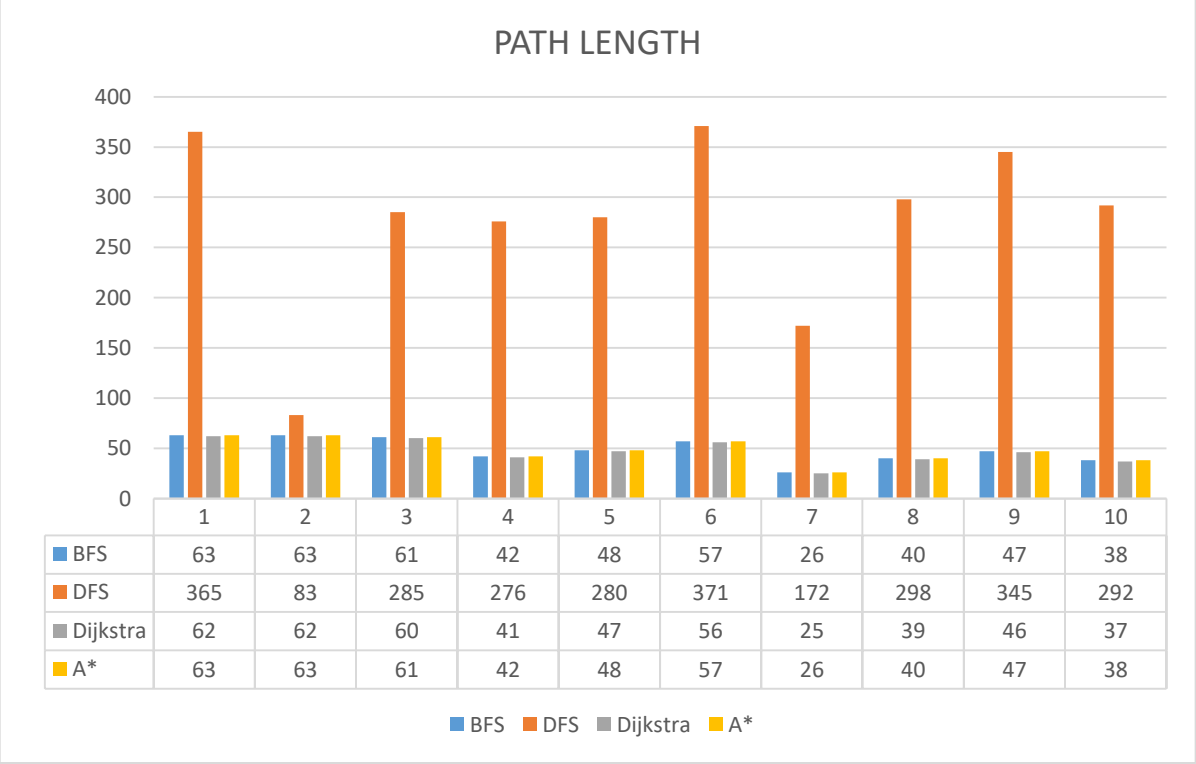
Figure 0-7 Best path found by the four algorithms in environment 2

4.2.5 Comparison of Scenario Number 2

Based on the average path length criterion in Figure 4-8, both BFS and A* algorithms perform similarly with an average path length of 48.5. Dijkstra's algorithm follows closely with an average path length of 47.5, while DFS has a significantly higher average path length of 276.7. Therefore, in terms of path length criterion, Dijkstra's algorithm can be considered the best algorithm based on the average values. It consistently provides the shortest paths on average compared to the other algorithms.

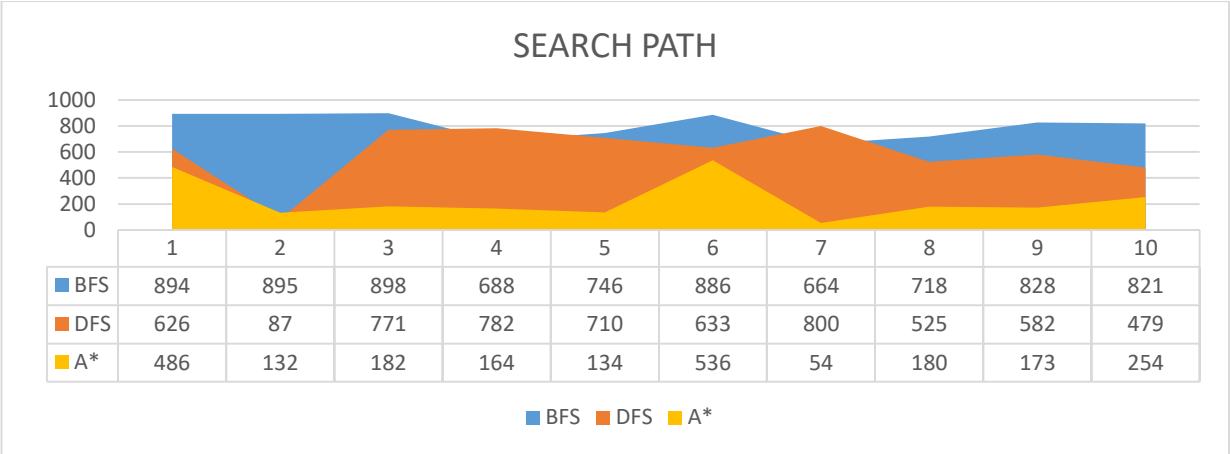
Based on the average search path criterion in Figure 4-9, A* algorithm has the lowest average search path value of 229.5. This indicates that A* algorithm generally explores a more efficient path, involving fewer nodes, compared to the other algorithms in the given scenarios. DFS has the highest average search path value of 599.5, suggesting that it explores a larger number of nodes and may not always find the most optimal path. BFS falls in between with an average search path value of 803.8.

Based on the percentage of total runtime criterion in Figure 4-10, A* algorithm has the lowest percentage at 4.47%. This indicates that A* algorithm generally has faster execution times compared to the other algorithms in the given scenarios. Dijkstra's algorithm has the highest percentage of total runtime at 38.78%, followed by DFS at 33.93%. BFS falls in between with a percentage of 22.82%.



	1	2	3	4	5	6	7	8	9	10
BFS	63	63	61	42	48	57	26	40	47	38
DFS	365	83	285	276	280	371	172	298	345	292
Dijkstra	62	62	60	41	47	56	25	39	46	37
A*	63	63	61	42	48	57	26	40	47	38

Figure 0-8 Path length of Scenario Number 2



	1	2	3	4	5	6	7	8	9	10
BFS	894	895	898	688	746	886	664	718	828	821
DFS	626	87	771	782	710	633	800	525	582	479
A*	486	132	182	164	134	536	54	180	173	254

Figure 0-9 Search path of Scenario Number 2

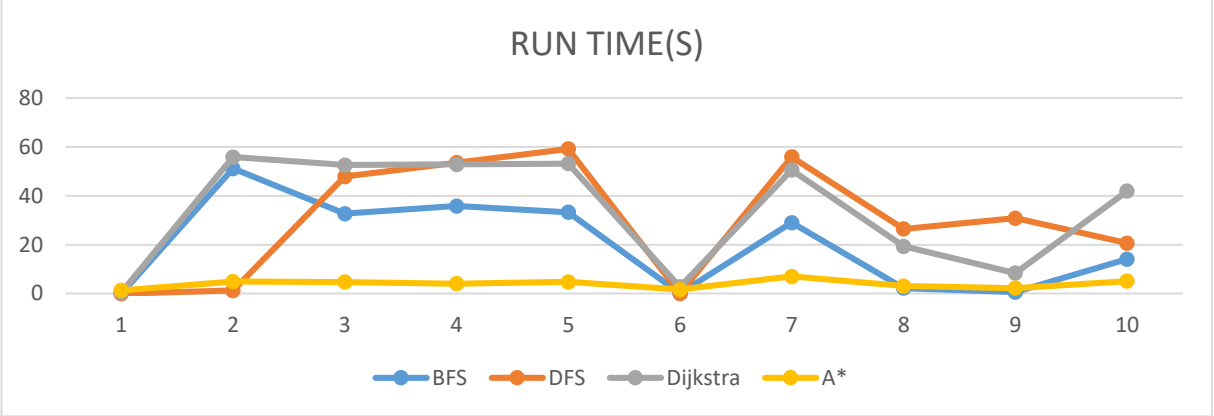


Figure 0-10 Run Time (S) of Scenario Number 2

4.2.6 Scenario Number 3

Simple maze (30×30): environment without obstacle and loopPercent =100%

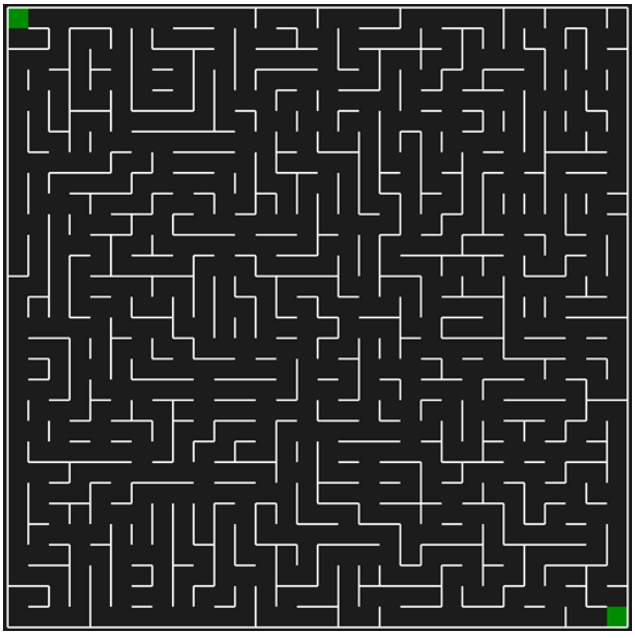
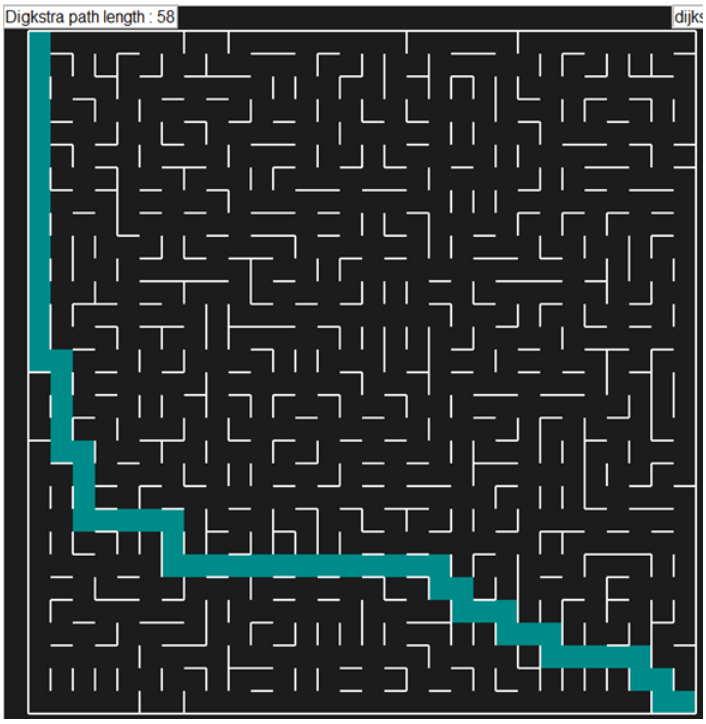
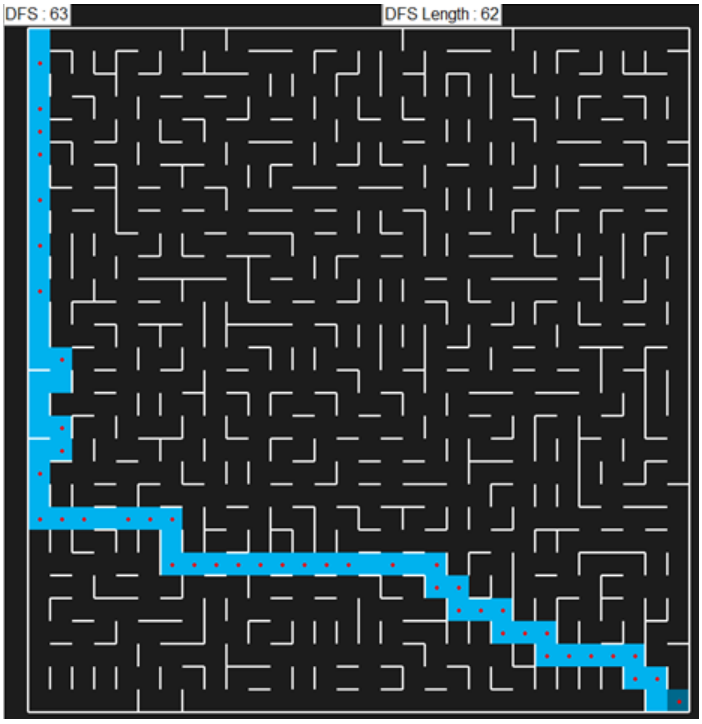


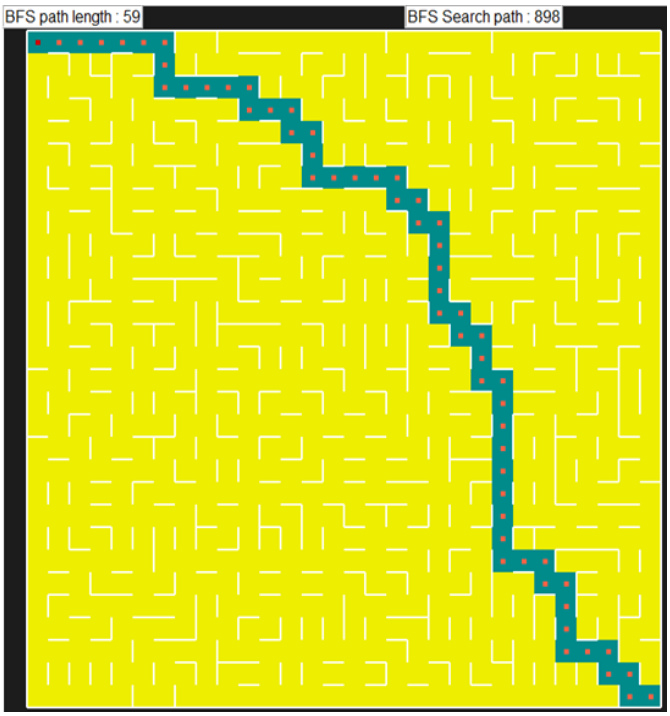
Figure 0-11 Environment 3



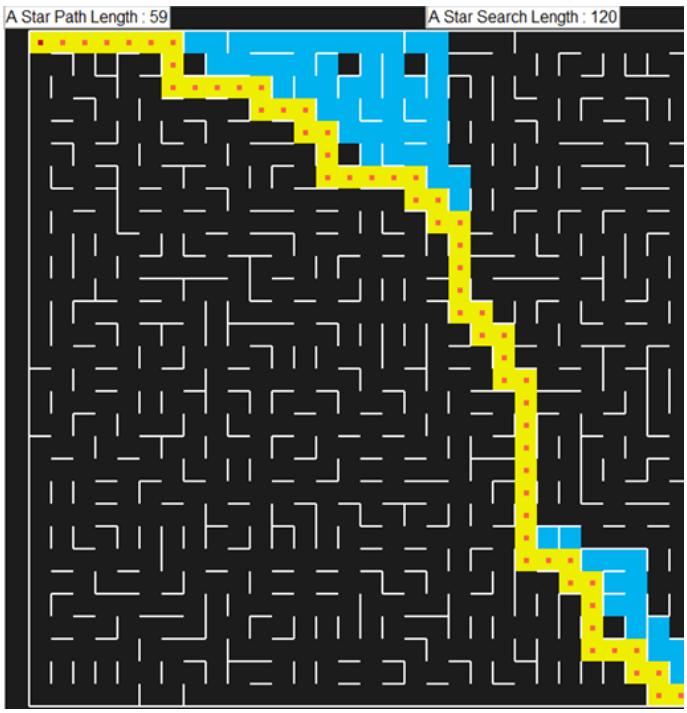
DIJKSTRA



BFS



BFS



A*

Figure 0-12 Best path found by the four algorithms in environment 3

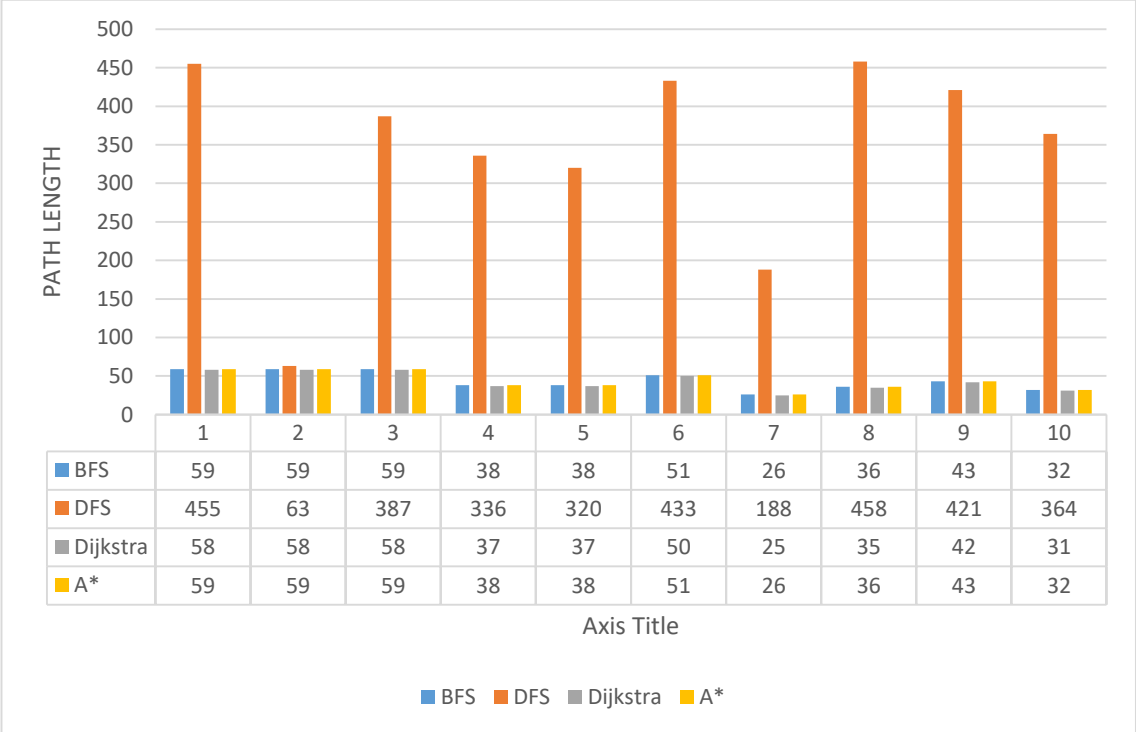
4.2.7 Comparison of Scenario Number 3

When analyzing Figure 4-13, Figure 4-14, and Figure 4-15, we can observe the following:

Path Length: In terms of path length, Dijkstra's algorithm consistently provides the shortest path for most of the cases, with A* algorithm following closely behind. BFS and DFS algorithms generally have longer path lengths.

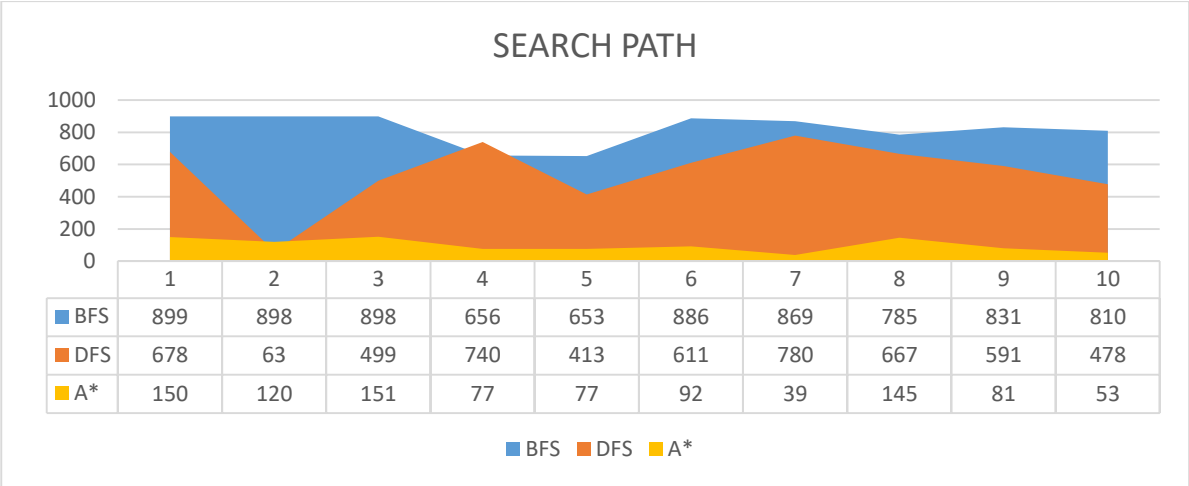
Search Path: A* algorithm consistently explores the smallest number of nodes, indicating a more efficient search path. BFS and DFS algorithms tend to explore a higher number of nodes, which can be less practical in certain scenarios.

Runtime: Dijkstra's algorithm has the highest runtime among the listed algorithms, followed by DFS algorithm. BFS and A* algorithms generally have lower runtimes compared to Dijkstra's and DFS algorithms, the A* algorithm typically executes more quickly than the other methods.



	1	2	3	4	5	6	7	8	9	10
BFS	59	59	59	38	38	51	26	36	43	32
DFS	455	63	387	336	320	433	188	458	421	364
Dijkstra	58	58	58	37	37	50	25	35	42	31
A*	59	59	59	38	38	51	26	36	43	32

Figure 0-13 Path length of Scenario Number 3



	1	2	3	4	5	6	7	8	9	10
BFS	899	898	898	656	653	886	869	785	831	810
DFS	678	63	499	740	413	611	780	667	591	478
A*	150	120	151	77	77	92	39	145	81	53

Figure 0-14 Search path of Scenario Number 3

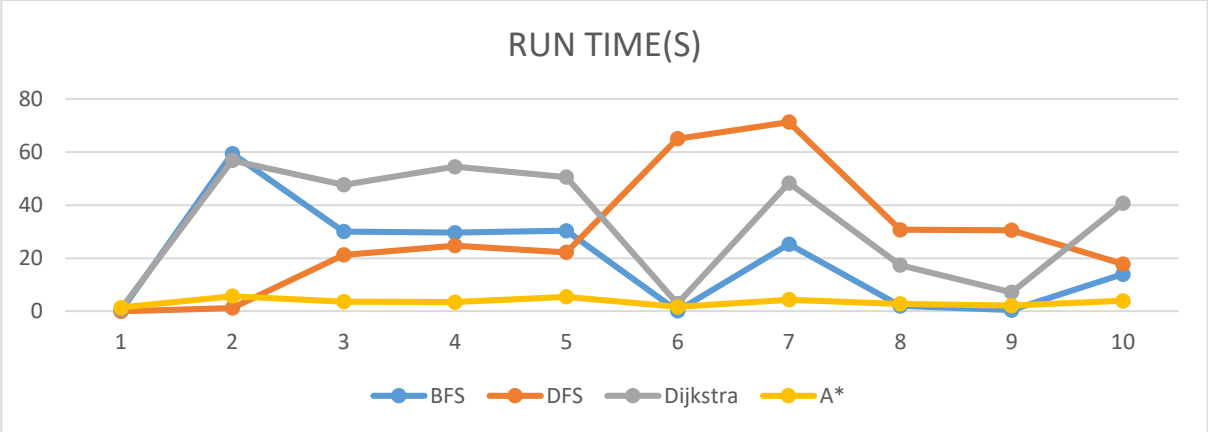


Figure 0-15 Run Time (S) of Scenario Number 3

4.2.8 Scenario Number 4

Complex maze (30×30): environment with obstacles and loopPercent =15%

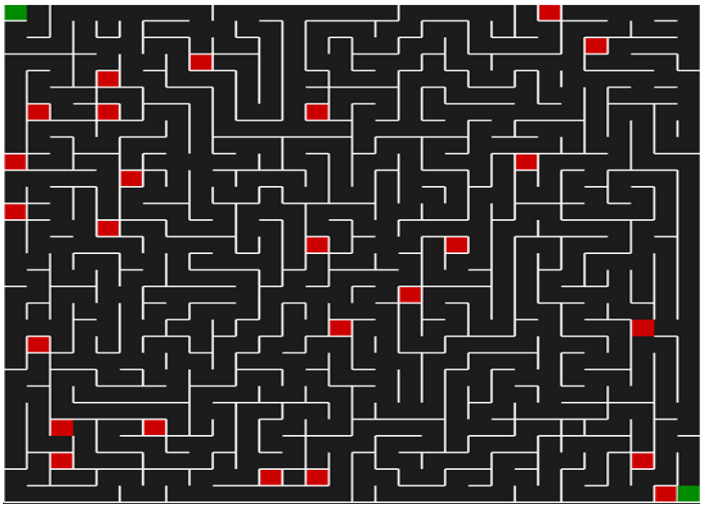
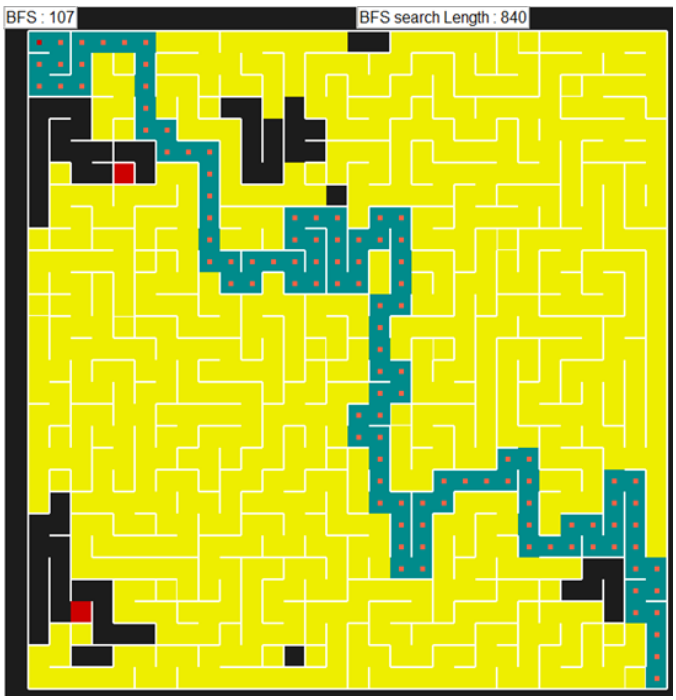
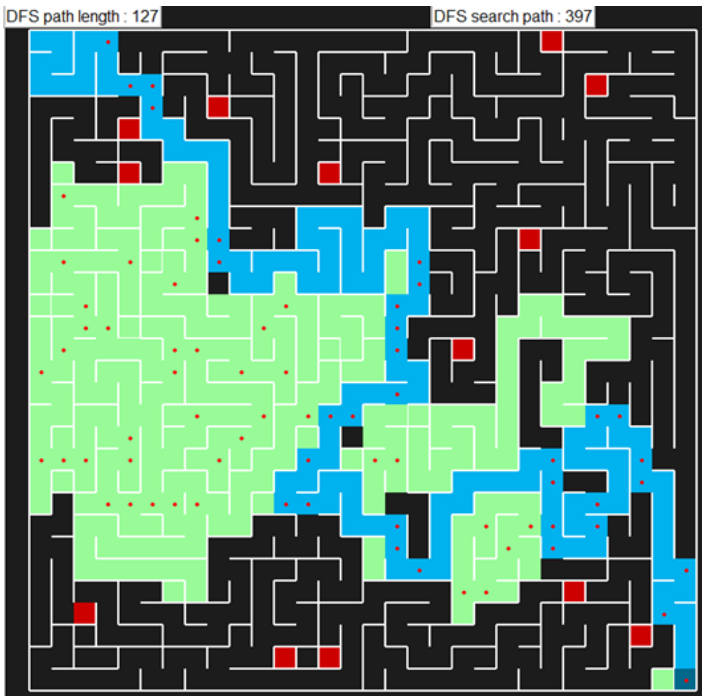


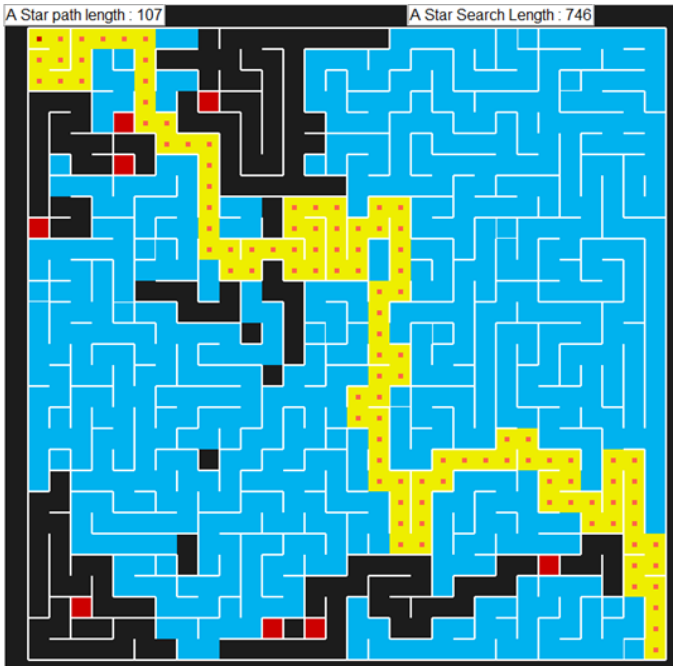
Figure 0-16 Environment 4



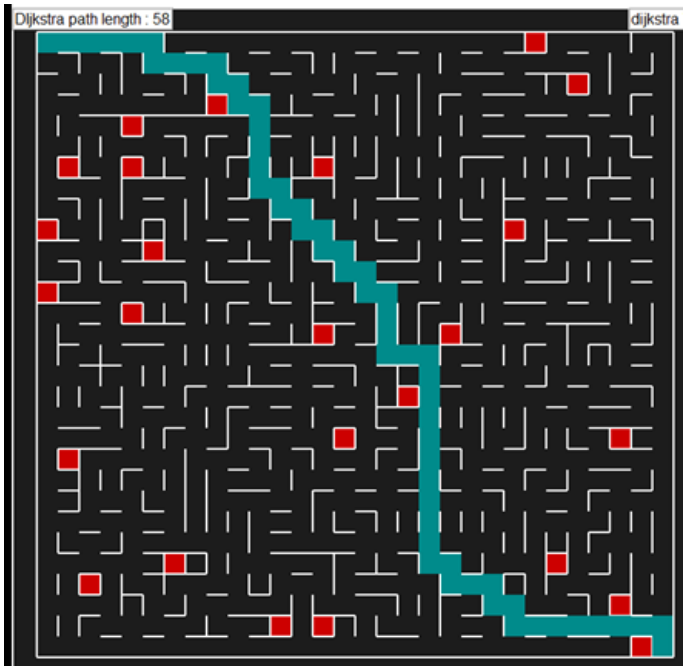
BFS



DFS



A*



DIJKSTRA

Figure 0-17 Best path found by the four algorithms in environment 4

4.2.9 Comparison of Scenario Number 4

PATH		PATH LENGTH				SEARCH PATH		
START	GOAL	A*	Dijkstra	DFS	BFS	A*	DFS	BFS
1.1	30,30,	101	100	163	101	371	803	623
30,30,	1.1	107	106	127	107	746	394	840
30.1	1,30,	103	102	127	103	435	424	783
1.1	1,30,	124	123	146	124	536	649	795
30,30,	1,30,	46	45	58	46	118	711	316
1.4	30.25	101	100	N	101	426	N	642
14.14	20.2	43	N	N	N	159	N	N
18.1	18.28	72	71	110	72	335	811	449
2.5	24.25	77	76	91	77	279	445	368
23.8	8.24	88	87	104	88	624	735	808

N: Not Found

RUN TIME(S)			
A*	Dijkstra	DFS	BFS
1.079210406	0.668680126	0.001877704	0.002357744
16.56250612	51.19480168	10.22869225	39.76945079
3.631693586	28.62813524	28.43441727	6.005179664
3.679875554	27.65288268	28.08866123	6.178019897
3.576389893	28.32674102	28.33728608	6.142223155
1.545573717	2.920613461	N	0.138455617
9.085819903	N	N	N
2.051977484	10.15082327	37.84563311	0.99136396
2.007511259	9.050199783	0.60899292	0.890699883
5.741358157	30.563271	34.94968406	7.133944928

Table 0-3 Simulation data for Scenario 4

Considering the updated Table 4-3, let's analyze the runtime, search path, path length, and the actual path for each algorithm:

BFS:

- The runtime for BFS varies across different scenarios, ranging from 0.002 to 39.769 seconds.
- The search path varies from 316 to 840, indicating that BFS explores a moderate to large number of nodes.
- The path lengths range from 46 to 124.

DFS:

- The runtime for DFS varies across different scenarios, ranging from 0.608 to 37.846 seconds.
- The search path varies from 394 to 811, indicating that DFS explores a moderate to large number of nodes.
- The path lengths range from 58 to 163.
- Not all paths were found by DFS (indicated by "N" in the table).

Dijkstra's algorithm:

- The runtime for Dijkstra's algorithm varies across different scenarios, ranging from 0.668 to 51.195 seconds.
- The path lengths range from 45to 123.

A star:

- The runtime for A* algorithm varies across different scenarios, ranging from 1.08 to 16.563 seconds.
- The search path varies from 118 to 746, indicating that A* algorithm explores a moderate number to large number of nodes.
- The path lengths range from 43 to 124.

Analyzing both the search path and path length, it seems that A* algorithm provides a good balance of efficiency and optimality.

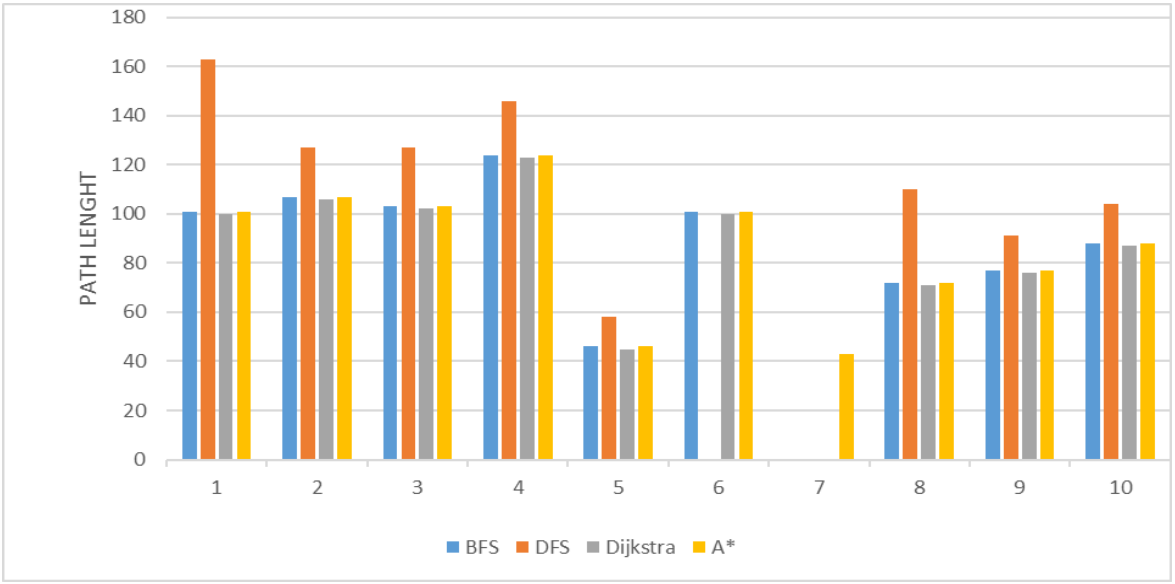


Figure 0-18 Path length of Scenario Number 4

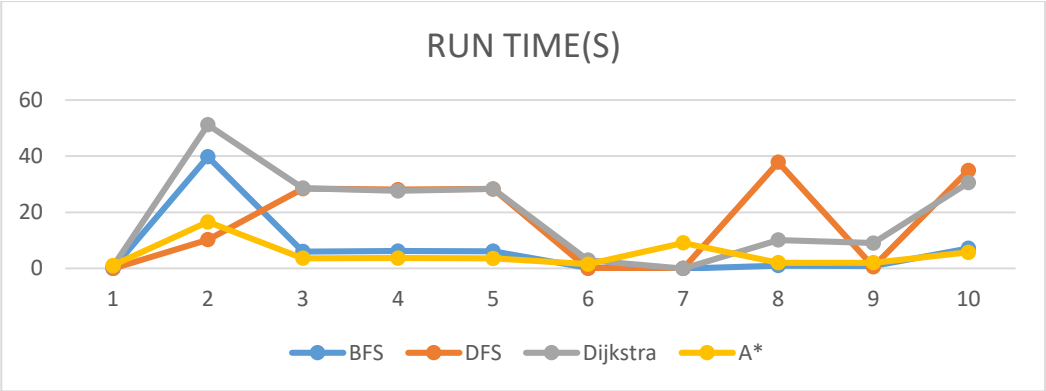


Figure 0-19 Run Time (S) of Scenario Number 4

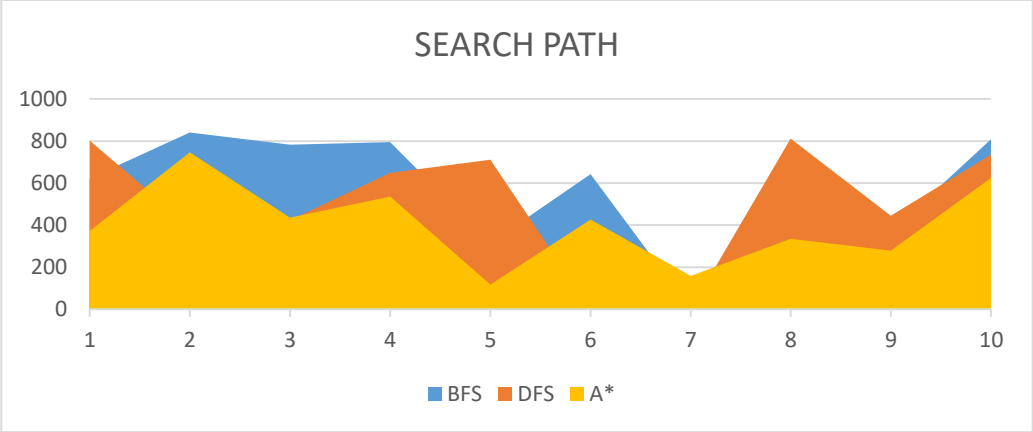


Figure 0-20 Search path of Scenario Number 4

4.2.10 Scenario Number 5

Complex maze (30×30): environment with obstacles and loopPercent =50%

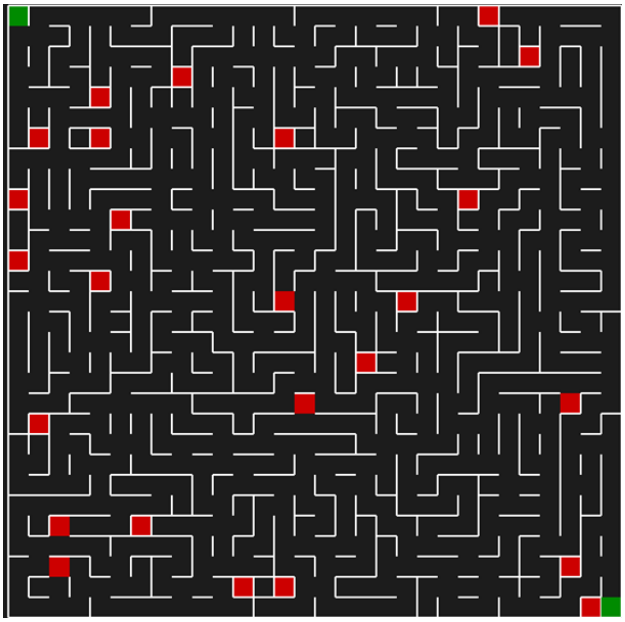


Figure 0-21 Environment 5

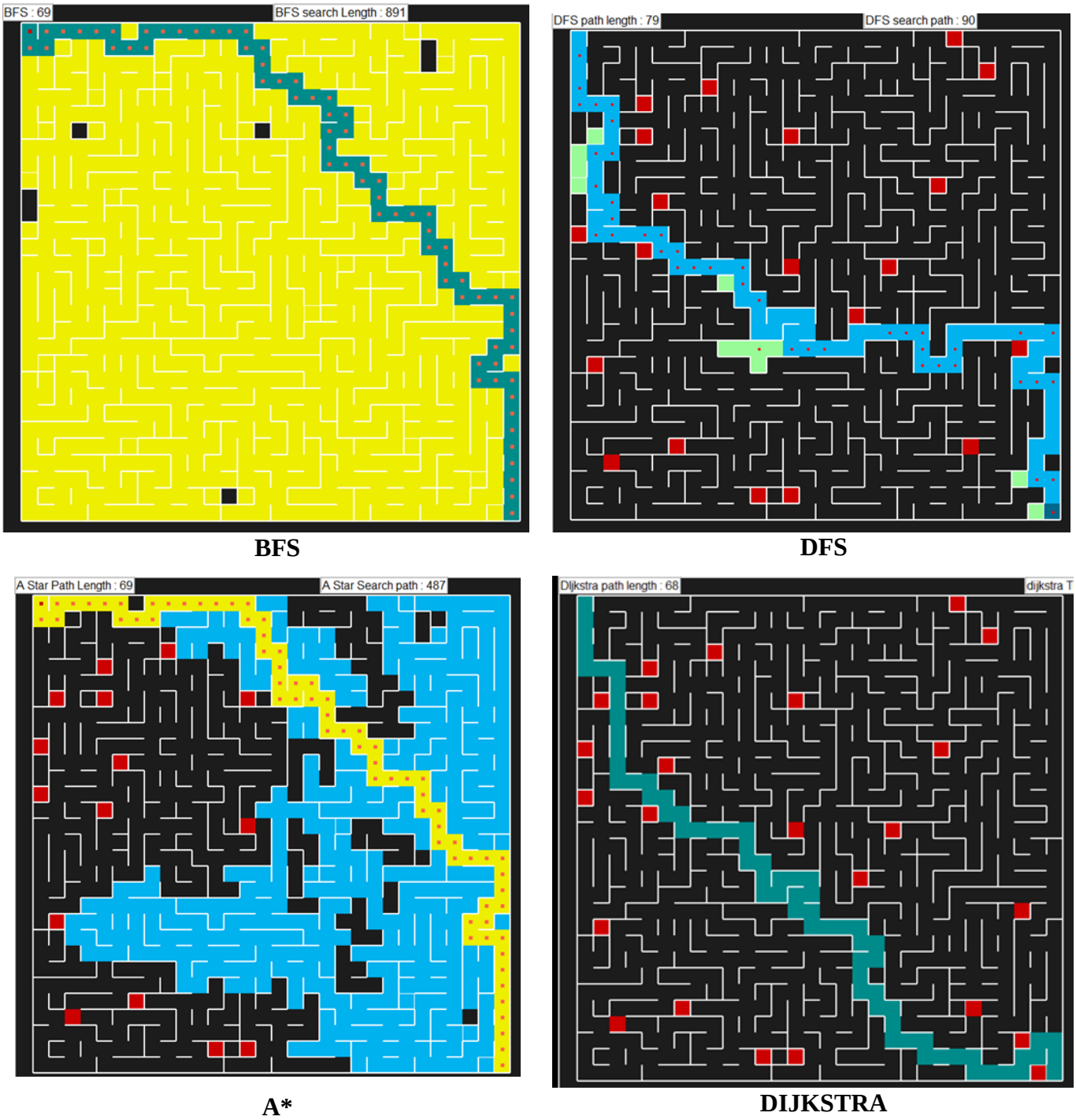


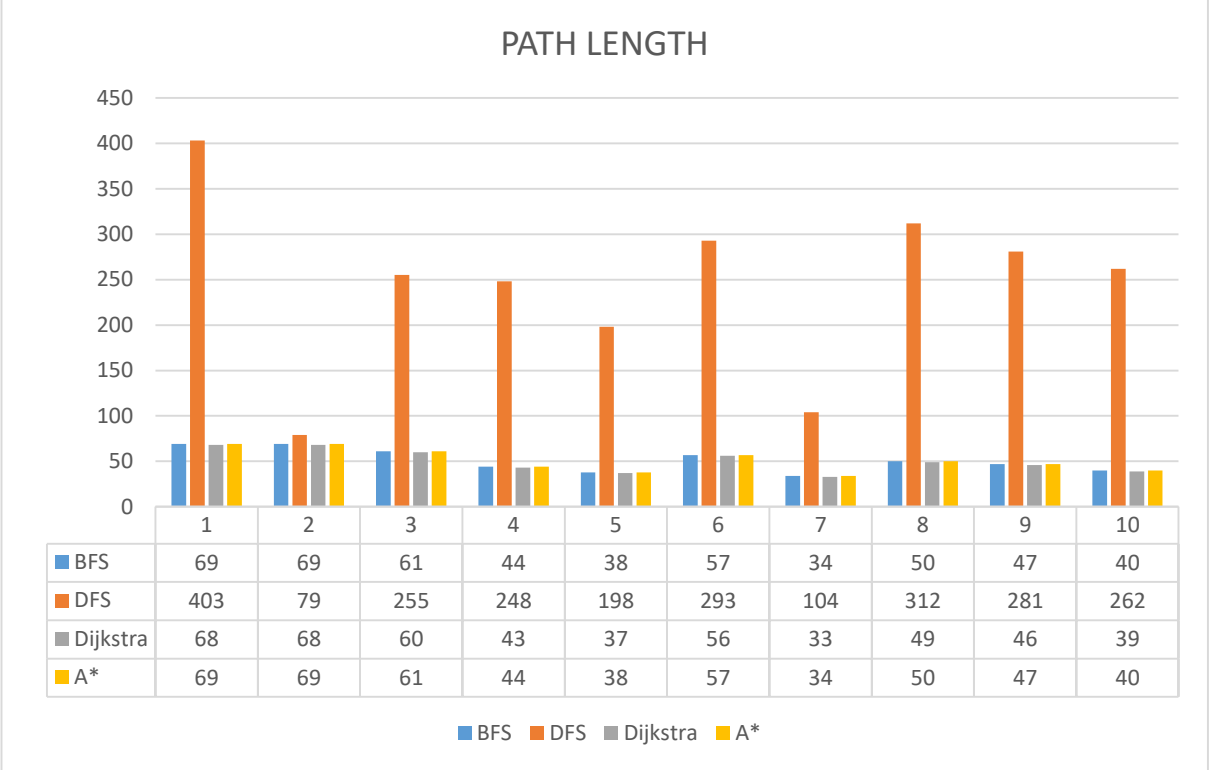
Figure 0-22 Best path found by the four algorithms in environment 5

4.2.11 Comparison of Scenario Number 5

Considering the path lengths in Figure 4-23, it appears that Dijkstra's finds shorter paths in some cases, while BFS algorithm and A* algorithm find comparable path lengths. DFS generally finds longer paths.

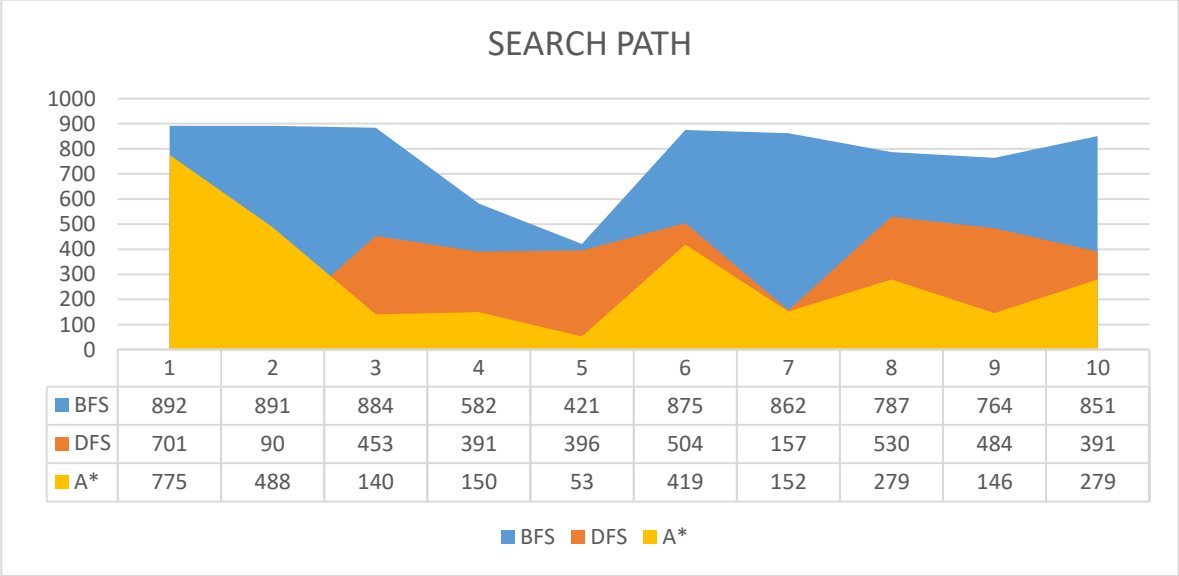
Analyzing the search path in Figure 4-24, A* explores a slightly smaller number of nodes compared to BFS, and DFS algorithm.

In terms of runtime in Figure 4-25, BFS and DFS are generally faster, while Dijkstra's algorithm takes more time. A* algorithm is faster than them.



	1	2	3	4	5	6	7	8	9	10
BFS	69	69	61	44	38	57	34	50	47	40
DFS	403	79	255	248	198	293	104	312	281	262
Dijkstra	68	68	60	43	37	56	33	49	46	39
A*	69	69	61	44	38	57	34	50	47	40

Figure 0-23 Path length of Scenario Number 5



	1	2	3	4	5	6	7	8	9	10
BFS	892	891	884	582	421	875	862	787	764	851
DFS	701	90	453	391	396	504	157	530	484	391
A*	775	488	140	150	53	419	152	279	146	279

Figure 0-24 Search path of Scenario Number 5

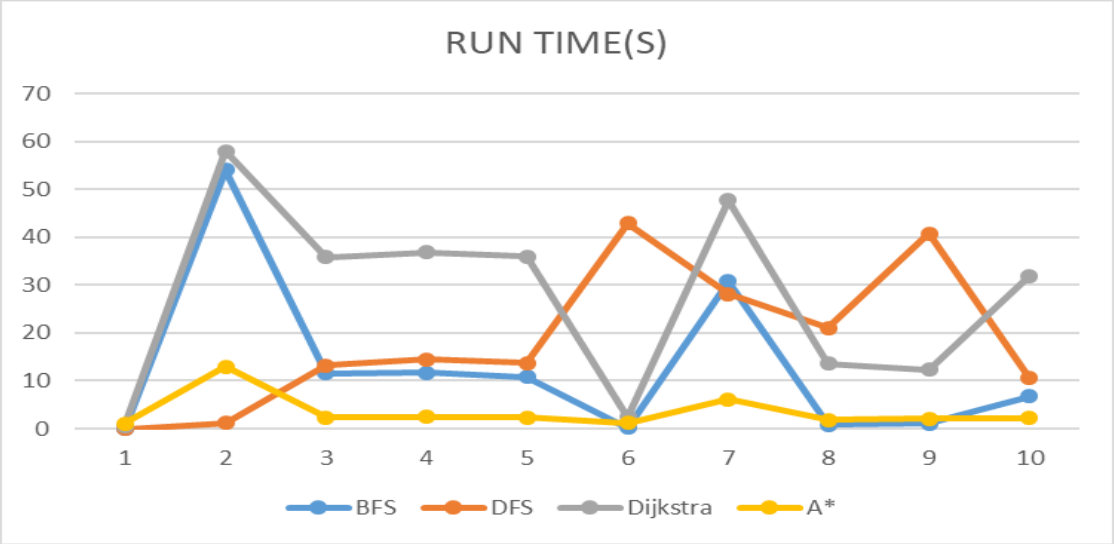


Figure 0-25 Run Time (S) of Scenario Number 5

4.2.12 Scenario Number 6

Complex maze (30×30): environment with obstacles and loopPercent =100%

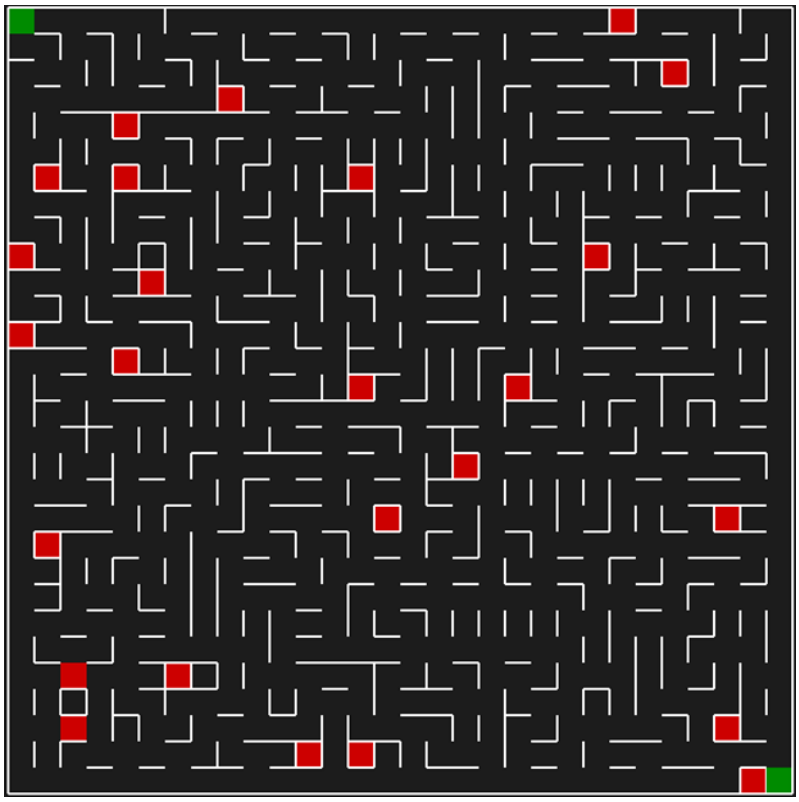
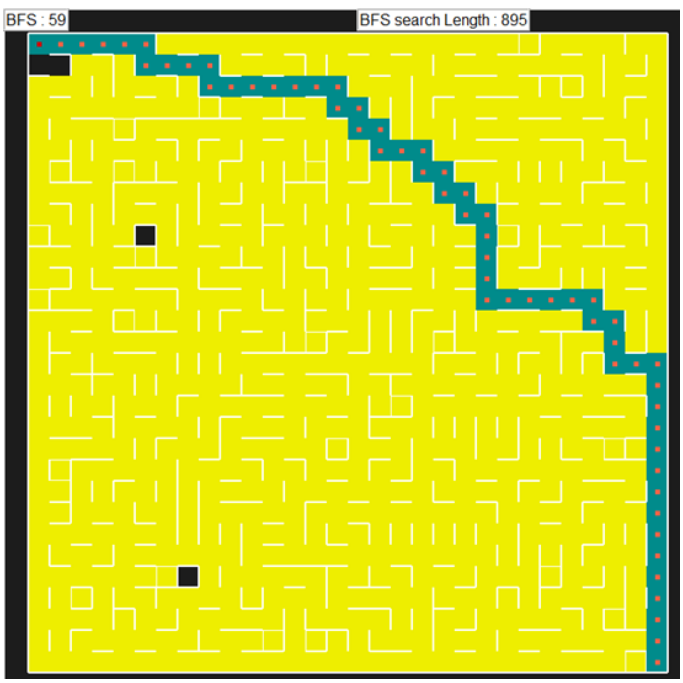
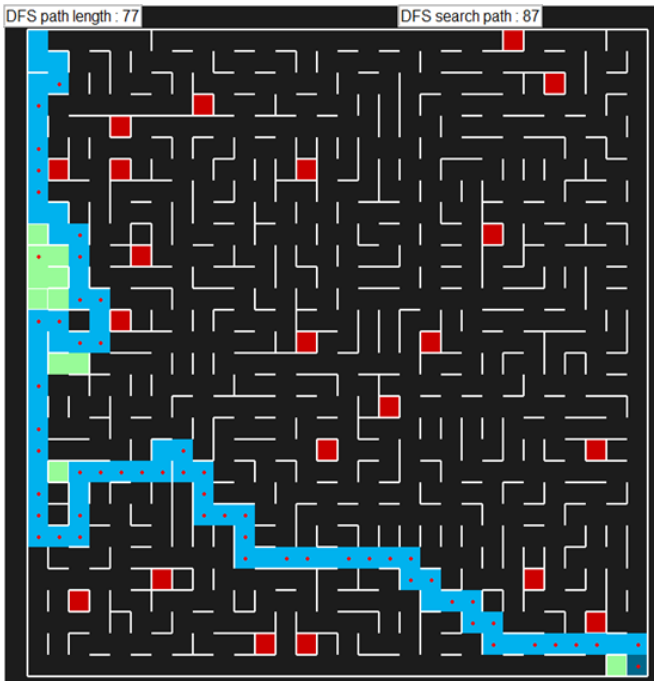


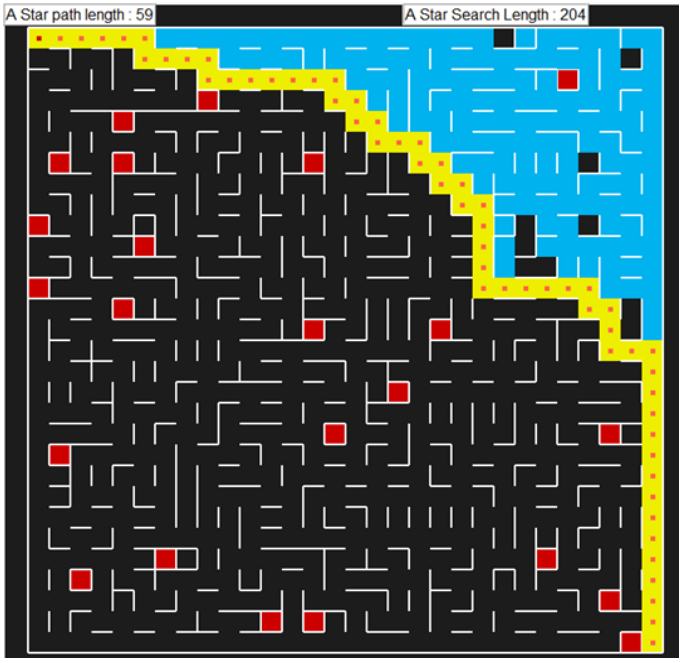
Figure 0-26 Environment 6



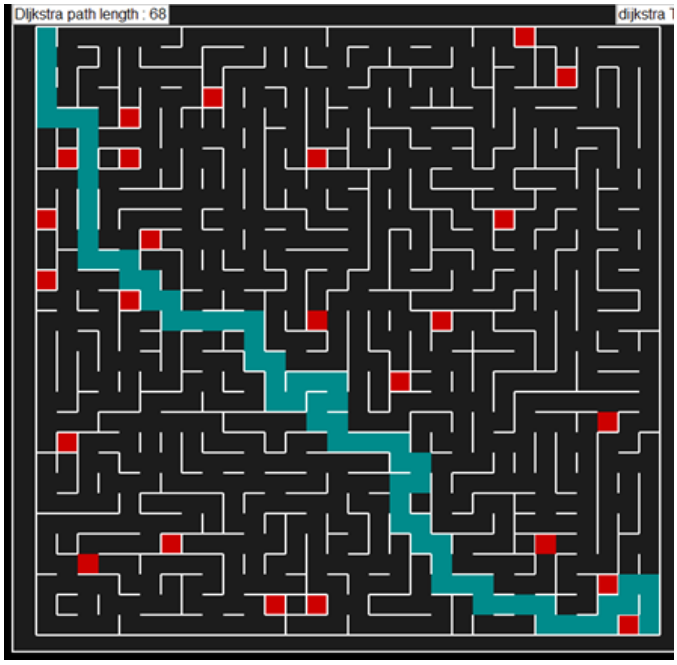
BFS



DFS



A*



DIJKSTRA

Figure 0-27 Best path found by the four algorithms in environment 6

4.2.13 Comparison of Scenario Number 6

As shown in Figure 4-28, Figure 4-29 and Figure 4-30**,** we can see that:

BFS:

- The BFS algorithm finds paths with lengths ranging from 26 to 59.
- The runtime for BFS varies across different scenarios, ranging from 0.002 to 56.007 seconds.
- The search path ranges from 445 to 897, indicating that BFS explores a moderate to large number of nodes.

DFS:

- The DFS algorithm finds paths with lengths ranging from 31 to 461.
- The runtime for DFS varies across different scenarios, ranging from 0.002 to 92.521 seconds.
- The search path ranges from 87 to 870, indicating that DFS explores a moderate to large number of nodes.

Dijkstra's algorithm:

- Dijkstra's algorithm finds paths with lengths ranging from 29 to 58.
- The runtime for Dijkstra's algorithm varies across different scenarios, ranging from 0.657 to 53.780seconds.

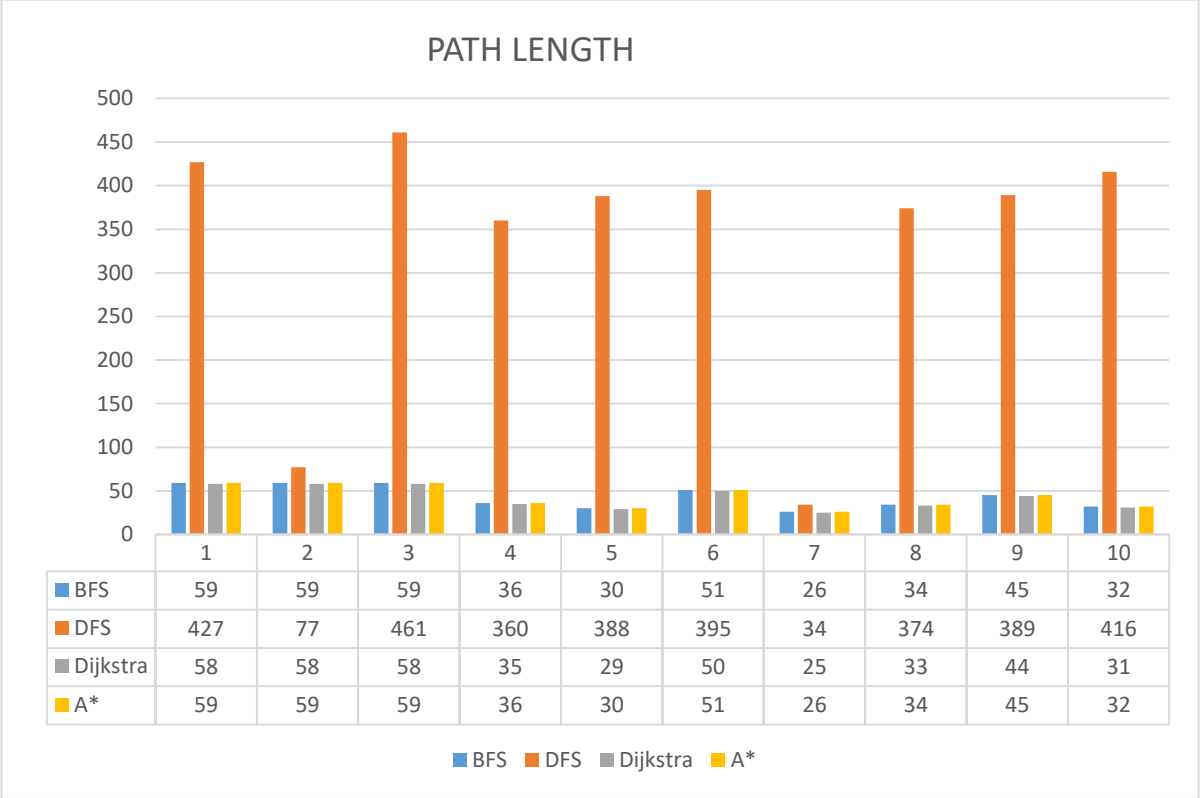
A*:

- The A* algorithm finds paths with lengths ranging from 26 to 59.
- The runtime for the A* algorithm varies across different scenarios, ranging from 1.266 to 6.41 seconds.
- The search path ranges from 30 to 204, indicating that A* explores a moderate to small number of nodes.

Considering the path lengths, DFS finds the longest paths in most cases, while BFS algorithm and A* algorithm find comparable path lengths. Dijkstra's is the faster one.

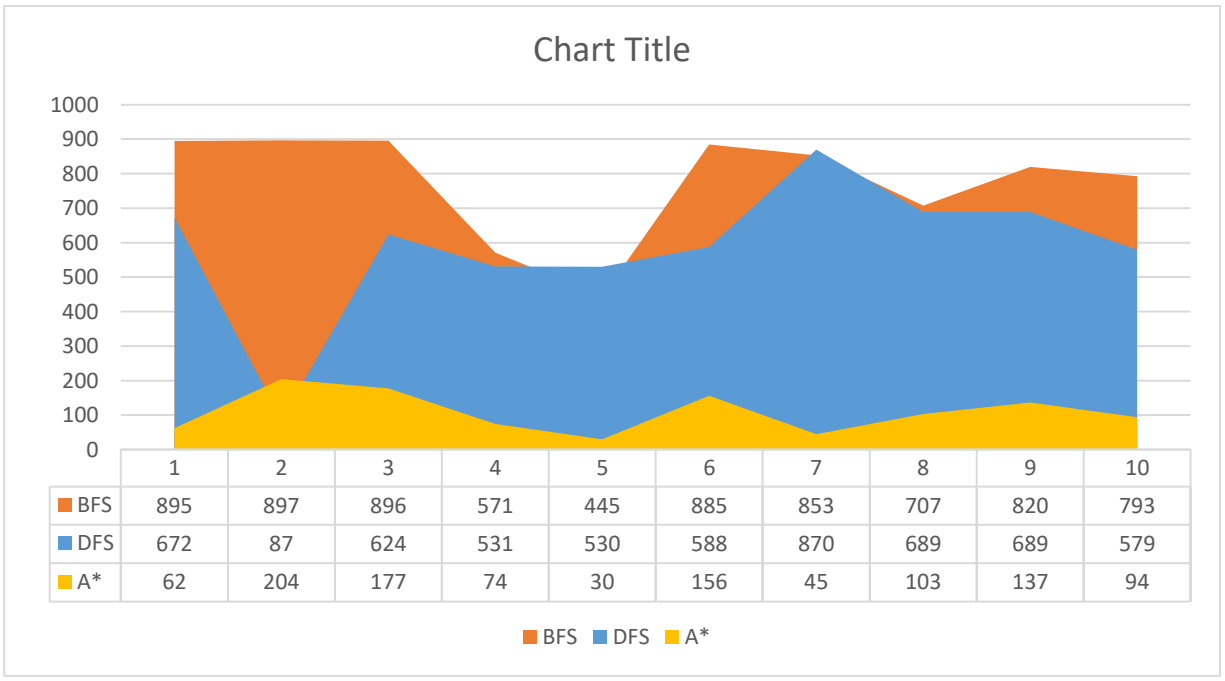
In terms of runtime, BFS and Dijkstra's algorithm generally have faster execution times, while DFS and A* algorithm take more time.

Analyzing the search path, A* algorithm explore a similar number of nodes, while BFS explores a slightly larger number of nodes. DFS algorithm explores a moderate number of nodes.



	1	2	3	4	5	6	7	8	9	10
BFS	59	59	59	36	30	51	26	34	45	32
DFS	427	77	461	360	388	395	34	374	389	416
Dijkstra	58	58	58	35	29	50	25	33	44	31
A*	59	59	59	36	30	51	26	34	45	32

Figure 0-28 Path length of Scenario Number 6



	1	2	3	4	5	6	7	8	9	10
BFS	895	897	896	571	445	885	853	707	820	793
DFS	672	87	624	531	530	588	870	689	689	579
A*	62	204	177	74	30	156	45	103	137	94

Figure 0-29 Search path of Scenario Number 6

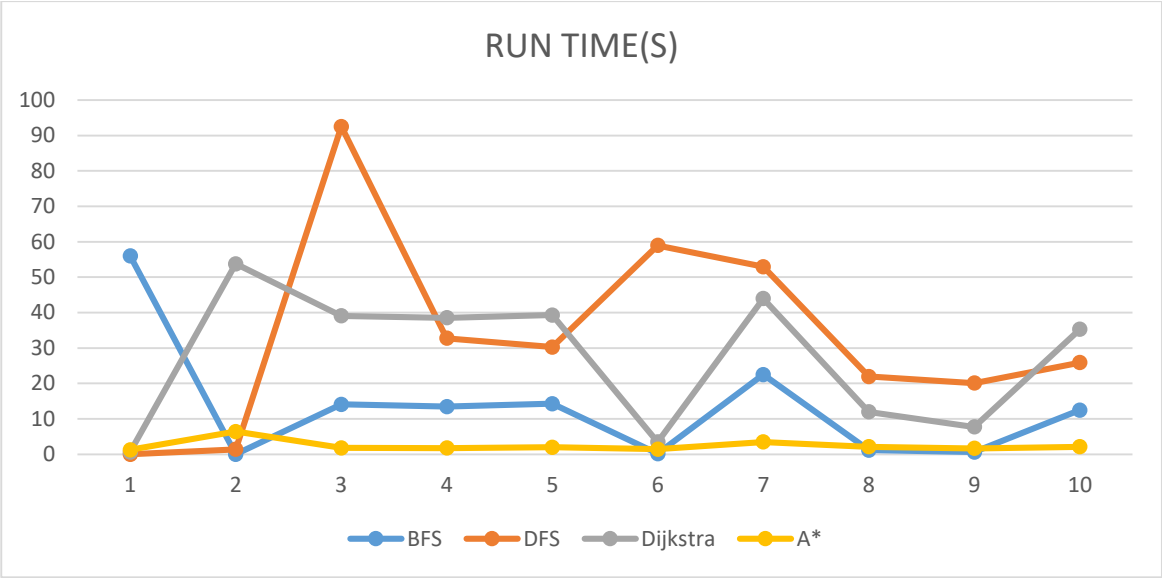


Figure 0-30 Run Time (S) of Scenario Number 6

4.3 Discussion

Given the foregoing, the A* method routinely outperforms the competition in terms of runtime and path length. This is so that A* can concentrate its search on the most promising locations by using a heuristic function to estimate the cost of the remaining path.

The other algorithms, such as BFS, DFS, and Dijkstra's algorithm, do not employ a heuristic function, necessitating an exhaustive search before the shortest path is discovered. This can take a long time, particularly for huge graphs.

In addition to being faster, A* is also more robustness than the other algorithms. This is because A* is guaranteed to find the shortest path, while the other algorithms may not find the shortest path if the graph contains loops or other obstacles.

Overall, due to its speed, dependability, and simplicity of use, A* is the greatest option for pathfinding issues.

4.4 Supplement

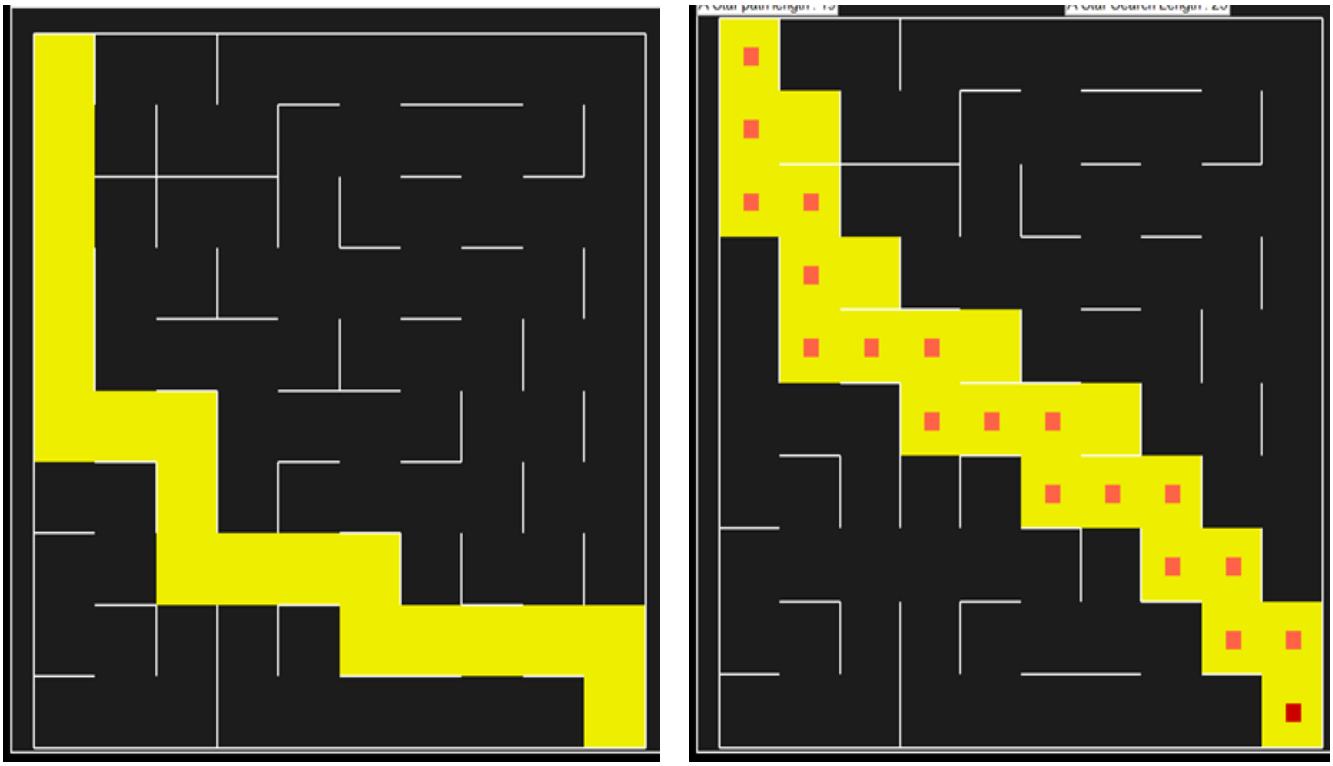
We choose one of the AI algorithms because it relates to the field we work in, and we chose GA to contrast it with A*, which was chosen as the best algorithm based on prior standards and values.

Due to time constraints we compared to a maze of (10.10) celles and another with (15.15) without obstacles and loopPercent =70% to 10 values and the comparison results are shown in the Table 4-4

Here is a brief overview of how GA is applied to find the shortest path:

1. Initialize a population of solutions. The population can be initialized randomly or using a heuristic function.
2. Evaluate the fitness of each solution. The fitness of a solution is a measure of how close it is to the shortest path.
3. Select the fittest solutions. The fittest solutions are selected to participate in the next generation.
4. Crossover. Two fittest solutions are randomly selected and their genes are crossed over to create new solutions.
5. Mutation. Some of the genes in the new solutions are randomly mutated.
6. Repeat steps 2-5 until a satisfactory solution is found.

We chose 1.1 as the starting point and 10.10 as the ending point to view the results of two algorithms. as shown in Figure 4-31.



GA A*

Figure 0-31 Best path found by the two algorithms A* and GA

4.4.1 Comparison of two algorithms

PATH		PATH LENGH		RUN TIME		Completeness		Generation
START	GOAL	A*	GA	A*	GA	A*	GA	GA
2.4	10.10.	17	19	0.17516493	0.096632242	yes	yes	37
1.1	10.10.	19	19	0.16725618	1.903086185	yes	yes	118
3.5	10.10.	15	23	0.17650522	0.060582638	yes	yes	40
4.9	10.10.	10	21	0.19313679	0.299594402	yes	yes	45
1.10.	10.1	12	19	0.16731543	0.123816967	yes	yes	66
1.1	15.15	29	29	0.31905826	155.4452035	yes	yes	83
2.3	15.15	26	37	0.32888344	30.36268878	yes	yes	667
3.4	15.15	24	31	0.32550264	11.90847039	yes	yes	1219
4.4	15.15	25	29	0.32829377	56.88920903	yes	yes	1926
5.6	15.15	22	29	0.30906888	4.695002556	yes	yes	464
7.8	15.15	18		0.50844991	29.55758047	yes	No	2499

Table 0-4 Simulation data for the two algorithms A* and GA

To determine which algorithm performs better in each criterion, we can analyze the provided table. Here's a breakdown of the criteria and the better algorithm in each case:

Completeness:

A* performs better in terms of completeness since it has "yes" in the Completeness column for all entries, indicating that it found a solution in each case. The Genetic Algorithm (GA) has one entry marked as "No," indicating it did not find a solution for that particular scenario.

Run Time:

A* performs better in terms of run time in most cases. The run time values for A* algorithm are consistently lower compared to GA. However, it's worth noting that there are a few cases where GA has a lower run time than A*. If faster execution time is the priority, A* is generally the better choice.

Path Lengths:

we can see that A* generally achieves shorter path lengths compared to GA in most cases. However, it's important to note that there are a few cases where GA has the same or slightly shorter path lengths.

Given this information, we can conclude that, on average, the A* algorithm tends to perform better in terms of path length, as it consistently achieves shorter paths compared to GA. The percentages at the bottom of the table, 33.36% and 28.28%, do not provide clear information regarding the comparison of path length between the two algorithms.

Based on this analysis, we can conclude that, on average, the A* algorithm tends to achieve shorter path lengths compared to GA. It outperforms GA in a majority of the cases, with only a few instances where GA achieves the same or slightly shorter path lengths.

Therefore, if minimizing path length is a critical criterion, A* algorithm would generally be considered the better choice based on the provided data.

As can be seen, GA and A* both find the shortest path in most cases. However, GA fails to find a path in one case, while A* always finds a path. GA is also slower than A* in most cases.

Overall, A* is a better choice for finding the shortest path than GA. It is more reliable, faster, and easier to implement.

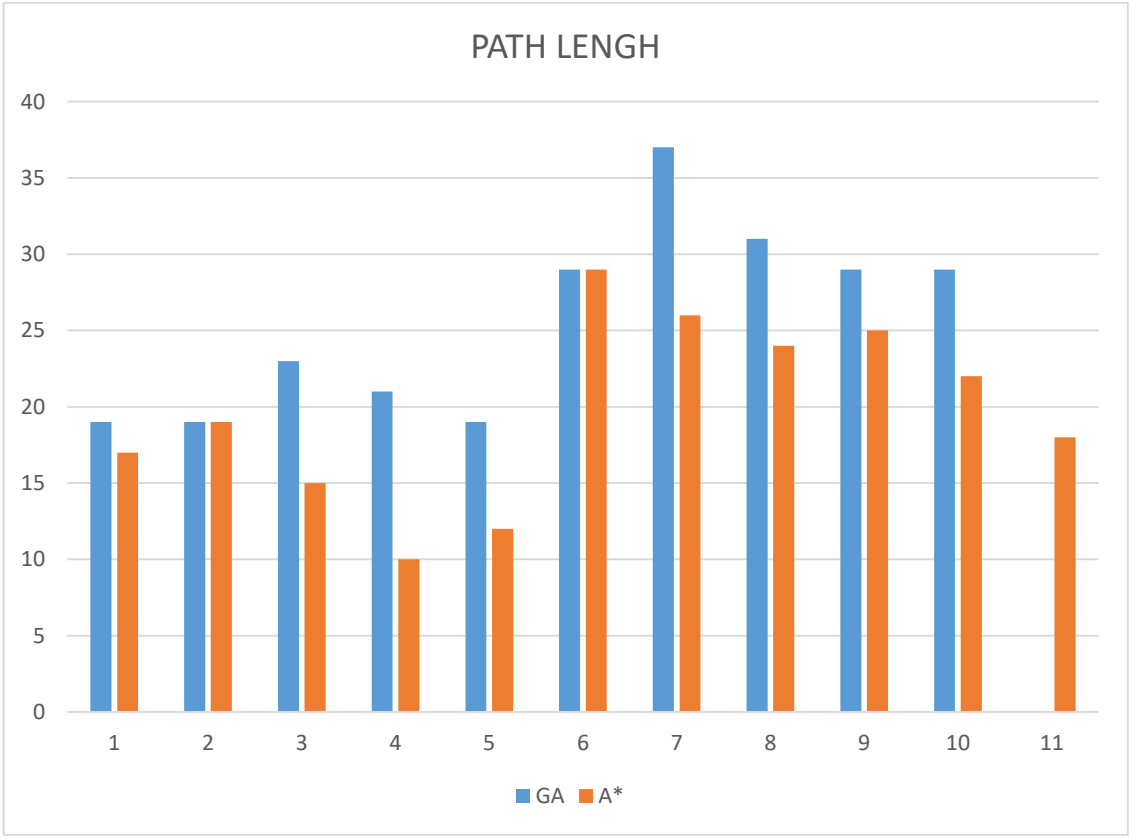


Figure 0-32 Path Length of the two algorithms A* and GA

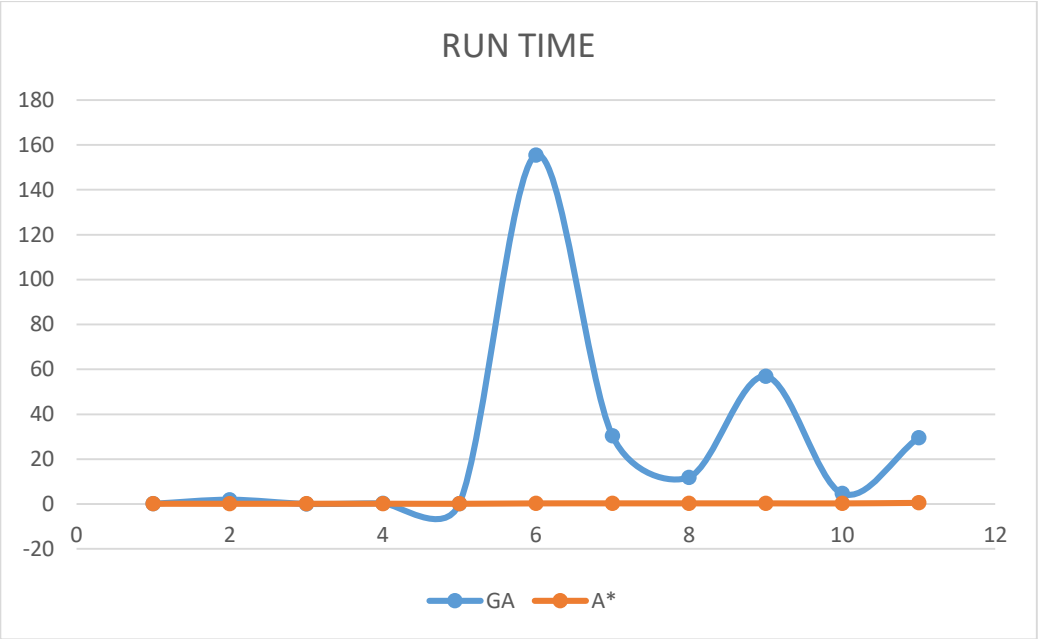


Figure 0-33 Runtime (s) of the two algorithms A* and GA

When comparing the performance of the A* algorithm and the Genetic Algorithm (GA) in finding the shortest path, it is evident that A* outperforms GA in many cases. To clarify the reasons behind this, we can consider the following:

- Heuristic Information: A* utilizes heuristic information in the form of an estimated cost to the goal from each state. This heuristic guides the search algorithm to prioritize exploration towards more promising paths, resulting in a more efficient search. The heuristic helps A* focus its search on the most likely optimal path, whereas GA relies on genetic operators without explicit guidance based on heuristics.
- Search Efficiency: A* combines both breadth-first and best-first search strategies by considering both the cost to reach the current state and the estimated cost to the goal. This combination allows A* to quickly explore promising paths while still considering alternative paths that may lead to a shorter path. GA, on the other hand, typically explores a larger portion of the search space due to its evolutionary nature, which can lead to a slower convergence towards the optimal solution.
- Problem Representation: A* operates on a graph representation of the problem, where nodes represent states, and edges represent transitions between states. This graph representation allows A* to take advantage of the problem's structure and exploit efficient search algorithms. GA, on the other hand, typically uses a genetic encoding

scheme that may not be as directly suited to the problem domain, requiring additional translation steps or adaptations.

- Convergence and Diversity: A* is designed to converge towards an optimal solution by exploring the most promising paths first. In contrast, GA is an evolutionary algorithm that maintains diversity within the population to avoid premature convergence. This focus on maintaining diversity may result in longer convergence times for GA compared to A*.
- Problem Complexity: A* is well-suited for problems with well-defined heuristics and relatively smaller search spaces. It is particularly effective in problems where the heuristic provides accurate estimates of the remaining cost to the goal. GA, on the other hand, is more suitable for problems where the search space is large, and the optimal solution is not well-defined or has complex interactions.

One of the advantages of genetic algorithms is their ability to find novel and creative solutions to complex problems. They are also able to handle non-differentiable and non-convex cost functions. However, they can be computationally expensive and require a large number of evaluations to converge to a satisfactory solution.

Based on the above, it can be concluded that A* is generally better than GA in the context of finding the shortest path. It is important to note that this analysis is based on the provided data in the specific table. Further analysis or experimental testing may be required for a more comprehensive and accurate evaluation of the two algorithms.

It's important to note that the performance of GA and A* can vary depending on the specific problem instance, problem size, heuristics used, and other factors. Therefore, it is recommended to analyze and compare the performance of both algorithms on a case-by-case basis to determine which algorithm is better suited for a particular problem.

Conclusion and Future Work

Conclusion

As robotics technology continues to advance, the future of path planning for mobile robots looks bright. New algorithms and techniques are being developed all the time, and the possibilities for real-world applications are virtually limitless.

Mobile robots, with their capacity to navigate complex settings, avoid obstacles, and arrive at their destinations promptly and safely, are poised to transform a wide range of industries and applications. The future of path planning for mobile robots is sure to be exciting and full of new possibilities, whether it's self-driving vehicles, drones, or warehouse automation.

Path planning is essential for mobile robots navigating in an unfamiliar environment. It is an important component of robots and autonomous systems, allowing them to navigate complex situations and complete tasks efficiently.

To address this difficulty, several methods and techniques have been created.Path planning approaches each have advantages and disadvantages, and their usefulness is determined by the individual application needs and restrictions.

Engineers and researchers can select the most suited method for their unique application and optimize the performance of their mobile robots by knowing the benefits and limitations of each algorithm.

Efficient path planning can assist improve mobile robot performance and production, resulting in cost savings and increased competitiveness. It is critical to select the appropriate path planning algorithm depending on the environment and task requirements.

Machine learning algorithms can be used to increase the accuracy and efficiency of maze solving algorithms by learning from previous encounters. Neural networks and genetic algorithms, for example, can be utilized to develop more sophisticated and flexible algorithms.

We investigated some of the most common path planning algorithms and evaluated their performance in solving maze challenges in this comparative study. We hope that this research provides useful insights into the topic of path planning and encourages further research and improvement in this area.

Finally, this thesis gave a complete comparative analysis on path planning algorithms for mobile robots, with a focus on the Maze Problem simulation environment. The research has

considerably improved our understanding of path planning algorithms and provided practical direction for implementing them in real-world applications.

This research has shed light on the performance, strengths, and limitations of algorithms such as A*, BFS, DFS, Dijkstra's algorithm, and the Genetic algorithm (GA) by comparing and assessing them. Path length, running time, robustness, search path, and temporal complexity were used as evaluation criteria, and they gave a full review of each algorithm's effectiveness and efficiency.

The experimental results from testing the algorithms in numerous scenarios have provided useful insights into their performance in a variety of environments and conditions. These insights can be used by researchers and practitioners to make educated judgments when picking the best path planning algorithm for their individual applications.

Discussion and Future Directions

With new breakthroughs in robotics and artificial intelligence, the future of path planning for mobile robots is bright. Developing algorithms that can manage more complicated and dynamic situations, such as metropolitan environments with pedestrians and automobiles, is one field of research. Another area of study is the creation of algorithms capable of learning from experience and adapting to changing circumstances.

Future research and investigation should focus on the following topics:

Real-world settings: This investigation concentrated on simulation environments, notably the Maze Problem. Extending the analysis to include real-world scenarios such as dynamic and uncertain surroundings, variable topography, and complicated obstructions would be beneficial. The performance of the algorithms in these realistic conditions would provide a more accurate knowledge of their applicability and robustness.

Hybrid approaches: Future study could benefit from investigating the potential of merging several path planning algorithms or blending classical and learning-based methods. Hybrid techniques can leverage the characteristics of various algorithms to increase overall performance and flexibility in difficult settings.

Advanced machine learning and deep learning techniques may be explored to further improve the performance of path planning algorithms, even though this study used the Genetic algorithm as a learning-based method. Future research may take into account techniques like evolutionary algorithms, reinforcement learning, and neural networks.

Dynamic path planning: Dynamic features in mobile robot environments, such as moving obstacles or changing conditions, are common. Future research could investigate how the comparative algorithms handle dynamic circumstances and build adaptive path planning systems capable of reacting and adapting to dynamic changes in real-time.

Multi-robot coordination: Path planning is a crucial component for a single mobile robot, but expanding the research to take coordination and cooperation amongst several robots into account would be a fascinating direction. Multi-robot systems could benefit from learning more about how various algorithms handle task allocation, coordination, and communication among a group of robots.

Optimization of performance: Future study might concentrate on improving the algorithms' performance in terms of computing effectiveness, memory utilization, and energy consumption. Real-time applications and resource-constrained systems would benefit from strategies to increase the algorithms' speed and resource usage.

Benchmarking and standardization: Creating benchmarks and uniform evaluation procedures would allow for consistent and fair comparisons between various research when evaluating path planning algorithms. To enable relevant and impartial comparisons, future studies may help build standardized evaluation metrics and benchmarks.

By focusing on these areas in future study, we can improve the effectiveness and dependability of autonomous robot navigation while also better understanding path planning algorithms for mobile robots.

BIBLIOGRAPHY

[1] Alessandro Gasparetto1, Paolo Boscariol1, Albano Lanzutti2, Renato Vidoni, "Path Planning and Trajectory Planning Algorithms: a General Overview," in *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*, vol. 29, Italy, Springer, Cham, 2015, p. 3–27.

[2] Choset, Howie and Lynch, Kevin M and Hutchinson, Seth and Kantor, George A and Burgard, Wolfram, Principles of robot motion: theory, algorithms, and implementations, London, England: MIT press, 2005.

[3] M. Alajlan, "Global Path Planning for Single and Multi-Robot Systems," Semantic Scholar, 2014.

[4] Mustafa S.Abeda, Omar F.Lutfyb,Qusay F. Al-Door, "A review on path planning algorithms for mobile robots," *Engineering and Technology Journal,* vol. 39, no. 5, pp. 804-820, 25 May 2021.

[5] C. Wang, "Comparative Research on Robot Path Planning Based on GA-ACA and ACA-GA," Canada, 2017.

[6] G Kalarani ,R Ranihemamalini, "A survey of the various path planning techniques used in the navigation of autonomous mobile robot," *Indian Journal of Applied Research,* vol. 4, no. 10, pp. 442--444, 2014.

[7] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, USA, IEEE, 1985, pp. 500-505.

[8] Syed Abdullah Fadzli,Sani Iyal Abdulkadir,Mokhairi Makhtar,Azrul Amri Jamal, "Robotic Indoor Path Planning Using Dijkstra's Algorithm with Multi-Layer Dictionaries," in *2nd International Conference on Information Science and Security (ICISS)*, Seoul, Korea (South), 2016.

[9] M.Bala Subramanian, Dr.K.Sudhagar,G.RajaRajeswari, "Intelligent Path Planning Of Mobile Robot Agent By Using Breadth First Search Algorithm," in *International Conference on Innovations in Engineering and Technology (ICIET'14)*, Chennai, India, 2014.

[10] D. KARABOGA, "D. Karaboga, “An idea based on honey bee swarm for numerical optimization,” Erciyes Univesity, Engineering Faculty, Computer Engineering Department, 2005.," Türkiye , 2005.

[11] Muna M. AL -Nayar ,Khulood E. Dagher ,Esraa A. Hadi, "A Comparative Study for Wheeled Mobile Robot Path Planning Based on Modified Intelligent Algorithms," *The Iraqi Journal For Mechanical And Material Engineering,* vol. 19, pp. 60-74, March 2019.

[12] ZHANG, Zhengwan and ZHANG, Chunjiong and LI, Hongbing and XIE, Tao, "Multipath transmission selection algorithm based on immune connectivity model," *Journal of Computer Applications,* vol. 40, no. 12, p. 3571, 2020.

[13] Li, Xue and Wang, Lei and Wang, L, "Application of improved ant colony optimization in mobile robot trajectory planning," *Mathematical Biosciences and Engineering,* vol. 17, no. 6, pp. 6756--6774, 2020.

[14] Hao, Kun and Zhao, Jiale and Wang, Beibei and Liu, Yonglei and Wang, Chuanqi, "The application of an adaptive genetic algorithm based on collision detection in path planning of mobile robots," *Computational Intelligence and Neuroscience,* vol. 2021, pp. 1-20, 2021.

[15] Song, Qisong and Li, Shaobo and Yang, Jing and Bai, Qiang and Hu, Jianjun and Zhang, Xingxing and Zhang, Ansi and others, "Intelligent Optimization Algorithm-Based Path Planning for a Mobile Robot," *Computational Intelligence and Neuroscience,* vol. 2021, 2021.

[16] Oscar Castillo ,Héctor Neyoy , José Soria , Mario García ,Fevrier Valdez, "Dynamic Fuzzy Logic Parameter Tuning for ACO and Its Application in the Fuzzy Logic Control of an Autonomous Mobile Robot," *International Journal of Advanced Robotic Systems,* vol. 10, no. 1, p. 51, 2013.

[17] AbuBaker, Ayman, "A novel mobile robot navigation system using neuro-fuzzy rule-based optimization technique,," *Research Journal of Applied Sciences, Engineering and Technology,* vol. 4, no. 15, pp. 2577-2583, 2012.

[18] Gopikrishnan Sasi Kumar,Lingam Ravikumar,Harshit Gole, "PATH PLANNING ALGORITHMS: A COMPARATIVE STUDY," in *National Conference on Space Transportation Systems*, India, 2011.

[19] Ahmed Mohamed Mohsen ,Maha Ahmed Sharkas,Mohamed Saad Zaghlol, "New Real Time (M-Bug) Algorithm for Path Planning and Obstacle Avoidance In 2D Unknown Environment," in *ICCTA*, Egybt, 2019.

[20] Qi, Jie and Yang, Hui and Sun, Haixin, "MOD-RRT*: A Sampling-Based Algorithm for Robot Path Planning in Dynamic Environment," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 8, pp. 7244-7251, 8 August 2021.

[21] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning:An Introduction, London, England: The MIT Press, 2015.

[22] M. A. Naeem, "A Python Module for Maze Search Algorithms," 15 Sept 2021. [Online]. Available: https://towardsdatascience.com/.

[23] Xuewu Wang1 · Jianbin Wei1 · Xin Zhou1 · Zelong Xia1 · Xingsheng Gu1, "AEB-RRT*: an adaptive extension bidirectional RRT* algorithm," *Springer*, p. 685–704, 2022.

[24] E. W. Dijksta, "A note on two problems in connexion with graphs," *{Numerische mathematik*, vol. 1, no. 1, pp. 269-271, 1959.

[25] R. M. Kangutkar, "Obstacle Avoidance and Path Planning for Smart Indoor Agents," Rochester Institute of Technology, 2017.

[26] M. W. Otte, "A Survey of Machine Learning Approaches to Robotic Path-Planning," Boulder, 2015.

[27] A. Singhal, "A* Algorithm Example in AI," 2020. [Online]. Available: www.gatevidyalay.com.

[28] Thomas H. Cormen,Charles E. Leiserson,Ronald L. Rivest,Clifford Stein, Introduction to Algorithms, USA: The MIT Press, 2009.

[29] Dasgupta, Dipankar and Michalewicz, Zbigniew, Evolutionary algorithms in engineering applications, Springer Science \& Business Media, 2013.

Abstract

This thesis presents a comprehensive comparative study on path planning algorithms for mobile robots, focusing on the simulation environment of the Maze Problem. Path planning is a crucial aspect of mobile robot navigation, and selecting the most appropriate algorithm is essential for efficient and robust operation. This research evaluates and compares the performance of several basic search algorithms, including Dijkstra's algorithm, A* algorithm, BFS algorithm, DFS algorithm, and the artificial intelligence method Genetic algorithm. The evaluation is based on multiple evaluation metrics, namely path length, running time, robustness, search path, and time complexity. To ensure a comprehensive analysis, each algorithm is tested in six different scenarios, with each approach executed ten times in each environment. The experimental results provide valuable insights into the strengths and limitations of the algorithms, aiding researchers and practitioners in selecting the most suitable path planning algorithm for mobile robot applications.

***Keywords*:** path planning, mobile robots, maze solving, Dijkstra's algorithm, A*, BFS, DFS, Genetic algorithm.

الملخص

تقدم هذه الأطروحة دراسة مقارنة شاملة عن خوارزميات تخطيط المسار للروبوتات المتنقلة، مع التركيز على بيئة محاكاة مشكلة المتاهة. فتخطيط المسار جانب حاسم من جوانب الملاحة الآلية المتنقلة، واختيار أنسب خوارزمية أمر ضروري للتشغيل الكفء والمتين. هذا البحث يقيّم ويقارن أداء العديد من خوارزميات البحث الأساسية، بما في ذلك خوارزمية Dijkstra ، خوارزمية A* ، خوارزميةDFSوBFS ، وطريقة الذكاء الاصطناعية خوارزمية جينية GA. ويستند التقييم إلى مقاييس تقييم متعددة، وهي طول المسار، ووقت التشغيل، والقوة، ومسار البحث، والتعقيد الزمني. ولضمان إجراء تحليل شامل، يتم اختبار كل خوارزمية في ستة سيناريوهات مختلفة، مع تنفيذ كل نهج عشر مرات في كل بيئة. وتوفر النتائج التجريبية معلومات قيّمة عن مواطن قوة الخوارزميات وحدودها، وتساعد الباحثين والممارسين في اختيار أنسب خوارزميات تخطيط المسار لتطبيقات الروبوتات المتنقلة.

الكلمات المفتاحية: تخطيط المسار، الروبوتات المتنقلة، حل المتاهة، خوارزمية ديجكسترا، خوارزميةA star ، خوارزميةDFS، BFS ، خوارزمية جينية.

Résumé

Cette thèse présente une étude comparative complète sur les algorithmes de planification des chemins pour les robots mobiles, en se concentrant sur l'environnement de simulation du problème du labyrinthe. La planification du parcours est un aspect crucial de la navigation robot mobile, et la sélection de l'algorithme le plus approprié est essentielle pour un fonctionnement efficace et robuste. Cette recherche évalue et compare les performances de plusieurs algorithmes de recherche de base, y compris l'algorithme de Dijkstra, l'algorithme A*, l'algorithme BFS,DFS et la méthode d'intelligence artificielle algoritme génétique. L'évaluation est basée sur plusieurs indicateurs d'évaluer, à savoir la longueur du chemin, le temps d'exécution, la robustesse, le chemin de recherche et la complexité du temps. Pour assurer une analyse complète, chaque algorithme est testé dans six scénarios différents, chaque approche étant exécutée dix fois dans chaque environnement. Les résultats expérimentaux fournissent des informations précieuses sur les points forts et les limites des algorithmes, aidant les chercheurs et les praticiens à choisir l'algorithme de planification du parcours le plus approprié pour les applications de robots mobiles.

Mots clés: : planification des chemins, robots mobiles, résolution de labyrinthe, algorithme de Dijkstra, A*, BFS, DFS, Algoritme génétique.