

**INSTITUTO TECNOLÓGICO DE AERONÁUTICA**



**Arthur José de Sousa Rodrigues**

**MULTI-AGENT GRAPH EXPLORATION WITHOUT  
COMMUNICATION**

Bachelor's Thesis  
2023

**Computer Engineering**

**Arthur José de Sousa Rodrigues**

**MULTI-AGENT GRAPH EXPLORATION WITHOUT  
COMMUNICATION**

Advisor

Prof. Dr. Luiz Gustavo Bizarro Mirisola (ITA)

Co-advisor

Prof. Dr. Vitor Venceslau Curtis (ITA)

**COMPUTER ENGINEERING**

SÃO JOSÉ DOS CAMPOS  
INSTITUTO TECNOLÓGICO DE AERONÁUTICA

2023

**Cataloging-in Publication Data**  
**Documentation and Information Division**

de Sousa Rodrigues, Arthur José  
Multi-agent graph exploration without communication / Arthur José de Sousa Rodrigues.  
São José dos Campos, 2023.  
49f.

Bachelor's Thesis – Course of Computer Engineering– Instituto Tecnológico de Aeronáutica,  
2023. Advisor: Prof. Dr. Luiz Gustavo Bizarro Mirisola. Co-advisor: Prof. Dr. Vitor Venceslau  
Curtis.

1. Maze. 2. Graph. 3. Search. 4. Multi-agent. I. Instituto Tecnológico de Aeronáutica. II. Title.

**BIBLIOGRAPHIC REFERENCE**

DE SOUSA RODRIGUES, Arthur José. **Multi-agent graph exploration without communication**. 2023. 49f. Bachelor's Thesis – Instituto Tecnológico de Aeronáutica, São José dos Campos.

**CESSION OF RIGHTS**

AUTHOR'S NAME: Arthur José de Sousa Rodrigues

PUBLICATION TITLE: Multi-agent graph exploration without communication.

PUBLICATION KIND/YEAR: Bachelor's Thesis / 2023

It is granted to Instituto Tecnológico de Aeronáutica permission to reproduce copies of this work and to only loan or to sell copies for academic and scientific purposes. The author reserves other publication rights and no part of this work can be reproduced without the authorization of the author.

---

Arthur José de Sousa Rodrigues  
Rua H8B 226  
12.228-461 – São José dos Campos–SP

# MULTI-AGENT GRAPH EXPLORATION WITHOUT COMMUNICATION

Bachelor's Thesis approved in its non-final version by the signatories below:

---

Arthur José de Sousa Rodrigues  
Author

---

Luiz Gustavo Bizarro Mirisola (ITA)  
Advisor

---

Vitor Venceslau Curtis (ITA)  
Co-advisor

---

Prof. Dr. Marcos Ricardo Omena de Albuquerque Máximo  
Course Coordinator of Computer Engineering

São José dos Campos: Jun 16, 2023.

To my mother Ana Paloma, daughter  
of Antônio and Vandnei. To my father  
Jairo José, son of José and Rosinete.

# Acknowledgments

I thank my family for investing time and financial resources in my studies. And to give me unconditionally love and motivation.

I thank my H8 friends for always supporting me with knowledge and positive words.

I thank my advisor Prof. Dr. Luiz Gustavo Bizarro Mirisola and my co-advisor Prof. Dr. Vitor Venceslau Curtis for all patience and attention that has been given to me.

I thank Brazilian Air Force for supporting my studies since I joined the Air Cadet Preparatory School (EPCAr).

*“Now to him who is able to do immeasurably more than all we ask or imagine, according to his power that is at work within us, to him be glory in the church and in Christ Jesus throughout all generations, for ever and ever”*

— SAINT PAUL

# Resumo

Algoritmos computacionais de busca têm sido estudados por cientistas e companhias de engenharia desde o último século devido às aplicações de tais algoritmos em situações reais, como agendamento de linhas aéreas, planejamento de rota em mapas, algoritmos de busca na Internet, planejamento de rotas em redes de computadores, robótica, etc. Usualmente, esses métodos se baseiam em tipos abstratos de dados, como grafos e árvores, para transferir um problema real para dentro de um contexto delimitado e inteligível. No campo da Ciência da Computação, grafos são tipos abstratos de dados que podem servir como suporte ferramental para algoritmos de busca.

De uma abordagem relacionada a algoritmos de busca em labirintos, este trabalho de graduação propõe um método para explorar grafos em um âmbito multiagente, em que os agentes não são capazes de se comunicarem entre eles, o que faz referência a problemas da realidade, como exploração no fundo do mar, busca em estruturas compostas por muros maciços, busca em locais inóspitos por agentes com restrição energética, etc.



# Abstract

Computational search algorithms have been studied by many scholars and engineering companies since the last century due to the real-life applications of such algorithms, such as in airline scheduling, planning path on maps, search engine algorithms, social media marketing, Internet routing protocols, robotics, etc. Usually these methods rely on abstract data types, including graphs and trees, to transfer a real-world problem into a delimited and intelligible environment. In the context of computer science, graphs are abstract data type that might support computational search algorithms.

From the maze-solving algorithms approach, this work aims to propose a method to explore a graph in a multi-agent situation, where the agents cannot communicate with each other, that makes reference to real-life problems, like deep sea exploration, search in large wall structures, search with low energy-based agents in inhospitable environments.

# List of Figures

FIGURE 2.1 – Blue and red agents traversing a $6 \times 6$ maze. The goal is represented by the green entity. This maze is a perfect maze (NAEEM, 2021). . .	25
FIGURE 2.2 – Maze cell indices. Source: Naeem (2021). . . . .	26
FIGURE 2.3 – Red agent traversing a maze toward the green goal. This maze is a perfect maze (NAEEM, 2021). . . . .	26
FIGURE 2.4 – Graph representation of the maze presented in Figure 2.3. The cell indices are pointed out. The agent starts at index (3,3), and achieves the goal at index (1,1). . . . .	26
FIGURE 2.5 – Three agents (Red, Blue, and Yellow) disperse from each other in a fictional and perfect maze. . . . .	29
FIGURE 2.6 – A fictional maze represented as a tree where each node has an established interval according to an agent’s traverse guided by Pseudocode 1. The green node represents the goal. . . . .	30
FIGURE 2.7 – A red agent with an interval $[0, \frac{1}{2}[$ prioritizing to fill completely its interval in a fictional maze. . . . .	30
FIGURE 2.8 – An agent with interval $[\frac{1}{3}, \frac{2}{3}[$ finishes its interval in the visit order indicated. Regardless of the behavior that it assumes after finishing its interval, it never finds the green goal since it doesn’t go through already visited nodes, except in backtracking situations. It is a fictional maze. . . . .	33

FIGURE 2.9 – Both trees have the same format, but the first one has its node convergence intervals presented in a fractional representation, and the second one has its node convergence intervals presented in a mixed radix representation. As the agent’s path carries information about the nodes’ convergence intervals when the path is structured in a mixed radix representation, it is possible to save, with only one numerical representation, information about the tree explored by the agent and the convergence intervals of the tree’s nodes. It is a fictional maze. . . . .	36
FIGURE 3.1 – Results of average of steps, pioneer’s average of steps, and standard deviation of the average of steps. 1 single agent until 40 cooperative agents were run in 250 different mazes for each type of size. . . . .	40
FIGURE 3.2 – Results of fraction of explored maze and fraction of maze explored when pioneer reaches the target. 1 single agent until 40 cooperative agents were run in 250 different mazes for each type of size. . . . .	41
FIGURE 3.3 – Example of an agent with interval $[0, \frac{1}{3}[$ under the incremental policy, where it changes its interval to $[\frac{1}{3}, \frac{2}{3}[$ when it finishes its starting interval, in the 5th visited cell, and it changes the order of action to right-left. It is a fictional maze. . . . .	42
FIGURE 3.4 – Comparison between the results of average of steps, pioneer’s average of steps, and standard deviation of the average of steps relating to our “1 interval” algorithm and our “2 intervals” algorithm. 1 single agent until 40 cooperative agents were run in 250 different mazes for each type of size, and for each algorithm. . . . .	43
FIGURE 3.5 – Comparison between the results of fraction of explored maze and fraction of maze explored when pioneer reaches the target relating to our “1 interval” algorithm and our “2 intervals” algorithm. 1 single agent until 40 cooperative agents were run in 250 different mazes for each type of size, and for each algorithm. . . . .	44
FIGURE 3.6 – Comparison between the results of pioneer’s average of steps relating to our algorithms and the extended Tarry’s algorithm. 1 single agent until 40 cooperative agents were run in 250 different mazes for each type of size, and for each algorithm. . . . .	46

---

FIGURE 3.7 – Comparison between the results of fraction of maze explored when pioneer reaches the target relating to our algorithms and the ex- tended Tarry’s algorithm. 1 single agent until 40 cooperative agents were run in 250 different mazes for each type of size, and for each algorithm. . . . .	47
---	----

# List of Tables

# List of Algorithms

1	Traverse of the agent through the maze (interpreted as a tree by the agent).	31
1	Traverse of the agent through the maze (interpreted as a tree by the agent).	32
2	Methods related to mixed radix numerical representation. . . . .	35

# List of Abbreviations and Acronyms

DFS Depth First Search

BFS Breath First Search

# List of Symbols



# Contents

1	INTRODUCTION . . . . .	18
1.1	Motivation . . . . .	18
1.2	Related work . . . . .	19
1.3	Definitions . . . . .	20
1.3.1	Graph . . . . .	20
1.3.2	Maze . . . . .	21
1.3.3	Agent . . . . .	21
1.3.4	Maze-solving algorithms . . . . .	22
1.3.5	Mixed Radix . . . . .	22
2	MODELS . . . . .	24
2.1	Maze and multi-agent exploration . . . . .	24
2.2	Maze from a graph topology perspective . . . . .	25
2.3	Multi-agent exploration without communication . . . . .	27
2.3.1	Our algorithm's key concepts . . . . .	27
2.3.2	Pseudocode for our algorithm . . . . .	29
2.4	Mixed radix representation to the agent path . . . . .	33
2.5	Extended Tarry's algorithm . . . . .	34
3	RESULTS AND DISCUSSION . . . . .	38
3.1	Our algorithm's performance . . . . .	38
3.2	Incremental policy for the agents . . . . .	39
3.3	Our algorithm vs. Tarry's algorithm . . . . .	42

---

4	CONCLUSIONS AND FUTURE WORKS . . . . .	48
	BIBLIOGRAPHY . . . . .	49

# 1 Introduction

This bachelor thesis aims to discuss the main maze exploration algorithms and then propose a method where multi-agents must find the maze solution without any type of communication, only working with coordinated and deterministic distributions to guide their behaviors. The main challenge of this research is to find a distributed exploration approach with total communication restriction since an agent’s partial knowledge cannot be shared with another agent and, at the same time, a single agent must avoid repeating a branch of another agent. The proposed maze structure abstraction is a traditional regular grid that can be generalized into graphs.

This report is the initial part of the bachelor’s thesis, and this chapter intends to introduce the general concept of the related thesis and present the motivation (Section 1.1), the related work (Section 1.2), and the main definitions about maze-solving algorithms (Section 1.3).

## 1.1 Motivation

Graph exploration has been the target of studies since Leonhard Euler proved that Seven Bridges of Königsberg (SHIELDS, 2012) has no solution. It has been researched not only in academia but also in the industry due to several practical applications, like airline scheduling, planning path on maps, search engine algorithms, social media marketing, Internet routing protocols, and robotics.

Specifically in robotics, graph exploration can be used to explore a maze with a single agent through a bunch of traditional algorithms: random mouse, wall follower, Trémaux, etc (SADIK et al., 2010). And it can be useful to guide many real-life problems such as search in nuclear plant disasters, burning buildings, and extraterrestrial environments. In these previous examples, the multi-agent exploration potentially can speed up the exploration, despite of it will be ineffective if different agents explore the same portions of the graph. Recent studies have explored multi-agent maze-solving algorithms as seen in the Multi-Agent Maze Exploration paper (KIVELEVITCH; COHEN, 2010), where authors proposed a Tarry’s algorithm generalization. It is important to emphasize that maze-

solving algorithms consider that the structure is unknown.

Traditional multi-agent maze exploration approaches is based on internal communication between agents, where each agent knows about visited cells by another agent. It avoids a second exploration in a useless path and thus it decreases computational costs. However, there are some real situations where communication is limited or impossible, such as deep sea exploration, search in large wall structures, or search with low energy-based autonomous agents.

However, the zero-communication approach between agents have not been concretely found in the literature despite it may have real applications and may guide search plans in real-world problems. In order to explore it, this work presents some ways to achieve the solution of a maze based on agents without communication, that might be generalized to graph exploration algorithms.

## 1.2 Related work

Multi-agent cooperative system approaches are common in literature, especially when it comes from robotics researches. In the context of communicable agents, mainly in robotics, this chapter presents some related works.

Mataric (1995) established common properties across different scenarios of mobile multi-agent interactions, such as dispersion - “the ability of a group of agents to spread out in order to establish and maintain some minimum interagent distance” -, aggregation - “the ability of a group of agents to gather in order to establish and maintain some maximum interagent distance” -, homing - “the ability of an agent to find a particular region or location” -, etc. The author proposes a synthetic structure in order to abstract different types of interagent basis behaviors.

Burgard et al. (2005) pointed out that an exploration group of robots takes several advantages over single agent exploration, despite a coordinating group might introduce redundancy. The authors present an algorithm to efficiently explore an environment by mobile and autonomous robots within a centralized communication range. These coordinated robots can completely cover the environment in a significantly reduced time compared to other related approaches cited in the article. However, this bachelor thesis intends to study a different technique based on zero-communication range.

Sadik et al. (2010) presents, as seen in the title of the paper, a comprehensive and comparative study of maze-solving algorithms techniques by implementing graph theory. The research is delimited in the “Micromouse competition” context, which is a famous maze competition that has been performed worldwide since late 1970s. The authors

compared maze-solving methods based on graph theory algorithms, such as DFS (Depth First Search) and BFS (Breath First Search) flood-fill, to common algorithms in the “Micromouse” context, such as Wall Follower. They concluded that, despite graph theory algorithms demands higher computational complexity, they are more proficient compared to related algorithms that doesn’t use graph representation, and only make decisions relied on the neighborhood.

Based around maze-solving algorithms, Kivelevitch & Cohen (2010) proposed a generalization of Tarry’s algorithm, but the new approach is that all visited cells of the maze are known by each agent, since each agent shares its knowledge with all the others. In that sense, each one holds an dynamic map of the maze excluding redundant information, allowing information sharing. The authors presents in the article the performance of the proposed solution, where a group of virtual coordinate agents is required to find the goal without an *a priori* knowledge of the maze, so-called “maze exploration”.

Beisel (2014), similarly to aforementioned, worked in simulation and mathematical analysis for strategies related to cooperative autonomous robots, that can share messages with each other to exit a maze. The author concluded that a cooperative approach might result in significant performance improvements compared to uncooperative and uncoordinated robots.

## 1.3 Definitions

Mainly considering these previous references, this chapter intends to define common terms about maze-solving algorithms, graph theory, and mathematical tools that were useful to abstract the proposed solution. Furthermore, it intends to clarify the approach domain of this work.

### 1.3.1 Graph

Graph theory has been used in various real-life applications, such as in biology, social sciences, engineering, computer science, etc. Manber (1989) establishes that a graph  $G = (V, E)$  consists of a set  $V$  of vertices (also called nodes), and a set  $E$  of edges. Each edge corresponds to a pair of vertices, and represents relationships among the vertices. A graph can be directed or undirected. The edges in a directed graph are ordered pairs, i.e., the order of the edge connection between two vertices is important. On the other hand, the edges in an undirected graph are unordered pairs, i.e., the order of the edge connection between two vertices is not important. The author gives an example: a graph may represent a set of people, and the edges may connect any two persons who know each

other. Moreover, he discuss several computational problems in terms of graphs, where he describes famous graph exploration algorithms, such as DFS (Depth First Search) and BFS (Breath First Search). Please check Manber (1989) for more details.

Manber (1989) works with some representations of graphs in his book. One of representations is the adjacency list of a graph, that is described by an array of lists. In the adjacency list representation, each vertex is associated with a linked list consisting of all the edges adjacent to this vertex. Supposing that  $|V| = n$ , the adjacency list will have  $n$  linked lists. In the case of this work, graphs are always undirected, therefore, if the linked list  $i$  of a vertex  $v_i$  has a vertex  $v_j$ , the linked list  $j$  will have the vertex  $v_i$ .

It is worth to mention that there is a subdomain of graph data type called tree. A tree is also an abstract data type, which can be represented topologically as a hierarchical structure of nodes. Manber (1989) indicates that, in a tree, it is possible to hierarchize all the edges from the root (the head node in terms of tree hierarchy), and hence trees are sometimes called rooted trees.

From Section 2.2, a perspective of a maze from a tree and a graph topologies will be explored.

### 1.3.2 Maze

This report define a maze at the same perspective presented in Kivelevitch & Cohen (2010). A maze is a  $n$ -dimensional gridded space of any size, usually rectangular. The gridded space is composed by a set of cells, while a cell is the elementary item of a maze, defined as a delimited  $n$ -dimensional space. Cells might be connected or not connected to another adjacent cell, separated by a “wall” in the latter case. Without losing generality, as will be presented in Section 2.1, this work considered a maze, for simulation purposes, as a two-dimensional gridded space composed by two-dimensional bounded cells.

Thus, from the above definition, a 4-neighbor 2D grid graph, where a wall is represented by eliminating the edge between 2 neighboring cells, is a good computational representation of a maze, which will be a tree if the maze has no loops, or equivalently, if there is only one path between any two cells, including the start and/or goal positions.

### 1.3.3 Agent

As defined in Kivelevitch & Cohen (2010), an agent is an autonomous entity that can traverse the maze obeying the connection of the cells.

In a maze context, a multi-agent approach describes the coordinating behavior of several autonomous agents.

### 1.3.4 Maze-solving algorithms

A maze has a goal to be achieved, which is usually a single marked cell or a path for an agent to exit the maze. Since the last century, many scholars have studied algorithms to solve the maze computationally, and the proposed solutions are so-called “maze-solving algorithms”.

### 1.3.5 Mixed Radix

Mixed radix is a numerical representation which generalizes standard positional numerical systems to allow a different base for each digit. In a traditional numerical system, there is not a base variation through the numbers, nevertheless, in a mixed radix representation, a number is represented by a sequence of numerals in different bases, where each numeral is a multiple of the previous numerical sequence, however relied on a different base.

For example, Arndt (2011) establishes a arithmetical method to manipulate a specific subset of mixed radix numbers. The mixed radix representation  $A = [a_0, a_1, a_2, \dots, a_{n-1}]$  of a number  $x$  with respect to a radix vector  $M = [m_0, m_1, m_2, \dots, m_{n-1}]$  is given by:

$$x = \sum_{k=0}^{n-1} a_k \prod_{j=0}^{k-1} m_j \quad (1.1)$$

where  $0 \leq a_j < m_j$  (and  $0 \leq x < \prod_{j=0}^{n-1} m_j$ , so that  $n$  digits suffice). In a traditional positional numerical system, the vector  $M$  is like  $M = [r, r, r, \dots, r]$ , and then the relation is simply given by:

$$x = \sum_{k=0}^{n-1} a_k r^k \quad (1.2)$$

For instance, if  $x$  is represented by  $1_2 3_9 5_6$ ,  $x$  might be transformed to decimal base by the following steps:

$$\begin{aligned} x &= 1_2 3_9 5_6 \\ A &= [5, 3, 1] \\ M &= [6, 9, 2] \\ n &= 3 \\ x &= \sum_{k=0}^2 a_k \prod_{j=0}^{k-1} m_j = 77 \end{aligned} \quad (1.3)$$

---

Despite the aforementioned perspective, this work presents a slightly different approach in Section 2.4, since it uses a mixed radix representation for numbers contained in the real range  $[0, 1]$ .



## 2 Models

### 2.1 Maze and multi-agent exploration

For simulation purposes, this work used an open-source tool that models a maze under a 4-neighbor 2D grid graph perspective, where each cell is a square with its edges composed by a wall or not. If there is a wall, an agent cannot traverse the maze through the corresponding edge. On the other hand, if there is not a wall, an agent has a free way to traverse the maze through the related edge. It is worth to mention that, if an edge has a wall, the edge of the adjacent corresponding cell also has necessarily a wall. Likewise, if an edge has not a wall, the edge of the adjacent corresponding cell doesn't have a wall.

The maze has a goal that is a single marked cell, and an agent inside the maze aims to find the marked cell, traversing the maze cell by cell. This agent is an autonomous entity that follows a specific algorithm and previously programmed rules. So that it doesn't go through the same path more than one time, it stores the visited cells. Thus, when there are various possible branches to extend the agent's path, it ignores cells already visited by itself. If there is no candidate to be a possible branch, the agent goes back to the previous visited cell. Furthermore, if more than one agent traverses the maze in order to find the goal, differently from some approaches presented in Beisel (2014), Burgard et al. (2005), and Kivelevitch & Cohen (2010), there is no intercommunication between agents in the approach of this research, i.e., an agent knows nothing about another agent's search.

Naeem (2021) developed an open-source maze generator. It is a python module that creates randomly mazes and enables the user to simulate its own maze-solving algorithm. In this context, this work presents a multi-agent maze-solving algorithm that has been simulated over Naeem (2021) open-source code, with modifications. Figure 2.1 presents, for example, 2 agents traversing a  $6 \times 6$  maze toward the goal.

Naeem (2021) software creates by default a "perfect maze", which means that there is one and only one path to the goal from any cell. However, it is possible to set the code to generate a imperfect maze.

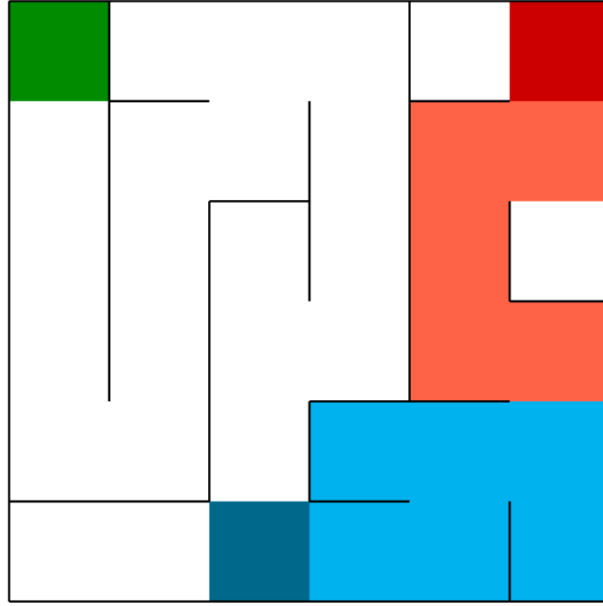


FIGURE 2.1 – Blue and red agents traversing a  $6 \times 6$  maze. The goal is represented by the green entity. This maze is a perfect maze (NAEEM, 2021).

## 2.2 Maze from a graph topology perspective

As pointed out in Section 2.1, given that an agent ignores visited cells, there are important statements related to this work:

- if there is only one path to the goal from any cell, it is valid to consider a maze as a tree, i.e., a perfect maze (NAEEM, 2021);
- if there is more than one path to the goal from any cell, the maze cannot be considered as a tree, but it might be considered as a general 2D grid;
- regarding the last statement, even though not every maze can be considered as a tree, an agent path might be individually interpreted as a tree if such agent does not go through already visited cells, except in backtracking situations.

Figure 2.4 gives an example of the graph representation of the maze presented in Figure 2.3, where a red agent is traversing the maze toward the green goal. As established by Naeem (2021), the maze cells are programmatically addressed as indices of a matrix, such as represented in Figure 2.2.

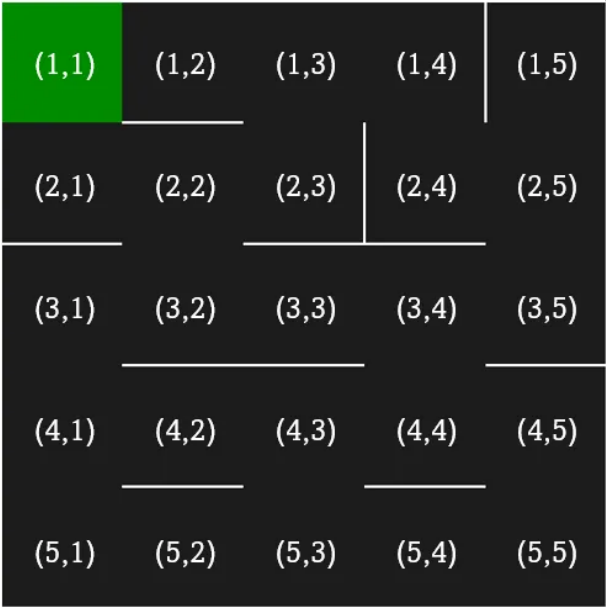


FIGURE 2.2 – Maze cell indices. Source: Naeem (2021).

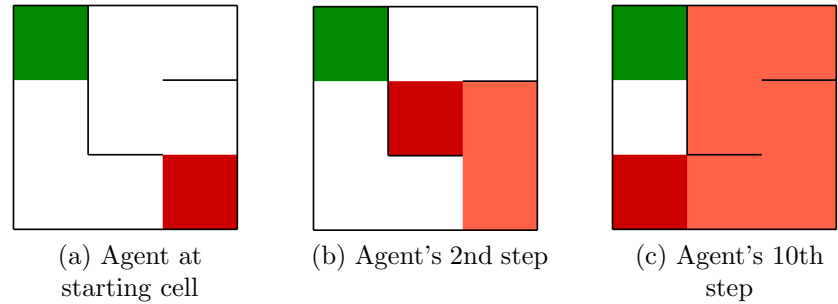


FIGURE 2.3 – Red agent traversing a maze toward the green goal. This maze is a perfect maze (NAEEM, 2021).

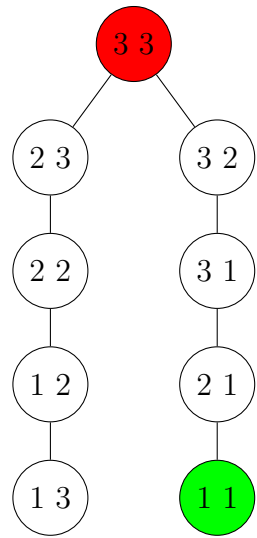


FIGURE 2.4 – Graph representation of the maze presented in Figure 2.3. The cell indices are pointed out. The agent starts at index (3,3), and achieves the goal at index (1,1).

## 2.3 Multi-agent exploration without communication

### 2.3.1 Our algorithm's key concepts

The goal of this work is to present a maze-solving algorithm in a multi-agent environment, where an agent cannot communicate with the other agents. Thus, each agent must be previously programmed to avoid exploring the same portion of the maze as other agents. Indeed, they should be as dispersed as possible.

First of all, this research considers a maze as a graph, where each node is a cell representing the maze. Given an initial node from which the agent starts its path, the agent checks if the node has children. Since the graph is undirected, i.e., there are no completely walled cells, this work establishes some statements:

- if an agent finds the cell where the goal is, the agent finishes its path;
- if an agent is not in the cell where the goal is and the cell has only one unwalled edge, the agent necessarily goes through such edge;
- if an agent is not in the cell where the goal is and the cell has more than one unwalled edge, the agent needs to decide which edge it will go through.

To establish a decision algorithm to the last statement, this work proposes firstly an interval division related to the agents and the nodes around them. Supposing that there are  $k$  agents  $a_1, a_2, \dots, a_k$  to explore the maze, each agent will have a corresponding and proportional range of action related to the graph, as presented below:

$$\begin{aligned}
 d &= 1/k \\
 a_1 &: [0, d[ \\
 a_2 &: [d, 2d[ \\
 a_3 &: [3d, 4d[ \\
 &\dots \\
 a_k &: [(k-1)d, 1]
 \end{aligned} \tag{2.1}$$

where  $d$  is the size of the partition corresponding to each agent. On the other hand, this work has handled convergence intervals to each node, where the agents tends to match its interval with the node's convergence interval. In this way, an agent intends to traverse the maze as dispersed from the others as possible.

To disperse the agents through the maze, this work establishes some steps in order to guide the agent, that knows nothing about another agent's search. As a starting

point for graph exploration, our algorithm works specifically in the context of perfect mazes (NAEEM, 2021). The following steps base the reasoning expressed in the proposed pseudocode in Section 2.3.2:

- every agent  $a_i$  has a previously programmed interval;
- every agent starts its search at the same cell, i.e., the root of an agent's tree is the same for any agent. The root's converge interval is  $[0, 1]$ ;
- every agent explores the maze calculating dynamically the converge intervals of the children of the current node where the agent is. These convergence intervals are calculated based on the following steps:
  - if there is only one child, the child's convergence interval is the same as the current node's convergence interval;
  - if there is more than one child, the current node's convergence interval is uniformly split between its children;
- every agent must go to the first child that its convergence interval intersects the agent's interval. The order of children must be well defined so that every agent gets the same calculation result of some node's convergence interval. Then, in the context of perfect mazes, it ensures that every cell (node) has the same convergence interval regardless of the agent. This research established a clockwise order for the cells - North, East, South, West - from left to right regarding the insertion of children;
- an agent doesn't visit an already visited node, except in backtracking situations where there are no children to visit, i.e., the node in fact has no children, the children have already been visited, or there are no more children that match its convergence intervals to the agent's interval;
- every agent must fill its interval going through the correspondent nodes until finds the goal. If it doesn't find the goal, but it fills its interval, surely the convergence interval of the goal node is outside of the agent's interval. In this context, the agent stops the search or changes its decision algorithm. In the case that it changes its behavior, this work proposes two optional secondary conducts:
  - the agent modifies its interval and the exploration order from left-right to right-left. Section 3 discusses broader about this point;
  - the agent explores the maze using some DFS algorithm, but ignores already visited nodes.

The following pictures expose key concepts of our algorithms:

- Figure 2.5 is related to the dispersion concept, in which every agent explores the maze as dispersed as possible from another agent;
- Figure 2.6 presents the convergence intervals of the cells (nodes) of a fictional maze when a given agent explores it based on the aforementioned steps;
- Figure 2.7 exposes an agent with an interval  $[0, \frac{1}{2}[$  prioritizing to fill completely its interval, despite there being a bigger subtree on the right of the root. In fact, our algorithm is an exploration algorithm, so an agent has no information about the structure of the maze. It is worth to emphasizing that when an agent finishes its interval, it needs to change its behavior or simply stop the search;
- Figure 2.8 exposes the visit order of an agent with interval  $[\frac{1}{3}, \frac{2}{3}[$ . Due to the fact that the agent finishes its interval in the 7th visited node, and it doesn't go through already visited nodes, except in backtracking situations, it never finds the green goal, regardless of the behavior that it assumes after finishing its interval. Therefore, an agent might be unsuccessful in the search because it doesn't repeat a crossing - from a node to another node in the same direction - more than one time.

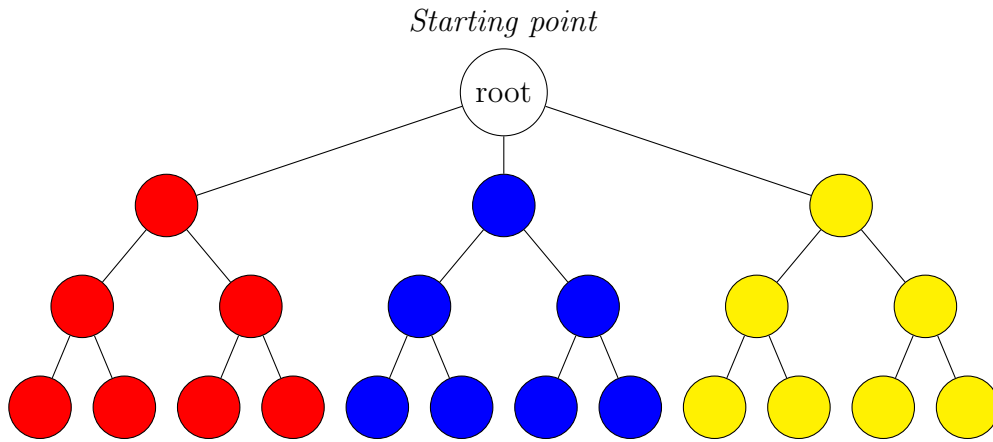


FIGURE 2.5 – Three agents (Red, Blue, and Yellow) disperse from each other in a fictional and perfect maze.

### 2.3.2 Pseudocode for our algorithm

As pointed out in Section 1.3.1, a graph may be represented as an adjacency list. Given that an agent path through an unknown maze may be represented as a tree in the situation approached in Section 2.2, where an agent does not go through already visited cells, except in backtracking situations, and considering the agent path as an adjacency list  $A$ , where the first explored node (root)  $v_1$  has a convergence interval equal to  $[0, 1]$ , while the other nodes  $v_i$  of the maze are unknown, i.e., the agent doesn't know previously

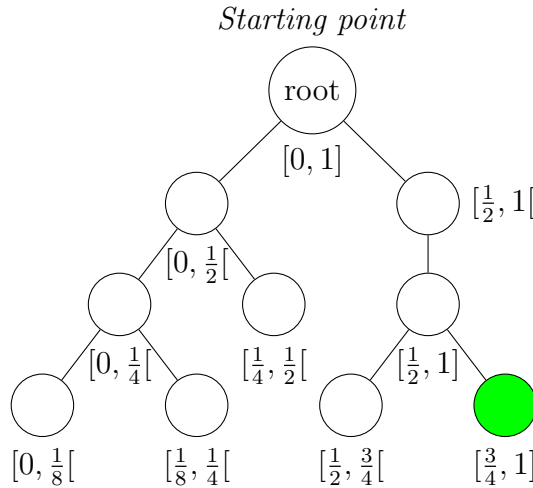


FIGURE 2.6 – A fictional maze represented as a tree where each node has an established interval according to an agent’s traverse guided by Pseudocode 1. The green node represents the goal.

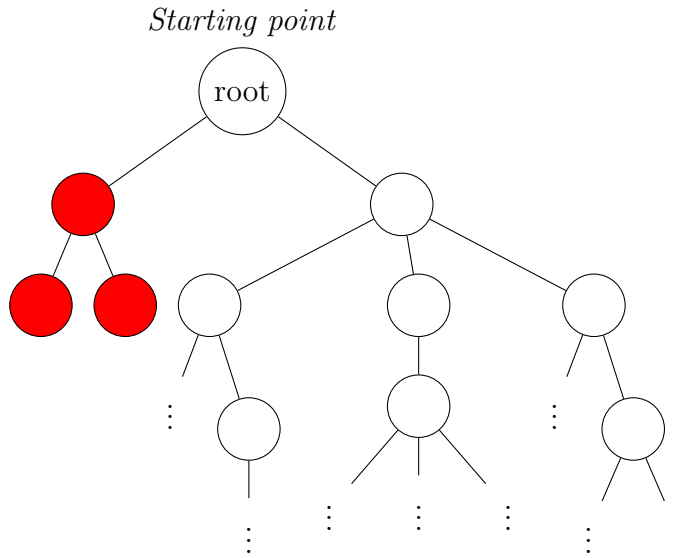


FIGURE 2.7 – A red agent with an interval  $[0, \frac{1}{2}[$  prioritizing to fill completely its interval in a fictional maze.

their convergence intervals, Pseudocode 1 establishes the convergence intervals of each node specifically in the context of perfect mazes (NAEEM, 2021), despite being a starting point to solve similar problems in the context of imperfect mazes, i.e., the graph of the maze cannot be considered as tree in a macro view.

As mentioned previously, there is no communication between agents. Therefore, Pseudocode 1 considers that each agent needs to explore the maze individually and calculate by itself the convergence intervals of each node. Furthermore, it is important to emphasize that each agent considers individually the maze as tree, where the root is the agent’s starting node.

---

**Algorithm 1** Traverse of the agent through the maze (interpreted as a tree by the agent).

---

```

/* Previously defined interval of the agent */
agent_interval  $\leftarrow [a_{min}, a_{max}[$ 

/* Agent starts the exploration at node  $v_1$  */
current_node  $\leftarrow v_1$ 

/* Array of visited nodes by the agent is initially empty */
visited_nodes  $\leftarrow [ ]$ 

/*  $v_1$  is the root, so its interval comprises the entire range of visits */
intervals[ $v_1$ ]  $\leftarrow [0, 1]$ 

/* Initialize the empty adjacency list of the tree  $A$  */
 $A \leftarrow \emptyset$ 

/* In the tree  $A$ , create a node for  $v_1$ , and fill its children list with the neighboring cells
of  $v_1$ . The children insertion order must follow some immutable rule. In the case of
this work, the insertion is clockwise (North, East, South, West) */
complete( $A, v_1$ )

/* Create a flag to know if the agent filled its interval passing by all the correspondent
nodes in the tree */
finished_interval  $\leftarrow \text{FALSE}$ 

while current_node is not the goal cell do
  if current_node has no child or all of current_node's children were visited then
    if current_node ==  $v_1$  then  $\triangleright$  Unsuccessful search
      break
    else
      current_node  $\leftarrow$  get_parent( $A, \text{current\_node}$ )
      continue
    end if
  end if
  if current_node is not in visited_nodes then
    append(visited_nodes, current_node)
    max  $\leftarrow$  get_max_value(intervals[current_node])
    min  $\leftarrow$  get_min_value(intervals[current_node])  $\triangleright$  Node's mixed radix value
    partition  $\leftarrow$  (max - min) / amount_of_children
    count  $\leftarrow 0$ 
    for all child  $v_j$  of current_node in  $A$  do
      intervals[ $v_j$ ]  $\leftarrow$  [min + count  $\cdot$  partition, min + (count+1)  $\cdot$  partition[
      complete( $A, v_j$ )
      count  $\leftarrow$  count + 1
    end for
  end if

```

---



---

**Algorithm 1** Traverse of the agent through the maze (interpreted as a tree by the agent).

---

```

for all child  $v_j$  of current_node in  $A$  do
    max_node  $\leftarrow$  get_max_value(interval[ $v_j$ ])
    min_node  $\leftarrow$  get_min_value(interval[ $v_j$ ])
    max_agent  $\leftarrow$  get_max_value(agent_interval)
    min_agent  $\leftarrow$  get_min_value(agent_interval)
    if  $v_j$  is in visited_nodes then
        continue
    else if max_agent < min_node then
        /* If the node's interval is on the right of the agent's interval, surely the
           agent has finished its interval. Since the order of the nodes are well defined,
           and in increasing order of intervals, the following condition improve the
           algorithm's performance. It is optional */
        finished_interval  $\leftarrow$  TRUE
        break
    else if min_agent < max_node and max_agent > min_node then
        /* This condition verifies if there is intersection between the agent's interval
           and the node's interval. The order of children must be well defined */
        current_node  $\leftarrow v_j$ 
        break
    end if
end for

/* If current_node remains the same as it was at the beginning of the iteration,
   and it is the root, the agent has finished its interval */
if current_node ==  $v_1$  and finished_interval == FALSE then
    finished_interval  $\leftarrow$  TRUE
end if

/* If the agent completely filled its interval, it needs to change its behavior, since its
   behavior is domain dependent. It might continue the DFS under some criteria.
   Otherwise, it must come back to parent in order to verify if there are more nodes to
   visit */
if finished_interval == TRUE then
    current_node  $\leftarrow$  continue_DFS_under_some_criteria( $A$ , current_node)
else
    current_node  $\leftarrow$  get_parent( $A$ , current_node)
end if
end while

```

---

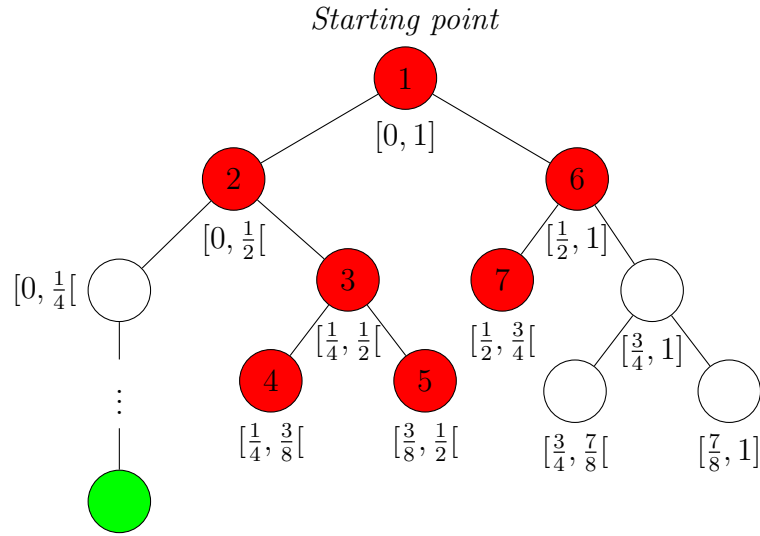


FIGURE 2.8 – An agent with interval  $[\frac{1}{3}, \frac{2}{3}[$  finishes its interval in the visit order indicated. Regardless of the behavior that it assumes after finishing its interval, it never finds the green goal since it doesn't go through already visited nodes, except in backtracking situations. It is a fictional maze.

Thus, the goal of this work aims to establish an efficient algorithm that interrelates agent intervals to node convergence intervals. Furthermore, the authors observed that a mixed radix representation to the agents' path is a powerful mathematical tool for relating such path to the node's convergence interval that the agent is visiting. Section 2.4 discusses the mixed radix representation.

## 2.4 Mixed radix representation to the agent path

In Section 2.3, this work presented a method for an agent to traverse the maze by defining ranges of action. Pseudocode 1 exposes this method and establishes a way to explore the maze while an agent records the visited nodes and the explored tree. However, the agent doesn't need in fact to save all the structure of the tree, because it only needs to know the current node's convergence interval and the current node's edges to make a decision.

This work establishes a method to calculate the node's convergence intervals without the agent having to save necessarily all the structure of the tree. In order to traverse the maze while the agent autonomously makes a decision node by node, it saves its path in a mixed radix numerical representation, where the starting representation is "0.", and it is limited to 1, i.e.,  $[0, 1]$ . Pseudocode 2 proposes a computational method to get the value of mixed radix numerical representation. Equation 2.2 shows examples of numbers under a mixed radix representation.

$$\begin{aligned}
0.0_21_2 &= 0.25 \\
0.0_22_3 &= 0.333... \\
0.0_22_31_2 &= 0.41666... \\
0.1_51_30_21_2 &= 0.28333... \\
0.8_92_72_32_35_8 &\approx 0.93584656
\end{aligned} \tag{2.2}$$

According to the order of analysis of the children and the mathematical tools to calculate node convergence intervals that are presented in Pseudocode 1, the items below present how the agent saves its path based on a mixed radix representation. These points guide the reasoning in Pseudocode 2.

- If the current node has only one child and the agent goes through this child, it appends  $I_1$  to the mixed radix numerical representation of the path;
- If the current node has  $n$  children, where  $n > 1$ , and the agent decides to go through the  $i$ th child, it appends  $(i - 1)_n$  at the end of the mixed radix numerical representation of the path;
- If the agent must go back to the parent, it removes the last appended value from the mixed radix numerical representation of the path.

It is important to note that the value of the mixed radix numerical representation is the same as the minimum value of the convergence interval of the node that the agent is visiting. Moreover, the mixed radix numerical representation carries information about the maximum value of the node's convergence interval, since it is the sum of the minimum value with the lowest possible value of the current mixed radix representation. For instance, if the agent's current path is  $0.1_30_22_4$ , the minimum value of the current node's convergence interval is  $0.1_30_22_4$ , and the maximum is  $0.1_30_23_4$  ( $0.1_30_22_4 + 0.0_30_21_4$ ). Figure 2.9 presents an example of the approach of a mixed radix numerical representation, where the agent's path carries information about the current node's convergence interval.

## 2.5 Extended Tarry's algorithm

Rao et al. (1999) explain that Tarry's algorithm is a graph search algorithm where a single agent passes through each edge of a graph in order to visit all of its vertex, going through each edge once and only once in each direction. As Rao et al. (1999) state, it is similar to the common depth-first search algorithm used in graphs. Thus, if a maze is structured as a graph, Tarry's algorithm might be a tool to solve such maze.

---

**Algorithm 2** Methods related to mixed radix numerical representation.

---

```

/* Structure of mixed radix element */
struct mr_element
{
    type numeral
    type radix
}

/* Initialize the agent's path. The agent is at the tree's root. Numerically it means zero
in the mixed radix representation */
path ← [ ]

procedure APPEND(path, numeral, radix):
    struct mr_element element
    element.numeral ← numeral
    element.radix ← radix
    add(path, element)           ▷ Append the element to the end of the array path
end procedure

procedure REMOVE(path):
    subtract(path)               ▷ Remove the last element of the array path
end procedure

procedure GET_MIXED_RADIX_VALUE(path):
    value ← 0
    interval_size ← 1           ▷ In order to calculate the interval size step by step
    for all element in path do
        if element.numeral == I then           ▷ "I" represents a node with only one child
            continue
        end if
        value ← value + interval_size · (element.numeral / element.radix)
        interval_size ← interval_size / element.radix
    end for
    return value
end procedure

```

---

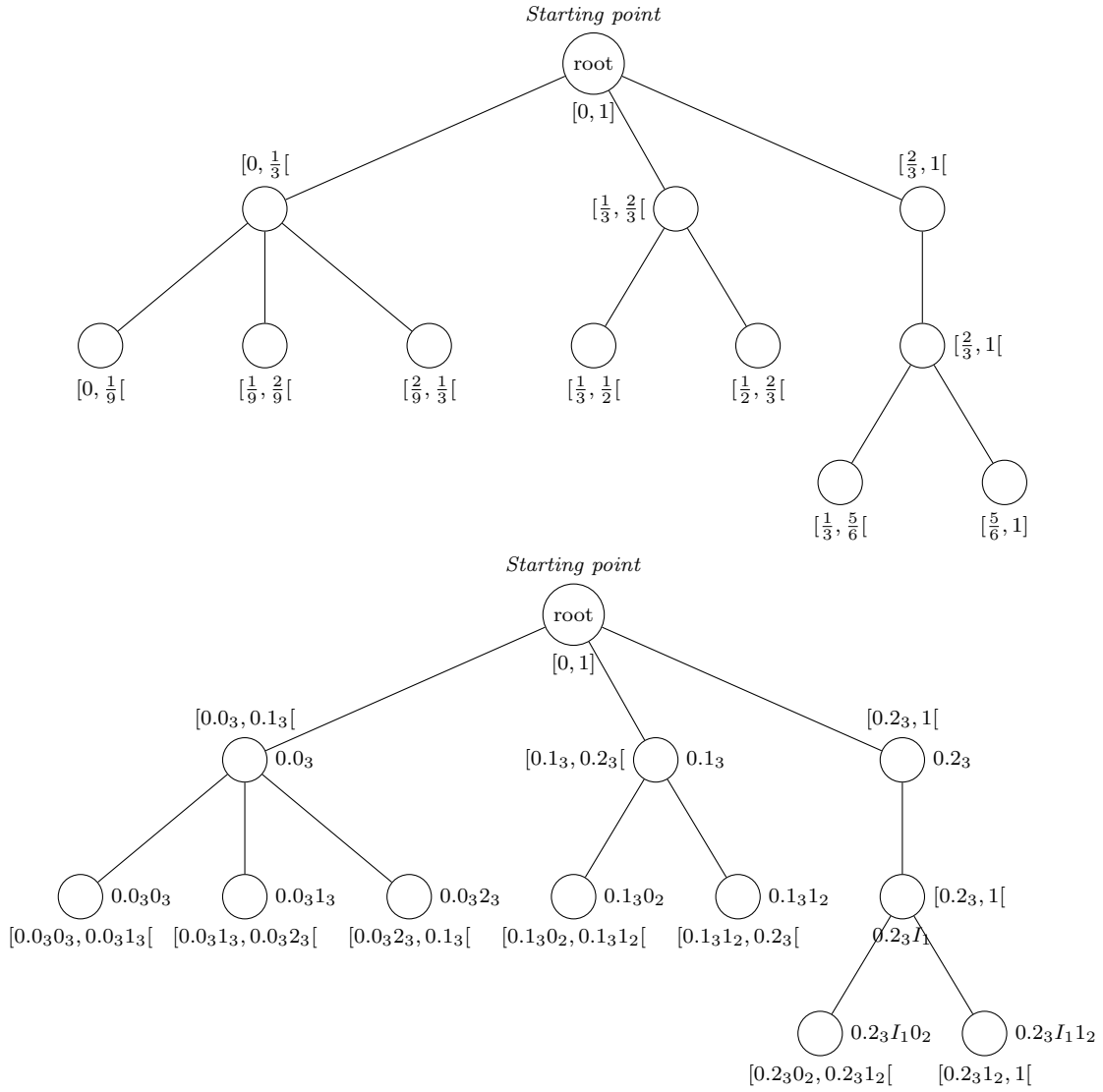


FIGURE 2.9 – Both trees have the same format, but the first one has its node convergence intervals presented in a fractional representation, and the second one has its node convergence intervals presented in a mixed radix representation. As the agent's path carries information about the nodes' convergence intervals when the path is structured in a mixed radix representation, it is possible to save, with only one numerical representation, information about the tree explored by the agent and the convergence intervals of the tree's nodes. It is a fictional maze.

Kivelevitch & Cohen (2010) propose an extended version of Tarry's algorithm to solve a maze where a set of agents tries to find the goal cooperatively. Such agents have no *a priori* knowledge about the maze's structure, but they can share information dynamically about the cells explored by each agent. Therefore, the authors present a multi-agent maze-solving algorithm with communication, whose performance was compared to our algorithm's performance, even though the agents cannot share information in the scope of this work.

Basically, to solve a maze in a multi-agent context, Kivelevitch & Cohen (2010) pro-

gram each agent to follow a generalization of Tarry’s algorithm. However, all the visited nodes are shared with each agent since the agents communicate with each other. Thus, the following steps guide the reasoning of the proposed algorithm (KIVELEVITCH; COHEN, 2010):

- The agent should move to cells that have not been traveled by any agent.
- If there are several such cells, the agent should choose one arbitrarily.
- If there is no cell that has not been traveled by an agent, the agent should prefer to move to a cell that has not been traveled by it.
- If all the possible directions have already been traveled by the agent, or if the agent has reached a dead-end, the agent should retreat until a cell that meets one of the previous conditions.
- All the steps should be logged by the agent in its history.
- When retreating, mark the cells retreated from as “dead end”.

Furthermore, the algorithm has a second phase: given the path of the pioneer agent that first finds the goal, the other agents must backtrack to the last location that matches a location on the pioneer’s path in order to achieve the goal as the pioneer did. So, when the pioneer finds the goal, no more cells are explored, since the other agents come back to the “Last Common Location (LCL)” relating to the pioneer and finally follow the pioneer’s path until the goal.

This work used this maze-solving method for comparison to our algorithm, as exposed in Section 3.3. To know more about the aforementioned generalization of Tarry’s algorithm, please see Kivelevitch & Cohen (2010).

## 3 Results and Discussion

We developed a code in Python based on the methods exposed in Section 2.1 for assessing our algorithm’s performance. Pseudocode 1 and Pseudocode 2 guided the reasoning behind the code, that is available on [github.com/ArthurJose2000/mazeexploration](https://github.com/ArthurJose2000/mazeexploration).

This work used perfect mazes for simulating multi-agent scenarios where agents go through the mazes in order to find the goal. Section 3.1 presents the agents’s performance when they use our algorithm. Section 3.2 proposes an agent policy modification when it finishes its interval for improving its performance. Finally Section 3.3 compares our algorithm’s performance to the performance of the extended Tarry’s algorithm (KIVELEVITCH; COHEN, 2010), despite the latter having communication between agents.

### 3.1 Our algorithm’s performance

In order to assess our algorithm’s performance, this work generates, using the code provided by Naeem (2021), 250 random and perfect mazes for each type of size -  $10 \times 10$ ,  $20 \times 20$ ,  $30 \times 30$ , and  $40 \times 40$ . As they are perfect mazes, i.e., there is only one path to the goal from any cell, any node of the maze’s tree representation has the same convergence interval for every agent. Thus, the developed code aims to fill the concept of dispersion, where the agents try to go through the maze as dispersed as possible without communication.

For each maze size, the code ran 1 single agent until 40 cooperative agents going through 250 different mazes, since this work generated 250 random perfect mazes for each type of size. In order to analyse the results, the authors computed the following classes of analysis:

- Average of steps: the average of steps of one agent in each maze, where the total number of steps is divided by the number of agents.
- Pioneer’s average of steps: the average of steps of the first agent that reaches the goal, i.e., the pioneer.

- STDEV - Average of steps: the standard deviation of the average of steps of one agent considering all the 250 different mazes.
- Fraction of maze explored: the percentage of visited cells until every agent stops.
- Fraction of maze explored when pioneer reaches target: the percentage of visited cells at the moment when the pioneer reaches the goal.

As pointed out in Section 2.3.1, where the key concepts of this research are exposed, it is worth emphasizing that some agents doesn't find the goal, so it is possible that some agent stops the search before the pioneer finds the goal. It is clearer for big mazes, as seen in Figure 3.1d, where in some situations the average of steps is smaller than the pioneer's average of steps, that might seem incoherent at first looking. If the approach of Pseudocode 1 is modified so that each agent must find the goal after fills its interval, obviously the average of steps must be bigger than the pioneer's average of steps.

Figure 3.1 presents, for each maze size -  $10 \times 10$ ,  $20 \times 20$ ,  $30 \times 30$ , and  $40 \times 40$  -, the results of average of steps, pioneer's average of steps, and standard deviation of the average of steps considering all the 250 different mazes. Similarly, Figure 3.2 presents the results of fraction of explored maze and fraction of maze explored when pioneer reaches the target.

## 3.2 Incremental policy for the agents

After establishing an algorithm where the agents focus to fill its interval, this work implemented a slight modification, where the agent change its interval and its order of action after finishing its starting interval.

As pointed out in Pseudocode 1, the order of action of each agent is clockwise in the context of maze - North, East, South, and West. Supposing that there are  $k$  agents  $a_1, a_2, \dots, a_k$  to explore the maze, there will be  $k$  different acting intervals, as seen in Equation 2.1. With our incremental policy, when the agent  $a_i$  finishes its interval, its interval gets the value of the starting interval of  $a_{i+1}$  if  $i < k$ . If  $i = k$ , it gets the value of the starting interval of  $a_0$ . Furthermore, the agent changes its order of action from clockwise to counter-clockwise - West, South, East, and North -, i.e, from right to left relating to the order of children.

Figure 3.3 gives an example of such incremental policy, where an agent with interval  $[0, \frac{1}{3}[$  finishes its interval in the 5th node, and then it changes its interval to  $[\frac{1}{3}, \frac{2}{3}[$ , and changes the order of action from left-right to right-left.



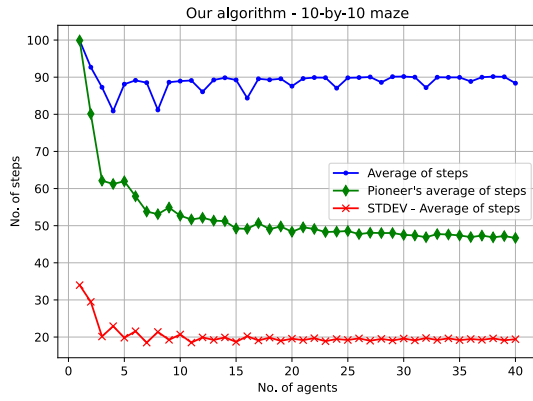
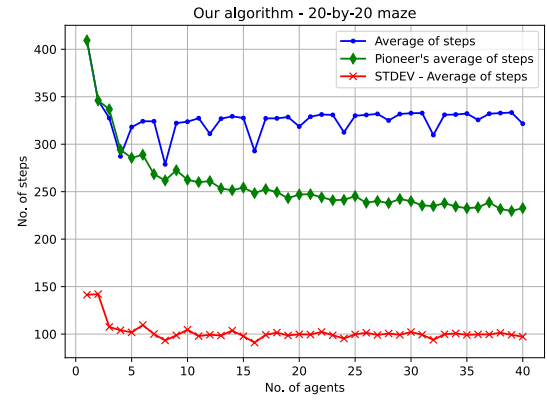
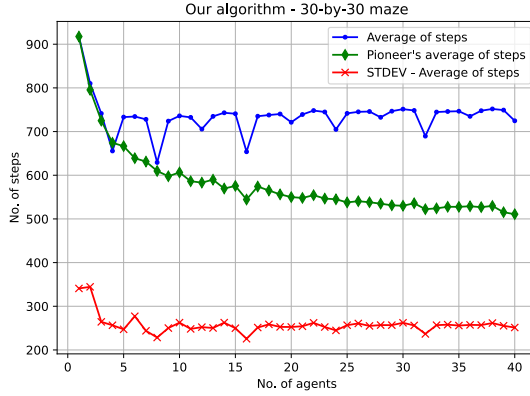
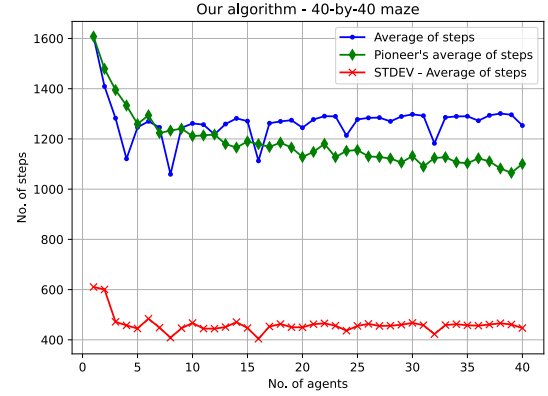
(a)  $10 \times 10$  maze(b)  $20 \times 20$  maze(c)  $30 \times 30$  maze(d)  $40 \times 40$  maze

FIGURE 3.1 – Results of average of steps, pioneer's average of steps, and standard deviation of the average of steps. 1 single agent until 40 cooperative agents were run in 250 different mazes for each type of size.

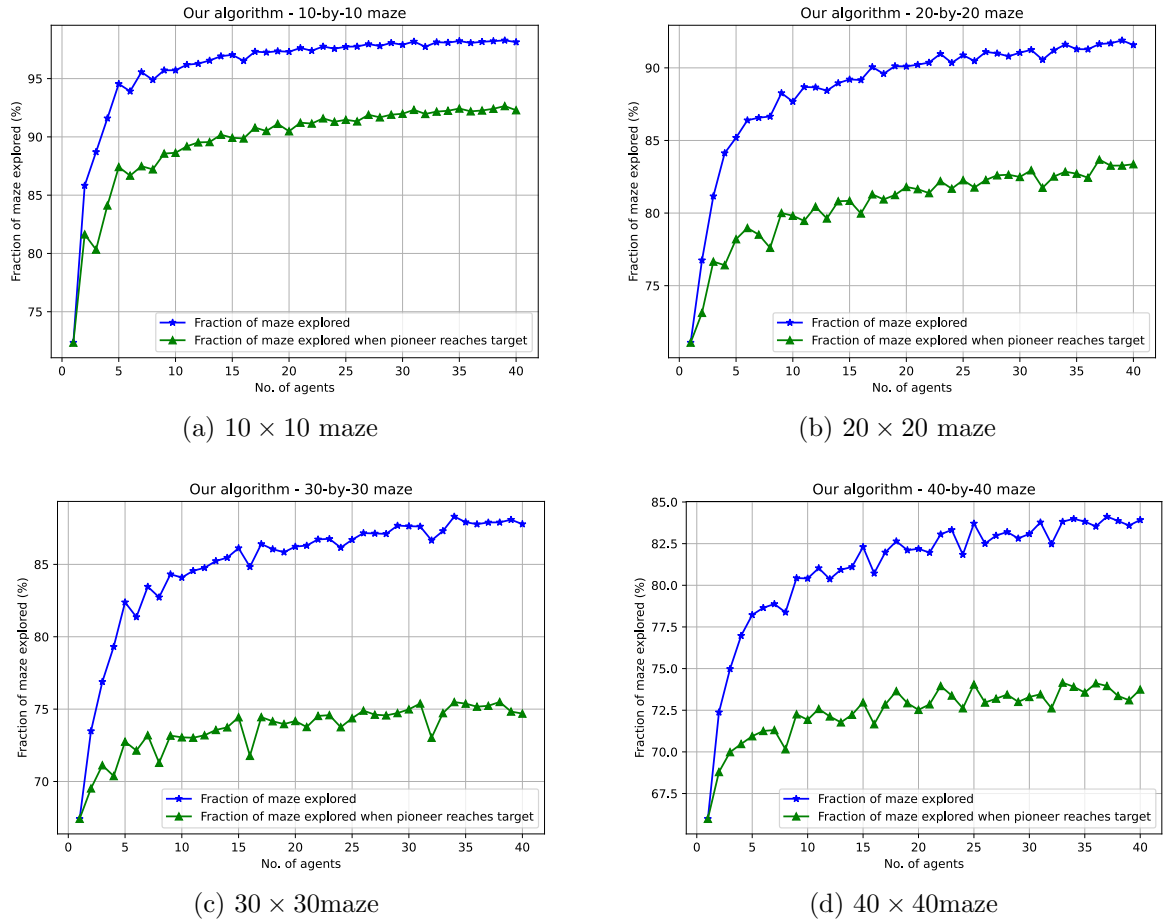


FIGURE 3.2 – Results of fraction of explored maze and fraction of maze explored when pioneer reaches the target. 1 single agent until 40 cooperative agents were run in 250 different mazes for each type of size.

Figures 3.4 and 3.5 present the comparison between our “1 interval” algorithm - evaluated in Section 3.1 - and our “2 intervals” algorithm, where the incremental policy was implemented.

The first conclusion that might be taken from the incremental policy is that it improves the search in the sense of amount of steps the a pioneer must do to reach the target. For instance, as seen in Figure 3.4e, in the context of 40 cooperative agents in  $40 \times 40$  mazes, with the incremental policy, the pioneer reaches the target with about 18% less steps compared to the our “1 interval” approach. Moreover, as such policy improves the dispersion, more cells are visited, as seen in Figure 3.5.

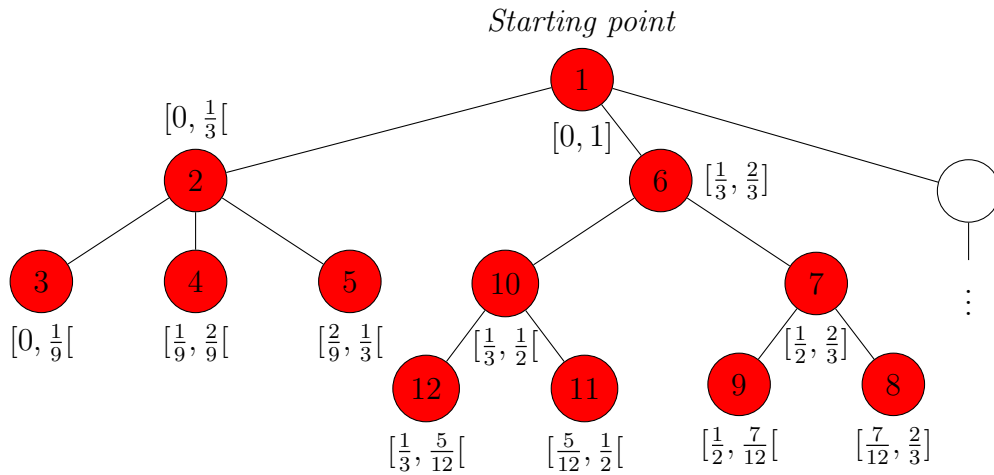


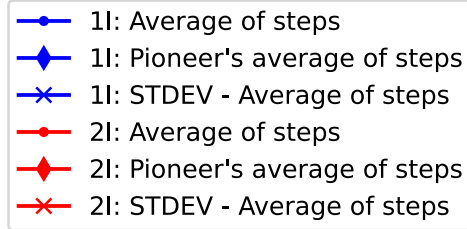
FIGURE 3.3 – Example of an agent with interval  $[0, \frac{1}{3}[$  under the incremental policy, where it changes its interval to  $[\frac{1}{3}, \frac{2}{3}[$  when it finishes its starting interval, in the 5th visited cell, and it changes the order of action to right-left. It is a fictional maze.

### 3.3 Our algorithm vs. Tarry’s algorithm

Finally our algorithm was compared to the extended Tarry’s algorithm (KIVELEVITCH; COHEN, 2010), that considers all agents with the capability to communicate with each other. As pointed out in Section 2.5, the extended Tarry’s algorithm also aims to find the goal node of some undirected graph. Thus, this work implemented such algorithm for testing in the same mazes previously generated.

It is important to emphasize that, differently from the extended Tarry’s algorithm, our algorithm considers that the agents cannot communicate with each other. Furthermore, Kivelevitch & Cohen (2010) use the “Last Common Location” concept, where the agents are able to come back to the last location that matches on the pioneer’s path, and then they go through the maze following the pioneer’s path to also achieve the goal.

Figure 3.6 compares the pioneer’s average of steps of the three algorithms, i.e., our “1



(a) Legend

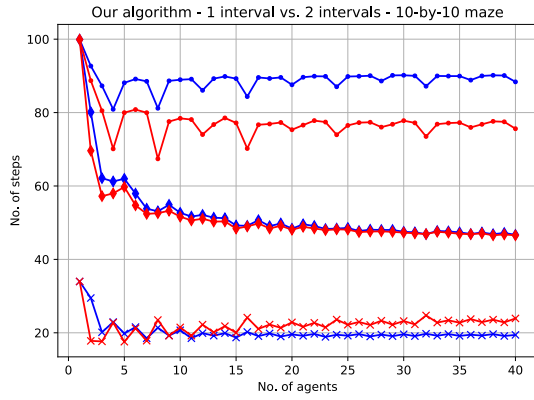
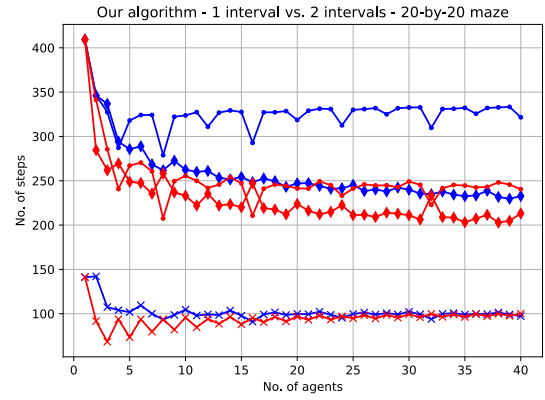
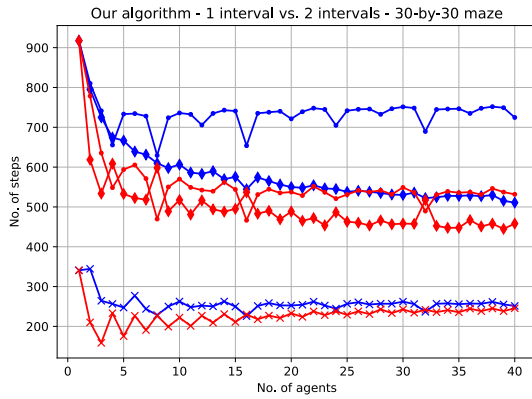
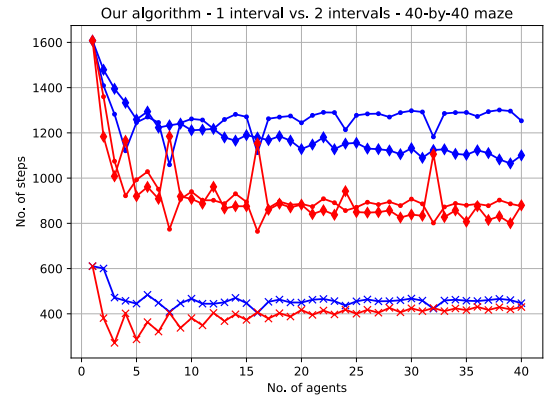
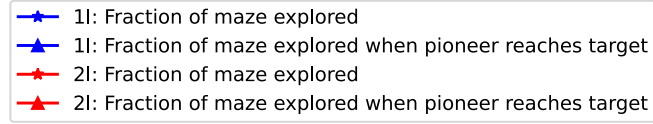
(b)  $10 \times 10$  maze(c)  $20 \times 20$  maze(d)  $30 \times 30$  maze(e)  $40 \times 40$  maze

FIGURE 3.4 – Comparison between the results of average of steps, pioneer's average of steps, and standard deviation of the average of steps relating to our “1 interval” algorithm and our “2 intervals” algorithm. 1 single agent until 40 cooperative agents were run in 250 different mazes for each type of size, and for each algorithm.



(a) Legend

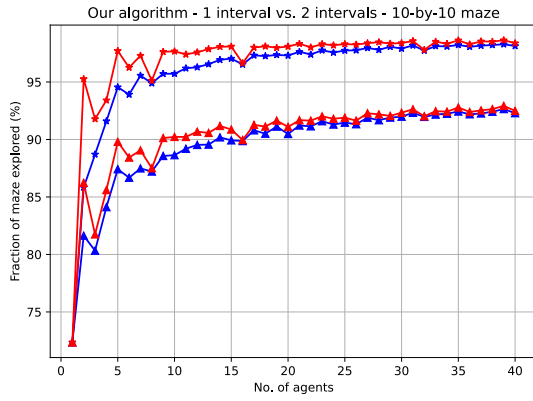
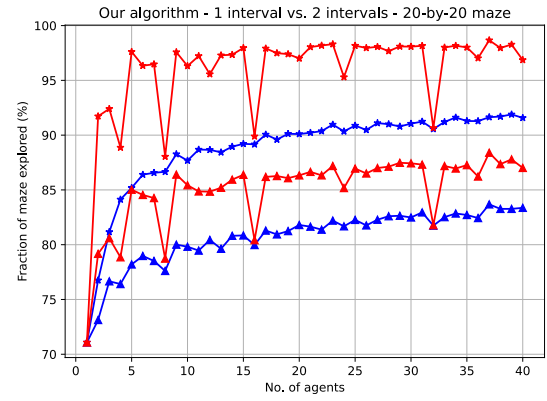
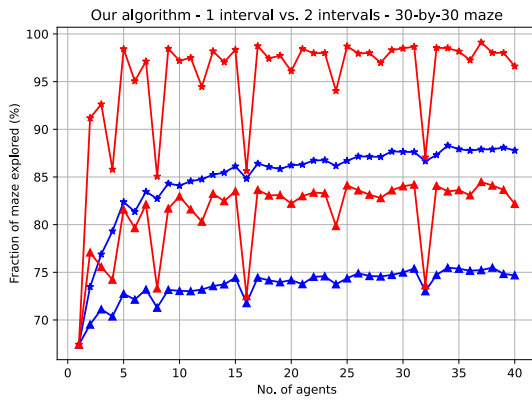
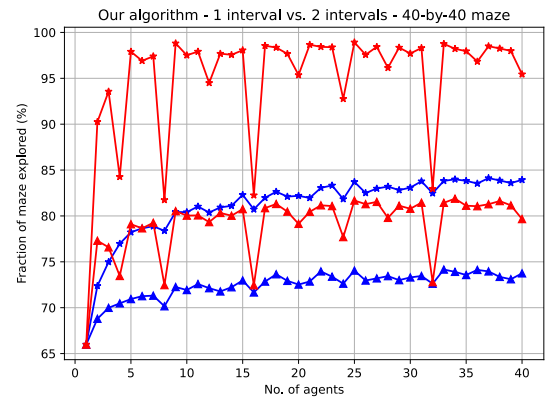
(b)  $10 \times 10$  maze(c)  $20 \times 20$  maze(d)  $30 \times 30$  maze(e)  $40 \times 40$  maze

FIGURE 3.5 – Comparison between the results of fraction of explored maze and fraction of maze explored when pioneer reaches the target relating to our “1 interval” algorithm and our “2 intervals” algorithm. 1 single agent until 40 cooperative agents were run in 250 different mazes for each type of size, and for each algorithm.

interval” algorithm, whose results were exposed in Section 3.1, our “2 intervals” algorithm, whose results were exposed in Section 3.2, and the extended Tarry’s algorithm. Similarly, Figure 3.7 compares the fraction of maze explored when pioneer reaches the target. As seen, this work doesn’t compare the average of steps of the three algorithms, since all the agents in the extended Tarry’s algorithm reach the goal, and it also doesn’t compare the fraction of maze explored, since the agents stop the exploration when the pioneer finds the goal, once they come back to the “Last Common Location” to follow the pioneer’s path.

For reaching the goal, the performance of the extended Tarry’s algorithm is better than our algorithm, since the first improves the agent dispersion mainly because of the network of visited cells shared with each agent, in which they are able to make the decision of avoiding paths that have already been covered. Fortunately, our algorithm is open to improvements, since it can accept new policies to guide the agent after it finishes its interval. As seen in Section 3.2, our incremental policy improves our algorithm performance, since the pioneer finds the goal faster than our algorithm without incremental policy. Moreover, as seen in Figure 3.7, the agents explore more maze cells in the context of the extended Tarry’s algorithm, since it boosts the dispersion.

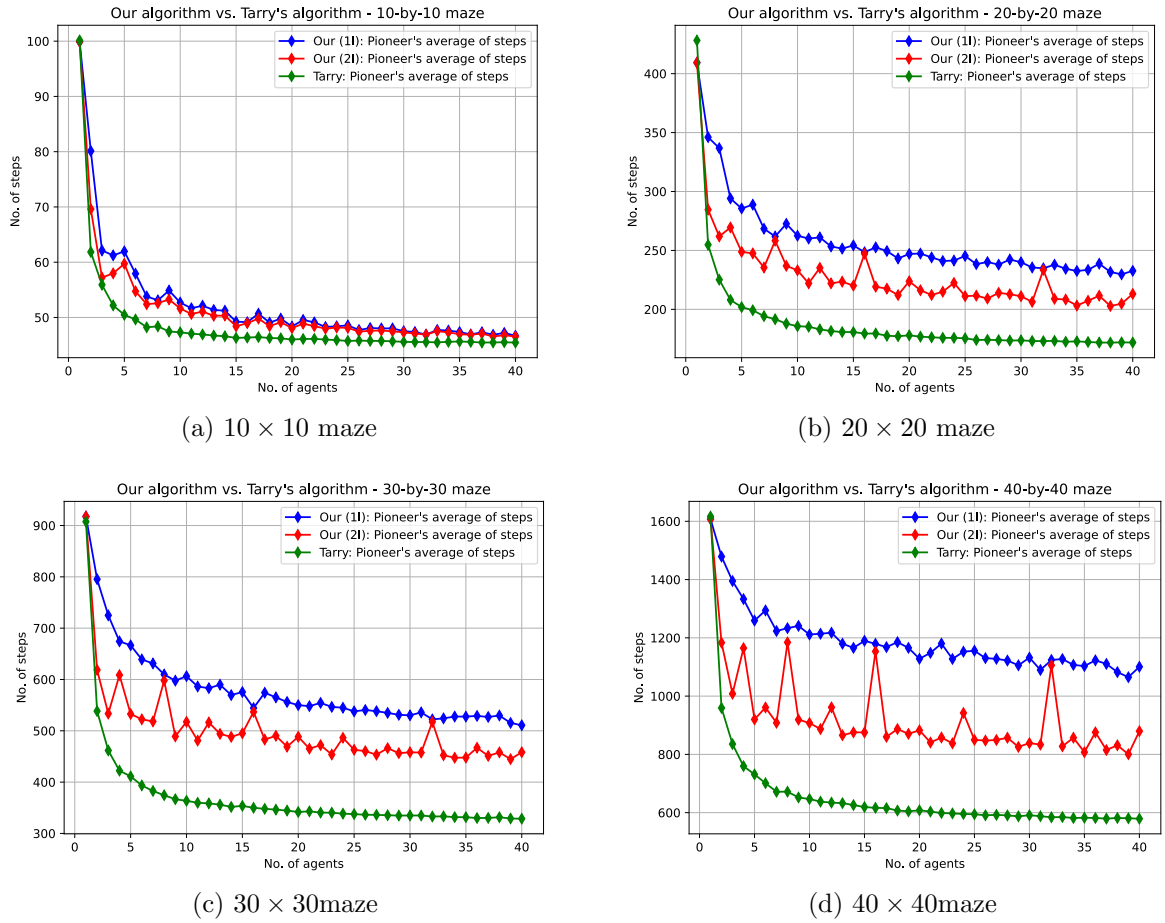


FIGURE 3.6 – Comparison between the results of pioneer's average of steps relating to our algorithms and the extended Tarry's algorithm. 1 single agent until 40 cooperative agents were run in 250 different mazes for each type of size, and for each algorithm.

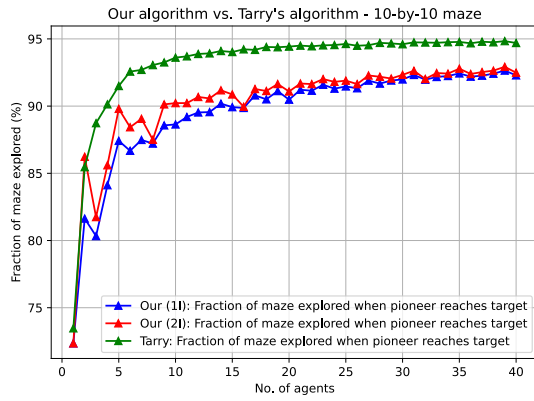
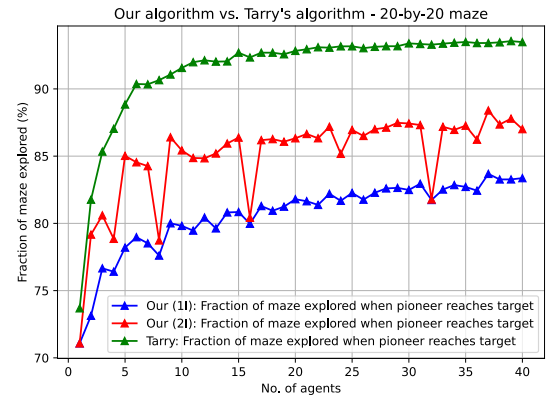
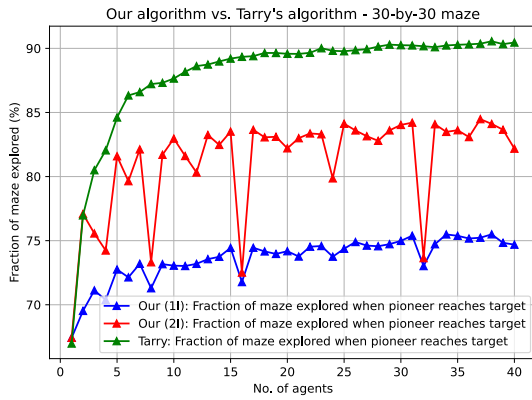
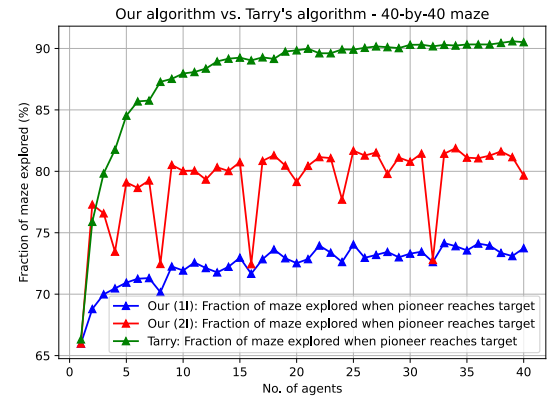
(a)  $10 \times 10$  maze(b)  $20 \times 20$  maze(c)  $30 \times 30$  maze(d)  $40 \times 40$  maze

FIGURE 3.7 – Comparison between the results of fraction of maze explored when pioneer reaches the target relating to our algorithms and the extended Tarry's algorithm. 1 single agent until 40 cooperative agents were run in 250 different mazes for each type of size, and for each algorithm.



## 4 Conclusions and Future Works

Next steps has been discussed with the advisor and the co-advisor, and the main points are described below.

- Until 18th August: to find out a tree simulation application;
- Until 1st September: to establish a mixed radix representation to the agent path;
- Until 6th October: to compare the proposed algorithm with well-known algorithms in the context of maze-solving challenges;
- Until 10th November: to generalize the proposed algorithm to search algorithms in graphs.

boa op~£o para ser alvo de compara~£o co outros algoritmos futuros em que n~£ h~; comunica~£o

# Bibliography

Arndt, J. (2011). **Mixed radix numbers**. In: Matters Computational. Springer, Berlin, Heidelberg. [https://doi.org/10.1007/978-3-642-14764-7\\_9](https://doi.org/10.1007/978-3-642-14764-7_9)

Beisel, B. W. (2014). **Distributed Maze Solving By Cooperative Robotic Platforms**. University of Maryland, Department of Electrical and Computer Engineering.

Burgard, W., Moors, M., Stachniss, C., Schneider, F. E. (2005). **Coordinated multi-robot exploration**. IEEE Transactions on Robotics, 21(3), 376-386. doi:10.1109/tro.2004.839232

Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L. and Stein, Clifford. **Introduction to Algorithms**. 4th : The MIT Press, 2022.

Kivelevitch, Elad & Cohen, Kelly. (2010). **Multi-Agent Maze Exploration**. Journal of Aerospace Computing Information and Communication. 7. 391-405. 10.2514/1.46304.

Matarić, M. J. (1995). **Designing and Understanding Adaptive Group Behavior**. Adaptive Behavior, 4(1), 51-80. doi:10.1177/105971239500400104

Manber, U. **Introduction to Algorithms: A Creative Approach** 1th : Addison-Wesley, 1989.

Naeem, M. A. **pyamaze**. GitHub, 2021. Available at: <<https://www.github.com/MAN1986/pyamaze>>. Accessed on: June 6, 2023.

Rao, Nageswara & Kareti, Srikumar & Shi, Weimin & Iyengar, Sundararaj. (1999). **Robot Navigation in Unknown Terrains: Introductory Survey of Non-Heuristic Algorithms**. 10.2172/10180101.

Sadik, A. M. J., Dhali, M. A., Farid, H. M. A. B., Rashid, T. U., Syeed, A. (2010). **A Comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory**. 2010 International Conference on Artificial Intelligence and Computational Intelligence. doi:10.1109/aici.2010.18.

Shields, R. (2012). **Cultural Topology: The Seven Bridges of Königsberg, 1736**. Theory, Culture & Society, 29(4-5), 43-57. doi:10.1177/0263276412451161.

## FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO TC	2. DATA 06 de junho de 2023	3. DOCUMENTO Nº DCTA/ITA/DM-000/2023	4. Nº DE PÁGINAS 49
5. TÍTULO E SUBTÍTULO: Multi-agent graph exploration without communication			
6. AUTOR(ES): <b>Arthur José de Sousa Rodrigues</b>			
7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES): Instituto Tecnológico de Aeronáutica – ITA			
8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR: Maze; Graph; Search; Multi-agent			
9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO: Maze; Graph; Search; Multi-agent			
10. APRESENTAÇÃO: <span style="float: right;">(X) Nacional ( ) Internacional</span> ITA, São José dos Campos. Curso de Engenharia de Computação. Orientador: Prof. Dr. Luiz Gustavo Bizarro Mirisola. Coorientador: Prof. Dr. Vitor Venceslau Curtis. Defesa em 00/00/2023. Publicada em 00/00/2023.			
11. RESUMO: <p>Algoritmos computacionais de busca têm sido estudados por cientistas e companhias de engenharia desde o último século devido às aplicações de tais algoritmos em situações reais, como agendamento de linhas aéreas, planejamento de rota em mapas, algoritmos de busca na Internet, planejamento de rotas em redes de computadores, robótica, etc. Usualmente, esses métodos se baseiam em tipos abstratos de dados, como grafos e árvores, para transferir um problema real para dentro de um contexto delimitado e inteligível. No campo da Ciência da Computação, grafos são tipos abstratos de dados que podem servir como suporte ferramental para algoritmos de busca.</p> <p>De uma abordagem relacionada a algoritmos de busca em labirintos, este trabalho de graduação propõe um método para explorar grafos em um âmbito multiagente, em que os agentes não são capazes de se comunicarem entre eles, o que faz referência a problemas da realidade, como exploração no fundo do mar, busca em estruturas compostas por muros maciços, busca em locais inóspitos por agentes com restrição energética, etc.</p>			
12. GRAU DE SIGILO: (X) OSTENSIVO ( ) RESERVADO ( ) SECRETO			