

Universidade Federal da Paraíba  
Centro de Informática  
Programa de Pós-Graduação em Informática

ST-SPF & STMS: Two new Algorithms for Path Finding in  
Robotic Mobile Fulfillment Systems

Ítalo Renan da Costa Barros

Dissertação submetida à Coordenação do Curso de Pós-Graduação em  
Informática da Universidade Federal da Paraíba como parte dos requisi-  
tos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação  
Linha de Pesquisa: Sinais, Sistemas Digitais e Gráficos

Tiago Pereira do Nascimento  
(Orientador)

Luís Felipe Silva Costa  
(Co-Orientador)

João Pessoa, Paraíba, Brasil

©Ítalo Renan da Costa Barros, 15 de Junho de 2021

**Catálogo na publicação**  
**Seção de Catalogação e Classificação**

B277s Barros, Ítalo Renan da Costa.  
ST-SPF & STMS : two new algorithms for path finding  
in  
Robotic Mobile Fulfillment Systems / Ítalo Renan da  
Costa Barros. - João Pessoa, 2021.  
120 f. : il.

Orientação: Tiago Pereira do Nascimento.  
Coorientação: Luís Felipe Silva Costa.  
Dissertação (Mestrado) - UFPB/PPGI.

1. Informática. 2. RMFS. 3. MAPF. 4. Path Planning.  
5.  
Multi-Agent Systems. 6. Robotized Warehouses. I.  
Nascimento, Tiago Pereira do. II. Costa, Luís Felipe  
Silva. III. Título.

UFPB/BC

CDU 004(043)




*Universidade Federal da Paraíba*  
**PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA**

**ATA Nº 8**

Aos dezenove dias do mês de março do ano de dois mil e vinte e um, às nove horas, por meio de videoconferência, reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final de ÍTALO RENAN DA COSTA BARROS, vinculado a esta Universidade sob a matrícula nº 20191000951, candidato ao grau de Mestre em Informática do Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba. A Comissão Examinadora foi composta pelos professores doutores TIAGO PEREIRA DO NASCIMENTO (UFPB), Orientador e Presidente da Banca; GILBERTO FARIAS DE SOUSA FILHO (UFPB), Examinador Interno; TEOBALDO LEITE BULHÕES JUNIOR (UFPB), Examinador Interno; e ANDRÉ LUÍS MARQUES MARCATO (UFJF), Examinador Externo. Dando início aos trabalhos, o Presidente da Banca cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse a exposição oral do trabalho de dissertação intitulado ST-SPF E STMS: DOIS NOVOS ALGORITMOS PARA DESCOBERTA DE CAMINHOS EM SISTEMAS DE PREENCHIMENTO MÓVEL ROBÓTICO. Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o parecer APROVADO. Do ocorrido, eu, TIAGO PEREIRA DO NASCIMENTO, Presidente da Comissão Examinadora, lavrei a presente ata que vai assinada por mim e pelos membros da banca examinadora.

  
**Dr. ANDRÉ LUÍS MARQUES MARCATO, UFJF**

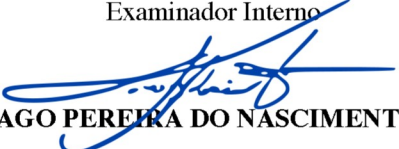
Examinador Externo à Instituição

  
**Dr. GILBERTO FARIAS DE SOUSA FILHO, UFPB**

Examinador Interno

  
**Dr. TEOBALDO LEITE BULHÕES JUNIOR, UFPB**

Examinador Interno

  
**Dr. TIAGO PEREIRA DO NASCIMENTO, UFPB**

Presidente

  
**ÍTALO RENAN DA COSTA BARROS**

Mestrando



*Universidade Federal da Paraíba*  
**PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA**

**FOLHA DE CORREÇÕES**

**ATA Nº 8**

**Autor:** ÍTALO RENAN DA COSTA BARROS

**Título:** ST-SPF e STMS: dois novos algoritmos para descoberta de caminhos em sistemas de preenchimento móvel robótico

**Banca examinadora:**

Prof. ANDRÉ LUÍS MARQUES MARCATO

Examinador Externo à Instituição

Prof. GILBERTO FARIAS DE SOUSA FILHO

Examinador Interno

Prof. TEOBALDO LEITE BULHÕES JUNIOR

Examinador Interno

Prof. TIAGO PEREIRA DO NASCIMENTO

Presidente

---

Os itens abaixo deverão ser modificados, conforme sugestão da banca examinadora.

COMENTÁRIOS GERAIS:

---

Declaro, para fins de homologação, que as modificações, sugeridas pela banca examinadora, acima mencionadas, foram aceitas e serão cumpridas integralmente.

**Prof. TIAGO PEREIRA DO NASCIMENTO**

Orientador



## Resumo

Um dos principais problemas enfrentado no Multi-Agent Path Finding aplicado a Robotic Mobile Fulfillment Systems é como trazer uma maior escalabilidade ao sistema conforme aumentamos o número de agentes. Este trabalho tem como objetivo propor dois novos algoritmos offline, o algoritmo descentralizado Space-Time Swarm Path Finding (ST-SPF), e o algoritmo centralizado Space-Time Multi-Start (STMS). Os algoritmos foram testados em um simulador desenvolvido no framework PyGame, onde foram realizados experimentos com até 250 agentes em três tipos de warehouses (instâncias) diferentes, e com dois tipos representações do mapa: Grid-Based e Graph-Based. Os resultados demonstram que o ST-SPF é escalável em instâncias grandes e populosas, alcançando até 48% de redução do tempo de execução quando comparado com o algoritmo de estudo da arte Conflict-based Search (CBS), enquanto que o STMS apresentou uma vantagem ao CBS por ser mais completo (*completeness*) em instâncias pequenas e populosas. Por fim, também foi notado que a utilização da representação Graph-Based possui uma alta utilização de memória para instâncias complexas (acima de 600 nós), sendo a representação Grid-Based mais eficiente.

**Palavras-chave:** RMFS, MAPF, Path Planning, Sistemas Multi-Agentes, Warehouses Robotizadas.

## Abstract

One of the main problems faced in Multi-Agent Path Finding (MAPF) applied to Robotic Mobile Fulfillment Systems (RMFS) is how to bring greater scalability as we increase the number of agents in the system. This work aims to propose two new offline algorithms, the Space-Time Swarm Path Finding (ST-SPF) a decentralized algorithm, and the Space-Time Multi-Start (STMS), an centralized algorithm. The algorithms were tested in a simulator developed in the PyGame framework, with up to 250 agents in three different types of warehouses (instances) and two types of map representations: Grid-Based and Graph-Based. The results show that the ST-SPF is scalable in complex and populous maps, achieving up to 48% reduction in execution time when compared to the Conflict-based Search (CBS) art study algorithm, while the STMS presented an advantage to CBS since achieves more completeness in small and populous instances. Finally, it was also noted that the use of the Graph-Based representation has a high use of memory for complex instances (above 600 nodes), with the Grid-Based representation being the most efficient.

**Keywords:** RMFS, MAPF, Path Planning, Multi-Agent Systems, Robotized Warehouses.

## **Agradecimentos**

À minha mãe, Cecília Barros, a maior maestra da minha vida e que sempre me mostrou que podemos levar conosco um sorriso no rosto independente da dificuldade que enfrentamos no caminho.

Ao meu pai, José Barros, o meu maior exemplo de perseverança e quem sempre me incentivou a dar um passo a mais independente do desafio.

À minha esposa, Luanna Barros, que durante todo este árduo processo me apoiou durante todos os altos e baixos me incentivando a permanecer firme e focado a continuar.

Ao meu orientador Dr. Tiago Nascimento, pela enorme paciência e por todos os ensinamentos durante todo este trajeto, um exemplo de educador e pesquisador que levarei como exemplo ao longo de toda minha trajetória.

Ao meu Co-orientador Dr. Luís Felipe, pelo enorme apoio no início do mestrado, ensinando e me preparando para os desafios que seriam enfrentados.

Aos professores Teobaldo Júnior e Gilberto Filho que me ajudaram imensamente no desenvolvimento desta pesquisa com suas precisas e importantes críticas e pontuações.

Ao CNPq pelo apoio financeiro concedido.

Aos meus amigos Vannuty Cabral, Vinicius Medeiros, João Vitor, e Erivan Filho pelo apoio durante toda esta jornada.

Aos demais membros e amigos do LaSER que me ajudaram a trilhar este caminho.

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                                   | <b>1</b>  |
| 1.1      | Motivation . . . . .                                  | 1         |
| 1.2      | Objectives . . . . .                                  | 3         |
| 1.3      | Contributions . . . . .                               | 4         |
| 1.4      | Dissertation Structure . . . . .                      | 4         |
| <b>2</b> | <b>Theoretical Concepts and Foundations</b>           | <b>6</b>  |
| 2.1      | Path Planning for Autonomous Mobile Robots . . . . .  | 6         |
| 2.1.1    | Path Planning Schemes and Approaches . . . . .        | 7         |
| 2.1.2    | Common Path Planning Algorithms . . . . .             | 17        |
| 2.1.3    | The Multi-Agent Path Finding (MAPF) Problem . . . . . | 23        |
| 2.2      | Warehouses Systems, Designs and Typologies . . . . .  | 28        |
| 2.2.1    | Warehouse Typologies . . . . .                        | 30        |
| 2.2.2    | Warehouse Layouts . . . . .                           | 32        |
| 2.2.3    | Warehouse Systems . . . . .                           | 34        |
| 2.2.4    | Robotic Mobile Fulfillment Systems (RMFS) . . . . .   | 35        |
| 2.2.5    | System Architecture . . . . .                         | 37        |
| 2.2.6    | Performance Improvements . . . . .                    | 38        |
| 2.2.7    | Discussion . . . . .                                  | 39        |
| <b>3</b> | <b>Related Researches</b>                             | <b>40</b> |
| 3.1      | Multi-Agent Path Finding . . . . .                    | 40        |
| 3.2      | Path Planning Algorithms in RMFS . . . . .            | 43        |
| 3.3      | Discussion . . . . .                                  | 46        |

|          |   |           |
|----------|---|-----------|
| <b>4</b> | <b>The ST-SPF &amp; STMS Algorithms</b>                       | <b>49</b> |
| 4.1      | Low-Level Algorithms . . . . .                                | 49        |
| 4.1.1    | The Breadth-First Search Algorithm . . . . .                  | 49        |
| 4.1.2    | The Dijkstra's Algorithm . . . . .                            | 50        |
| 4.1.3    | The A* Algorithm . . . . .                                    | 50        |
| 4.1.4    | Low-Level Algorithms Comparison . . . . .                     | 50        |
| 4.2      | The Space-Time Swarm Path-Finder (ST-SPF) Algorithm . . . . . | 55        |
| 4.2.1    | The ST-SPF Swarms . . . . .                                   | 55        |
| 4.2.2    | Discussion . . . . .  | 59        |
| 4.3      | The Space-Time Multi-Start Algorithm (STMS) . . . . .         | 60        |
| 4.3.1    | Discussion . . . . .  | 63        |
| <b>5</b> | <b>Experimental Evaluation</b>                                | <b>65</b> |
| 5.1      | Traditional Horizontal . . . . .                              | 66        |
| 5.1.1    | Tiny Layout . . . . .   | 66        |
| 5.1.2    | Small Layout . . . . .  | 68        |
| 5.1.3    | Medium Layout . . . . .                                       | 70        |
| 5.1.4    | Large Layout . . . . .  | 72        |
| 5.2      | Traditional Vertical . . . . .                                | 74        |
| 5.2.1    | Tiny Layout . . . . .   | 74        |
| 5.2.2    | Small Layout . . . . .  | 76        |
| 5.2.3    | Medium Layout . . . . .                                       | 78        |
| 5.2.4    | Large Layout . . . . .  | 79        |
| 5.3      | Flying-V . . . . .  | 81        |
| 5.3.1    | Tiny Layout . . . . .   | 81        |
| 5.3.2    | Small Layout . . . . .  | 83        |
| 5.3.3    | Medium Layout . . . . .                                       | 84        |
| 5.3.4    | Large Layout . . . . .  | 86        |
| <b>6</b> | <b>Conclusion and Future Directions</b>                       | <b>88</b> |
|          | Referências Bibliográficas . . . . .                          | 99        |

---

|   |            |
|---|------------|
| <b>A The WarehousePy Simulator</b>                | <b>100</b> |
| A.1 Discussion and Final Considerations . . . . . | 107        |

# Lista de Símbolos

**AGVs** : *Automatic Guided Vehicles*

**AMRs** : *Autonomous Mobile Robots*

**CBS** : *Conflict-Based Search*

**MAPF** : *Multi-Agent Path Finding*

**RFMS** : *Robotic Mobile Fulfillment Systems*

**ST-SPF** : *Space-Time Swarm Path Finding*

**STMS** : *Space-Time Multi-Start*

# List of Figures

|      |  |    |
|------|--|----|
| 2.1  | The Piano-Mover Problem . . . . .                                    | 8  |
| 2.2  | Different Map Representations for Robot Localization . . . . .       | 10 |
| 2.3  | Common Graph Construction Methods . . . . .                          | 13 |
| 2.4  | Types of Graph Representations . . . . .                             | 15 |
| 2.5  | Node Search Example . . . . .  | 16 |
| 2.6  | A-Star Algorithm . . . . .   | 21 |
| 2.7  | Common Classical MAPF Conflicts . . . . .                            | 26 |
| 2.8  | An Typical Warehouse Flow and Operation . . . . .                    | 29 |
| 2.9  | <i>Pick-and-Pass</i> and <i>Pick-and-Merge</i> Methods . . . . .     | 30 |
| 2.10 | Industrial Warehouse Typologies . . . . .                            | 31 |
| 2.11 | Traditional Warehouse Layout Design . . . . .                        | 33 |
| 2.12 | Irregular Warehouse Layouts Design . . . . .                         | 34 |
| 2.13 | Automated and Robotized Warehouse Robots . . . . .                   | 35 |
| 2.14 | An Generic RMF System . . . . .                                      | 36 |
| 3.1  | Where the Cooperative A* Fails . . . . .                             | 41 |
| 3.2  | Path Planning Methodologies generally used before the MAPF . . . . . | 44 |
| 3.3  | The Reservation Table . . . . .                                      | 47 |
| 4.1  | Path Planning Results for the Traditional Layout . . . . .           | 52 |
| 4.2  | Path Planning Results for the Traditional Vertical Layout . . . . .  | 53 |
| 4.3  | Path Planning Results for the Flying-V Layout . . . . .              | 54 |
| 4.4  | The Swarm Zone Creation for Isolated Agents . . . . .                | 56 |
| 4.5  | The Swarm Zone Appending . . . . .                                   | 56 |
| 4.6  | Avoiding Blind Spots and Conflicts . . . . .                         | 57 |



---

|      |   |     |
|------|---|-----|
| 4.7  | The ST-SPF Algorithm . . . . .                                    | 58  |
| 4.8  | The STMS Algorithm . . . . .                                      | 61  |
| 4.9  | Space-Time Multi-Start Algorithm . . . . .                        | 62  |
| 5.1  | Tiny Traditional Horizontal Graph . . . . .                       | 67  |
| 5.2  | Small Traditional Horizontal Graph . . . . .                      | 69  |
| 5.3  | Medium Traditional Horizontal Graph . . . . .                     | 71  |
| 5.4  | Large Traditional Horizontal Graph . . . . .                      | 73  |
| 5.5  | Tiny Traditional Vertical Graph . . . . .                         | 75  |
| 5.6  | Small Traditional Vertical Graph . . . . .                        | 77  |
| 5.7  | Medium Traditional Vertical Graph . . . . .                       | 79  |
| 5.8  | Large Traditional Vertical Graph . . . . .                        | 80  |
| 5.9  | Tiny Flying-V Graph . . . . .                                     | 82  |
| 5.10 | Small Flying-V Graph . . . . .                                    | 84  |
| 5.11 | Medium Flying-V Graph . . . . .                                   | 85  |
| 5.12 | Large Flying-V Graph . . . . .                                    | 87  |
| A.1  | The asprilo Simulator . . . . .                                   | 101 |
| A.2  | The WarehousePy Algorithm Flowchart . . . . .                     | 103 |
| A.3  | Warehouse Layouts implemented in WarehousePy . . . . .            | 104 |
| A.4  | The Tiny Traditional Horizontal Grid Environment . . . . .        | 105 |
| A.5  | The Tiny Traditional Horizontal Environment Connections . . . . . | 105 |
| A.6  | Examples of Obstacle Removal after the Initialization . . . . .   | 106 |
| A.7  | Tiny Traditional Horizontal Graph & Grid Environments . . . . .   | 107 |

# List of Tables

|      |   |    |
|------|---|----|
| 2.1  | Most Common Map Decomposition Techniques . . . . .                  | 11 |
| 2.2  | Advantages and Disadvantages of Common Graph Construction Methods . | 12 |
| 2.3  | Taxonomies of <i>EDCs</i> . . . . .                                 | 32 |
| 2.4  | Main Differences Between AMRs, AGVs and AGCs . . . . .              | 37 |
| 3.1  | Related Researches Comparison . . . . .                             | 48 |
| 5.1  | Tiny Traditional Horizontal Results . . . . .                       | 67 |
| 5.2  | Small Traditional Horizontal Results . . . . .                      | 69 |
| 5.3  | Medium Traditional Horizontal Results . . . . .                     | 70 |
| 5.4  | Large Traditional Horizontal Results . . . . .                      | 72 |
| 5.5  | Tiny Traditional Vertical Results . . . . .                         | 74 |
| 5.6  | Small Traditional Vertical Results . . . . .                        | 76 |
| 5.7  | Medium Traditional Vertical Results . . . . .                       | 78 |
| 5.8  | Large Traditional Vertical Results . . . . .                        | 80 |
| 5.9  | Tiny Flying-V Results . . . . .                                     | 81 |
| 5.10 | Small Flying-V Results . . . . .                                    | 83 |
| 5.11 | Medium Flying-V Results . . . . .                                   | 85 |
| 5.12 | Large Flying-V Results . . . . .                                    | 86 |

# Chapter 1

## Introduction

In this first chapter, a brief introduction to the research theme will be elaborated, also addressing the motivations, objectives, and contributions in the author's view that led to the development of this research. At the end of the chapter, a brief summary of the dissertation structure will also be made.

### 1.1 Motivation

Recent advances in the field of robotics had allowed several improvements in diverse fields of study. As a result of this new trend, many applications can acquired an better productivity, efficiency, robustness, and flexibility. While some applications require only one robot to perform tasks, others require multiple robots to perform certain actions. Thus, a **Multi-Robot Systems (MRS)** is a system where two or more robots perform a task using a cooperative approach.

Despite the new progresses in localization, mapping and autonomous exploration, we can still find new problems that need solutions or improvements. Some that can be cited is the ability to perform complex tasks similar to human execution, reliability on the robot's movement in the real world, and the improvement of information analysis in a more rational way [74]. Some improvements and current challenges are also mentioned in [32], where the objectives can be better coordination in homogeneous or heterogeneous systems, the physical identity of agents, and adaptations of artificial intelligence techniques to solve control problems.

The field of robotics is vast, but one branch that has been gaining attention is the **Robotic Mobile Fulfillment Center (RMFS)**. Although the industrial area already has a good implementation and several studies that analyze the use of Automated Guided-Vehicles (AGV), the RMFS (which is also based on the same concept as AGVs) is focused on the use of an applied Multi-Agent System Fulfillment Centers and Warehouses. The great advantage of this new technology concerning common AGVs is the ability to make decisions in real-time with a certain level of autonomy and more "intelligent" trajectory planning, based on computer vision or based on more powerful sensors such as LIDAR.

Although the first large-scale implementation of this system was only in 2008 [25], this new implementation brought a revolution to the Warehouses sector especially in 2012 after the purchase of Kiva Robotics by Amazon Robotics intending to improve the efficiency of their Fulfillment Centers and Warehouses autonomously. The first system implemented was characterized by a path planning based on the traditional A\* Algorithm, a localization system made by reading a *QR Code* on the floor, and infrared sensors and basic communication to avoid collisions. Over the years, this new research niche has allowed the union of an artificial intelligence branch that has been studied since 2006 [62], called **Multi-Agent Path Finding**.

The MAPF was initially proposed in 2006 [46] as a way to solve the path planning problem with multiple agents in Games. However, it started to take shape for the field of Robotics, Artificial Intelligence, and Transport Systems over the years. Most of the proposed algorithms are based on A\*, these being characterized by the non-collision key constraint while the agents perform the path concurrently. As can be imagined, the RMFS has benefited from the new implementations of this problem. Since it is also A\*-based, it has to deal with a large complexity and processing load in its planner, since the planning of multiple robots must be done in a way that there are no collisions.

In recent years, several state-of-the-art algorithms and optimizations have been proposed, such as STA\* (also known as CA\*) [59], RRA\* [60], WHCA\* [46], CBS [54]. However, despite the advances and new implementations, MAPF is still considered NP-hard, and even if some solutions are optimal, complete, or efficient, these variables directly depend on factors such as grid size, number of agents, bottlenecks, free space available, among others. It is correct to state then that the best solution for MAPF is unknown [62], and some questions still need to be tackled, such as:

- How to distribute the MAPF without losing efficiency and completeness?
- How to model MAPF considering Kinematic Constraints?
- How to plan for each agent separately maintaining soundness, completeness, and optimality?
- How to create complete and efficient MAPF algorithms but that are also optimal and non-complex?
- Which low-level solver to use?
- When to merge the agents?
- How reduce conflicts?
- When to use which algorithm?
- How to scale up to one thousand robots, yet achieve good throughput?
- How to do "lifelong" planning for Automated Warehouses?
- How to handle heterogeneous robots?

As can be seen, there are several questions that still need to be answered. In this specific work, we will focus on *how we could distribute this problem, what kind of low-level algorithm to use, how to reduce conflicts, and how to make the system more scalable.*

## 1.2 Objectives

This dissertation has the main objective of implementing two new algorithms applied to RMFS with a focus on Path Planning based on the latest state-of-the-art developments. The secondary objectives of this dissertation are:

- Develop a simulator that allows to manipulate the desired environment (map) so that different trajectory algorithms and desired layouts for single and multi-agent analysis;
- Implementation of two new algorithms that allows better scalability when compared to state-of-the-art algorithms;

## 1.3 Contributions

The main contributions of this work are:

- An algorithm based on the concept of swarm called Space-Time Swarm Path Finding (ST-SPF), that brought scalability as the number of agents increases, reaching up to 48% reduction in Running Time when compared to Conflict-Based Search (CBS). The ST-SPF also allows the modification of its internal algorithms, whether high or low level, thus allowing future tests with new state-of-the-art algorithms;
- An algorithm called Space-Time Multi-Start (STMS), capable of reducing Makespan in small and populous spaces by up to 20%, which also brought the possibility of solutions not found by other algorithms due to this. The STMS also allows the modification of its internal algorithms, whether high or low level, thus allowing future tests with new state-of-the-art algorithms;
- A simulator for pathfinding for single or multi-agent analysis, where the user can define the map and other settings in a Grid-Like or Graph-Like environment that will be made available to the community Open Source, tested with up to 250 agents;
- Efficiency tests using different state-of-the-art warehouse environments of different sizes, such as Traditional Vertical, Traditional Horizontal, and Flying-V;i
- Efficiency tests with low-level algorithms such as Dijkstra's, A\*, Breadth-First Search, to identify which performs better in different types of warehouse environments and which one should be used for the MAPF.

## 1.4 Dissertation Structure

In Chapter 2, a brief theoretical review of the subjects will be made to assist the understanding of the solutions proposed in the dissertation, covering topics of Robotics, Path Planning, and a review about Warehouse Systems and Robotic Mobile Fulfillment Systems. In Chapter 3, the latest research trends in the related fields of this work will be presented, where a brief discussion on the themes is also elaborated, presenting the author's point of view and the

---

possible research gaps. In Chapter 4, the developed algorithms will be presented, addressing the reason of choice for the A\* as the base low-level algorithm, the ST-SPF algorithm, and the STMS algorithm. In Chapter 5, the experimental evaluation is presented for 3 types of warehouse layouts with 4 different sizes from 1 to 250 robots. Finally, in Chapter 6 we discuss the conclusions and future directions.

# Chapter 2

## Theoretical Concepts and Foundations

This chapter aims to present the main concepts and theories used in this thesis, serving as a knowledge base for the later chapters.

### 2.1 Path Planning for Autonomous Mobile Robots

According to [18], the path planning of mobile robots involves different tasks, and the most classic approach is defined as the creation of a path on a respective map, which guides the robot movement from its initial position to the desired position, without collision with obstacles on the way. Although the solution is relatively simple in its definition, path planning can be really difficult at a computational level, since considerations such as mechanical problems, limitations of sensors in the robot, and uncertainties found in real scenarios can complicate the planner development [56].

In the field of artificial intelligence, planning and reaction are usually actions with different or opposite approaches, but applied to autonomous mobile robots, planning and reaction can be considered complementary to each other [58]. To prove this statement, let's consider a robot  $A$  at time  $t_i$  and located in a map  $M : \mathbb{R}^2$ , that needs to leave its current location  $P : \{X_M, Y_M\}$  and go to a certain goal  $G : \{X_M, Y_M\}$ , it is clear that to execute the movement, an action or planning will be necessary. But what happens if he encounters an obstacle along the way or the environment changes? In respect of an autonomous robot, it must react to the unexpected event (the obstacle) so that it can still reach the desired position, that is, without planning or reaction the robot will never reach the goal. In the best cases the reac-



tion will modulate the robot's behavior locally, however, we can approach a strategy to its maximum possible limit, where the planner will react to the new information in real time, achieving the concept of integrated planning and execution [58].

Before moving on, an extremely important concept that needs to be highlighted is the relationship between the system's ability to find the desired position regardless of whether a solution exists, called **completeness** [58]. In summary, a robotic system can be called as *complete* if for all defined problems such as initial states, maps and goals, the robot will find the expected solution. As can be seen, achieving this directly affects the computational complexity, where a sacrifice of completeness may be needed in exchange for performance.

### 2.1.1 Path Planning Schemes and Approaches

The way we develop an path planning solution will directly depends on the problem [58]. Industrial robots on a factory line for example, may need to be developed to achieve the maximum movement speed as possible, since this is directly linked to an economic return. In this case, the dynamics and kinematics of the movements are significant factors to achieve a satisfactory planning. However, if the speed is relatively low, as the mobile robots found in Warehouses for example, where high speeds can possible cause accidents due to continuous Human–robot interaction, the dynamics will rarely be considered in the path planning, simplifying the problem.

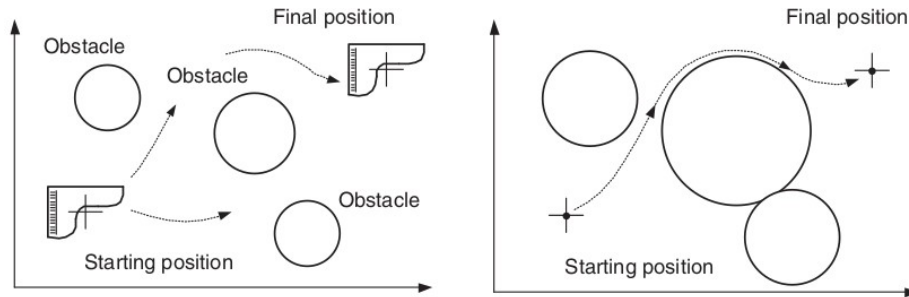
In the next subtopics, some basic concepts will be addressed, since they are fundamental for understanding how a Path Planning algorithm works. After that, some of the most common algorithms used in AMR and MRS systems will be covered.

#### Geometric Path Planning Problem

For mobile robots, the robot is generally considered only as a *point* [58], in order to facilitate the spatial configuration of the path planning in an almost identical 2D representation of the physical space, **differing only in the size of the obstacles on the map, which must be increased at least to the same radius as the robot** [18]. The reason for this approach can be found in the *piano-mover problem*, where an object with a complex shape needs to be transferred from a starting position  $P : \{X_M, Y_M\}$  to a goal  $G : \{X_M, Y_M\}$  around a

physical space with obstacles as seen in Figure 2.1, adapted from [18]. In order to find the best path, the piano is reduced to a point and the obstacles are increased in the same proportion, allowing to find a possible path between the starting and ending position.

Figure 2.1: The Piano-Mover Problem



By definition, the *piano-mover problem* [11] demonstrates how path planning can be done in an ideal environment without external interference. However, in conditions where the obstacles are unknown, or the obstacle moves randomly during the path execution, the planning must be done in a way that this change becomes noticeable to the robot (online) through sensors, cameras, and a controller with enough memory that allows calculating the distance between the robot and the object over time. In addition to these topics, there are also other prominent areas of research, such as the correct local and global robot pose in the physical space, the mapping and sensing of the surroundings, and the collective behavior [18].

### The Configuration Space and Workspace

The path planning of a robot is formally defined by a mathematical representation called *configuration space* (C-Space) [58]. As previously mentioned, the robot pose can be represented at different locations and orientations in the plane when it moves, and the configuration space allows us to find mathematically the minimum of order parameters that will define the robot's posture [18]. Thus, for a planar moving object, as discussed in the Kinematics topic of a mobile robot, only three parameters are needed: the 2D coordinates of the robot's center of gravity and its orientation angle. Regarding the *Workspace*, it can be defined as the geometric set of points that the robot can reach, that is, for a mobile robot it is the two-dimensional space in which it can operate.

An more formal definition can be found at [56], defining the Workspace  $W = \mathbb{R}^N$ , where  $N = 2 \vee 3$ , as a static environment populated with *Obstacles*  $\subset W$ . To find the collision-free path planning for an robot  $R$ , the configuration  $q$ , it is, every point on the robot geometry must be provided. The C-space, where  $q \in C$ , is all the possible *configurations* and transformations applied to a robot given its kinematics. Considering the closed set  $R(q) \subset W$  as the set of points occupied by the robot when  $q \in C$ , the C-space obstacle region will be:

$$C_{obstacle} = \{q \in C | R(q) \cap Obstacles \neq \emptyset\} \quad (2.1)$$

And the *free space* will be:

$$C_{free} = C \setminus C_{obs} \quad (2.2)$$

Finally, it is worth mentioning that  $C_{free}$  is usually coupled with a cost function [18], allowing that the path planning performance is measured, quantifying the error between predicted values and expected values, and presenting in the form of a single real number . Generally the most used functions for this work are the Mean Absolute Error (Equation 2.3), and the Mean Squared Error (Equation 2.4).

$$MAE = \frac{1}{m} \sum_{i=1}^m |\hat{y}^{(i)} - y^{(i)}| \quad (2.3)$$

$$MSE = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2 \quad (2.4)$$

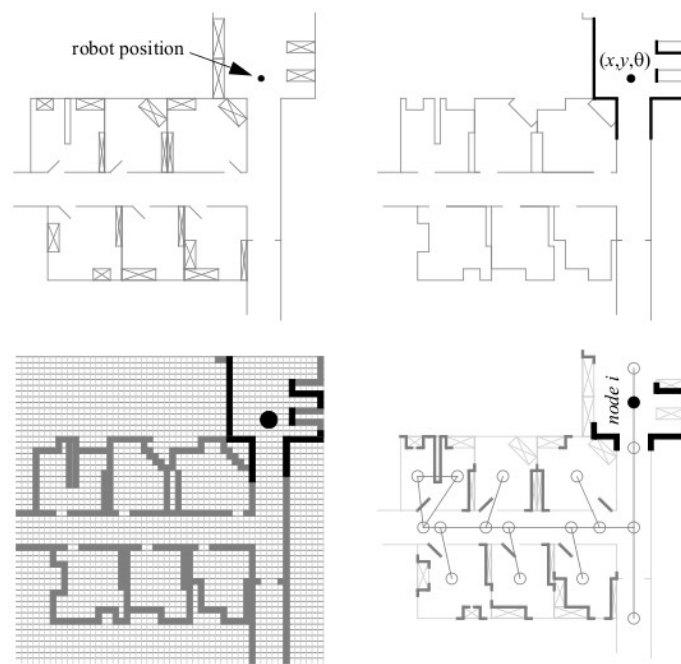
Where:

- $i$  its the sample index;
- $\hat{y}$  is the predicted value;
- $y$  is the expected value;
- $m$  is the number of samples in dataset.

## Map Representations

Before starting to study the most used schemes and approaches, it is necessary to understand how the environment representation works in conjunction with the path planning algorithm. Generally, this representation will vary from a continuous geometric description to a decomposition-based geometric map or topological map, in Figure 2.2 it is possible to observe different hypotheses of the robot's position using different map representations [58].

Figure 2.2: Different Map Representations for Robot Localization



Considering Figure 2.2, it is possible to observe that the robot movement will depend directly on how the environment (map) is represented since it provides the actual and future robot's real position. To obtain a better fidelity of this position, some relationships must be analyzed before choosing a specific map representation [58]:

- The map must be similar to the real environment in a precision sufficient for the robot to reach its objective;
- The accuracy of the map and its peculiarities must be represented in a way that the data read by the robot's sensors in the real environment is similar;

- The complexity of the map representation has a direct impact on the computational complexity of tasks such as location, navigation, and mapping.

Considering these relationships, several decomposition strategies of a continuous map (an exact decomposition of the environment) can be carried out in a way that the fidelity of the real and abstracted map is not lost. Some of the most common used decomposition methods are: *exact cell decomposition*, *fixed decomposition*, and *occupancy grid* [58]. In Table 2.1 it is possible to analyze some of the advantages and disadvantages of each.

Table 2.1: Most Common Map Decomposition Techniques

|                                 | Advantages  | Disadvantages  |
|---------------------------------|---|--|
| <i>Exact Cell Decomposition</i> | <ul style="list-style-type: none"> <li>- Extremely Compact</li> <li>- Effective to capture the Environment Connectivity</li> </ul>                          | <ul style="list-style-type: none"> <li>- If the information of the obstacles and free space is expensive or unknown becomes unfeasible</li> </ul>  |
| <i>Fixed Decomposition</i>      | <ul style="list-style-type: none"> <li>- Extremely Popular in mobile robotics;</li> <li>- Easy to implement</li> </ul>                                      | <ul style="list-style-type: none"> <li>- Inexact Nature since some narrow passages can be lost during transformation;</li> <li>- Not compatible with closed-world assumption;</li> </ul> |
| <i>Occupancy Grid</i>           | <ul style="list-style-type: none"> <li>- Solves the inexact nature found on the Fixed Decomposition;</li> <li>- Can provide high resolution maps</li> </ul> | <ul style="list-style-type: none"> <li>- The Size of the map in robot memory grows with the size of the environment;</li> <li>- The cell size can become untenable.</li> </ul>           |

As discussed in the previous subtopics, the environment representation in which the robot will perform his movement can be achieved through a geometric description, a geometric decomposition, or a topological map. Taking this into account, the first step to implement a path-planning system is the real environment conversion into a discrete map, with needs to be feasible to the path planning algorithm. According to [58], the planning algorithms will differ in how they use this discrete decomposition, which can be:

- **An Graph Search:** An connectivity graph is constructed based on the free space and searched, generally performed offline;
- **An Potential Field:** An mathematical function is imposed on the free space and the function's gradient is followed to the goal.

Although the Potential Field Planning can be found in several studies in the past years, due to the characteristic of the research proposed using Metaheuristics and Swarm Intelligence which focuses on a *Graph Search* approach, his explanation will not be addressed.

According to [58], the **Graph Search** technique is firmly linked to the field of mathematics, however, it has received a lot of attention in recent years in the field of mobile robotics.

Most of the methods that use this approach are based on two steps: the construction of the graph, where nodes are connected via edges, and the graph search, where the computation of an optimal solution is made. For building a set of nodes and edges on the map we first need to find some solutions that will construct the graph in a way that the path planning understands what paths are available. Generally, the most common methodologies to solve this problem are through a *Visibility graph*, *Voronoi Diagram*, *Exact Cell decomposition*, *Approximate Cell Decomposition*, or the *Lattice Graph*. The advantages and disadvantages of these methodologies can be seen at Table 2.2.

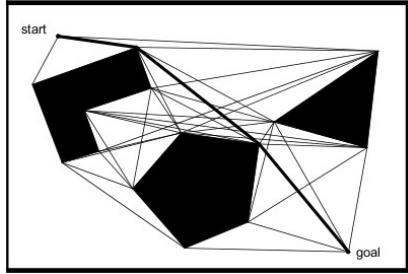
Table 2.2: Advantages and Disadvantages of Common Graph Construction Methods

|                                      | Advantages   | Disadvantages  |
|--------------------------------------|--|--|
| <i>Visibility Graph</i>              | <ul style="list-style-type: none"> <li>- Simple to Implement</li> <li>- Fast and efficient in sparse environments</li> <li>- The shortest solution is optimal in terms of path length.</li> <li>- More complete than the <i>Visibility Graph</i>;</li> </ul> | <ul style="list-style-type: none"> <li>- The size and number of edges and nodes increase with the number of obstacles;</li> <li>- The planner solution can take the robot close as possible to the obstacles as he moves to the goal;</li> <li>- The optimal-length results may be sacrificed in order to improve the disadvantage above.</li> </ul> |
| <i>Voronoi Diagram</i>               | <ul style="list-style-type: none"> <li>- Maximize the distance between the robot and obstacles;</li> <li>- Simple control rules can be implemented due to his unique (executability) characteristic.</li> </ul>  | <ul style="list-style-type: none"> <li>- Usually far from optimal in terms of path length;</li> <li>- Limited range sensor localization, causing the danger of don't sense the surroundings in short-range;</li> </ul>   |
| <i>Exact Cell Decomposition</i>      | <ul style="list-style-type: none"> <li>- The particular robot position within each cell (free space) does not matter;</li> <li>- In large or sparse environments the representation is efficient and lossless.</li> </ul>                                    | <ul style="list-style-type: none"> <li>- The computation planning efficiency depends on the objects size, density, and geometric complexity;</li> <li>- The exact cell decomposition is rarely used in mobile robots due to its complexities.</li> </ul>   |
| <i>Fixed-size Cell Decomposition</i> | <ul style="list-style-type: none"> <li>- Starts with a coarse resolution until the limit resolution is attained;</li> <li>- Low computational complexity induced to path planning.</li> </ul>  | <ul style="list-style-type: none"> <li>- Narrow Passages can be lost due to tessellation when a high map resolution is needed;</li> </ul>  |
| <i>Lattice Graph</i>                 | <ul style="list-style-type: none"> <li>- Freedom to design feasible edges.</li> </ul>  | <ul style="list-style-type: none"> <li>- Typically precomputed for a given platform and stored in memory.</li> </ul>   |

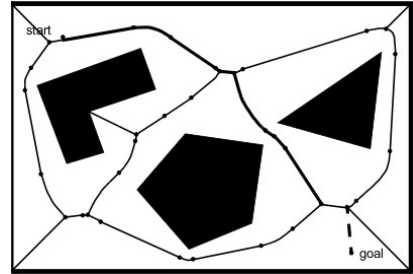
In Figure 2.3 (adapted from [58]) is possible to see the most common graph construction methods. The **Visibility Graph** (Figure 2.3a) consists of joining all the pairs of vertices that can see each other, including the starting and ending points, then the planner is responsible for finding the shortest path from the point initial to final. The **Voronoi Diagram** (Figure 2.3b) is a complete road map focused on maximizing the distance between robots and obstacles, where for each free-space point the distance to the next obstacle is computed, increasing the characteristic of moving away from the obstacle, and becoming extremely viable for automatic environment mapping. The **Exact Cell decomposition** (Figure 2.3c) is based on the geometric criticality of every cell boundary, making each resulting cell free or occupied. The **Fixed-size Cell Decomposition** (Figure 2.3d) is the most used in mobile robotics due to the grid representation popularity, and works similar to the previous method but using fixed grid-size cell decomposition's, where the cell size (*generally a 5 x 5 cm cell size*) is not dependent on the size of the objects in the environment. Finally, the **Lattice Graph** (Figure 2.3e) has been adapted in recent years for the graph search, and its graph formation is done through the construction of a base set of repeated edges over the whole configuration space

to form a graph.

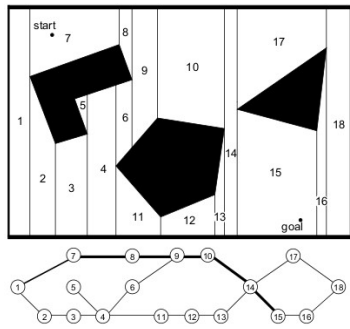
Figure 2.3: Common Graph Construction Methods



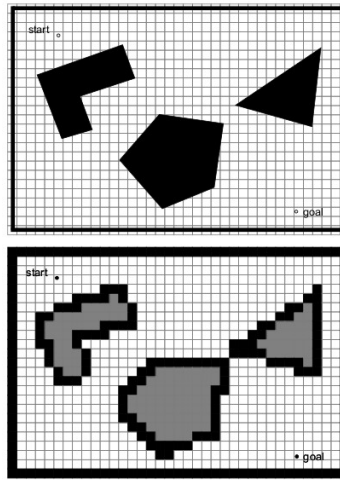
(a) Visibility Graph Method



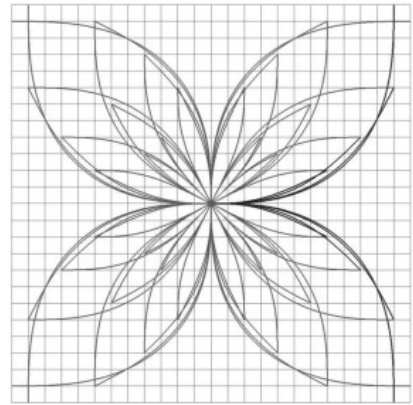
(b) Voronoy Diagram Method



(c) Exact Cell decomposition Method



(d) Fixed-size Decomposition Method



(e) Lattice Graph Method

Considering the explanations above, of what methods are used to construct a graph usable to the path planning algorithm, as the basic operational concepts of each one, we can enter now into the topic of how some algorithms elaborate the search to the most optimal path. In summary, considering the way that the graph construction method was elaborated, the algorithm will look for the best path in the connection map between the start point and the goal in a way that the chosen path is the best possible. In general, most deterministic graph search algorithms are relatively similar to each other but have some differences in the result of total cost  $f(n)$ , path cost  $g(n)$ , cross edge cost  $c(n, n')$  (cost of node  $n$  to adjacent node  $n'$ ), and heuristic cost  $h(n)$  (cost of node  $n$  to goal node), being these all functions of node  $n$  and the adjacent node  $n'$  [58]. Thus, the total cost of a deterministic path planning algorithm

from start to goal, considering that  $\epsilon$  will be the parameter that assumes the values dependent on the algorithm, can be described according to Equation 2.5:

$$f(n) = g(n) + \epsilon \cdot h(n) \quad (2.5)$$

It is worth mentioning that the value of  $\epsilon$  has a direct relationship with a faster convergence rate, or a more optimal algorithm, or a suboptimal algorithm and its value must be analyzed according to each algorithm and situation.

## Search Algorithms

Considering the behavior of acting rationally in an artificial intelligence system and the concept of agent defined earlier, we can now talk about a branch of Artificial Intelligence study that involves problem-solving based on an agent search. Let's consider that we have an agent in the city of João Pessoa enjoying his vacation and that wants to visit several places before leaving. For him to start his journey it is necessary to define a *goal*, that is, which place he should visit first. This will help to better organize his behavior and limit the actions that our agent wants to accomplish. For him to achieve this goal, we must follow a set of *states* that will be responsible for taking our agent to the desired goal, such as turning left on a certain street. As it may be possible to imagine, in the middle of the journey we may encounter several difficulties due to the "unknown", such as the decision of which street to take at a bifurcation, but which can be resolved if the agent has a map of the city. Considering this example, we can define some important concepts in the field of artificial intelligence.

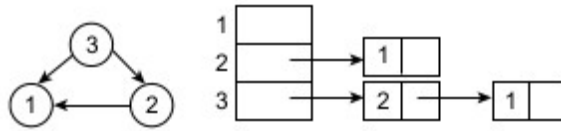
An environment is **unknown** if we have to make some random decisions like choosing the street without a map, but it becomes **observable** if we have a map in our hands. We can tell if the environment is **discrete** if we consider the execution of an action by states, and **deterministic** if each action has exactly one outcome. Considering this, *we can assume that the solution to any problem is a fixed sequence of actions, whether random or known in an observable environment or not.* Thus, a **search** is nothing more than this sequence of actions, which has defined inputs and returns a solution in the form of a sequence of actions. According to [61] a search is usually done through a directed network or **graph**, where nodes are states and links between nodes are actions. The **path** will then be a sequence of states connected by a sequence of actions, with your **cost** being a function that adds a numerical



cost to each path. With this cost, we can find not only a solution but a **optimal solution**, characterized by the path with the lowest cost among the others.

Before we continue the discussion, it is important to review the concept of Graphs and Heuristics. According to [61], a graph  $G = (V, E)$  is generally represented in two ways, as an adjacency matrix (Figure 2.4 (left), adapted from [61]), characterized by a two-dimensional Boolean array  $M$ , where  $M_{i,j}, 1 \leq i, j \leq n$  is true iff an edge contains a node with index  $i$  as a source and a node index  $j$  as a target. For graphs with few edges (*sparse graphs*) a *adjacency list* is more appropriate (Figure 2.4 (right), adapted from [61]), where its implementation is a  $L$  array of pointers to node list, where for each node  $u \in V$ , the entry  $L_u$  will have a pointer to the list of all nodes  $v$  with  $(u, v) \in E$ .

Figure 2.4: Types of Graph Representations

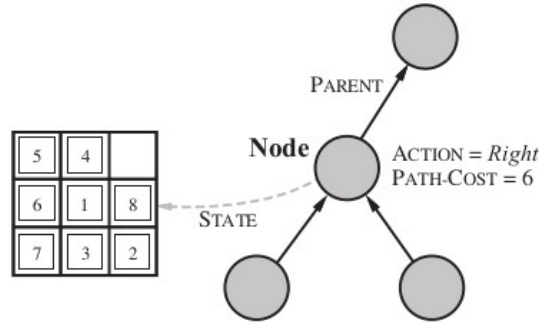


Another important point for a search is the *heuristic*. A heuristic is commonly used to estimate the remaining distance of a node to the goal, being used by search algorithms to determine which status or direction is more viable than the other. Thus, for a more formal presentation, we can define a heuristic as [61]:

**Definition 2.1.1.** (Heuristic) A heuristic  $h$  is a node evaluation function, mapping  $V$  to  $\mathbf{R}_{\geq 0}$ . If  $h(t) = 0$  for all  $t$  in  $T$  and if all other nodes  $u \in V$  then  $h(u) \neq 0$ . Thus, considering  $g(u)$  the path cost to a node  $u$  and  $h(u)$  its estimate:

$$f(u) = g(u) + h(u) \quad (2.6)$$

Figure 2.5: Node Search Example



Although there are currently several heuristics for Grid-like environments, such as *Diagonal distance* and *Euclidean distance*, one of the most used today is **Manhattan Distance** (Taxicab Metric, Manhattan Length, e.g.) which can be defined as:

**Definition 2.1.2.** (Manhattan Distance) The Manhattan Distance is the distance between two points measured along axes at right angles. In other words, in a plane defined as a  $n$ -dimensional real vector space with fixed Cartesian coordinate system, with vectors  $p$  at  $(x_1, y_1)$  and  $q$  at  $(x_2, y_2)$ , the Manhattan Distance is the sum of lengths for the projections in the line segment between the points onto the coordinate axes, it is  $(|x_1 - x_2| + |y_1 - y_2|) * D$ , where  $D$  is the cost. Formally we have:

$$d_1(p, q) = ||p - q||_1 = \sum_{i=1}^n D|p_i - q_i| \quad (2.7)$$

Before we enter in the topic of search algorithms that will be used in this thesis, it is also important to understand the search algorithms basic infrastructure and the methodologies for measuring problem-solving performance. The search algorithms needs a data structure to keep the search tree active, to achieve this for each  $n$  node in a tree the structure must have four components (see Figure 2.5 [51]):

- node.STATE: the state in the space to which the node corresponds;
- node.PARENT: the node in the search tree that generated this node;
- node.ACTION: the action that was applied to the parent to generate the node;
- node.PATH-COST: the cost, traditionally denoted by  $g(n)$  of the path from the initial state to the node;

Finally, to evaluate the performance of the algorithms, the following methodologies can be applied:

- **Completeness:** Characterized by the guarantee that a solution will be found if it exists;
- **Optimality:** Characterized by the strategy to find the optimal solution;
- **Running Time:** Characterized by the time spend to find the solution;
- **Space Complexity:** Characterized by the memory needed to perform the search;
- **Depth:** Characterized by the number of steps along the path from the root.

## 2.1.2 Common Path Planning Algorithms

### Basic Algorithm

Analyzing the piano-mover problem previously addressed, and mathematical explanation of the Configuration Space, and the Workspace, the above formal definition can be made [56]:

- Considering:
  - A *workspace*  $W$ , where  $W = \mathbb{R}^2$ ;
  - An *obstacle region* or the collection of obstacles where  $Obstacle \subset W$ ;
  - A *robot* defined in  $W$ , being that a rigid body  $R$  or the collection of  $x$  links:  $R_1, R_2, \dots, R_x$ ;
  - The *configuration space*  $C$  with the C-space obstacle region  $C_{obs}$ , and the free space  $C_{free}$  defined;
  - An *initial configuration*  $q_1 \in C_{free}$ ;
  - A *goal configuration*  $q_G \in C_{free}$ ;
  - An *query*  $(q_1, q_G) \in C_{free}$ .
- The continuous path can be computed for  $\tau : [0, 1] \rightarrow C_{free}$  such that  $\tau(0) = q_1$  and  $\tau(1) = q_G$ .

**Breadth-First Search Algorithm****Algorithm 1:** Breadth-First Search Pseudo Code

---

```

node  $\leftarrow$  a node with STATE = problem.INITIAL_STATE, node.PATH_COST = 0
if problem.GOAL_TEST(node.STATE) then return SOLUTION(node)
frontier  $\leftarrow$  a FIFO queue with node as the only element
explored  $\leftarrow$  an empty set
RUN_SEARCH = True
while RUN_SEARCH do
    if EMPTY(frontier) then
        | return failure
    end
    node  $\leftarrow$  POP(frontier)
    add node.STATE to explored nodes
    for each action in problem.ACTIONS(node.STATE) do
        node  $\leftarrow$  CHILD_NODE(problem, node, action)
        if child.STATE not in explored or frontier then
            if problem.GOAL_TEST(child.STATE) then
                | return SOLUTION(child) node  $\leftarrow$  INSERT(child, frontier)
                | RUN_SEARCH = False
            end
        end
    end
end
end

```

---

According to [51], the **Breadth-First Search** (see the Pseudo Code 1 adapted from the same author) is one of the simplest strategies that can be adopted, characterized by the initial expansion of the root node, followed by the successor nodes expansion through a FIFO queue. This algorithm is considered complete since the breadth-first search will eventually find the goal node in some finite depth  $d$ . He can be also considered optimal, but only if the path cost is a non-decreasing function of the depth of the node. One of the biggest disadvantages of using this search method is due to the high need for available memory [61], as well as its execution time since the total number of nodes generated in the worst case will

be the value at Equation 2.8.

$$b + b^2 + b^3 + \dots + b^d = O(b^d) \quad (2.8)$$

### Dijkstra's Algorithm

The **Dijkstra's algorithm** is based on the principle of optimality, that is, if an optimal path has the property of initial conditions and choices over some initial period, then the decision variables chosen should be optimal for the rest of the problem. In other words, the minimum distance of a node  $s$  to  $v$  is equal to the minimum distance sum of the predecessor  $u$  of  $v$  plus the edge weight between  $u$  and  $v$ . More formally, we can build the equation 2.9 [61]:

$$\delta(s, v) = \min_{v \in Succ(u)} \{ \delta(s, u) + w(u, v) \} \quad (2.9)$$

In addition, the Theorems 2.1.1 and 2.1.2 [61] are also important for defining and understand the algorithm's exploration scheme.

**Theorem 2.1.1** (Correctness Dijkstra's Algorithm). In weighted graphs with non-negative weight function, the algorithm is optimal, since at the first node  $t \in T$  selected to expansion we have  $f(t) = \delta(s, T)$

**Theorem 2.1.2** (Dijkstra's Algorithm on Infinite Graphs). If the weight function  $w$  of a problem graph  $G = (V, E, w)$  is strictly positive and the weight of every infinite path is infinite, the algorithm terminates with an optimal solution.

According to [18], this algorithm is most used when the workspace is dense with obsta-

cles, and the pseudocode can be described as follows:

---

**Algorithm 2:** Dijkstra Pseudo Code
 

---

```

Create a vertex set  $v_i \in V$ 

Mark  $dist(v_i)$  the distance from  $c_s$  (start point) to  $v_i$ 

Mark  $v_i$ . the parent node of  $v_i$ 

Set  $dist(c_s)$  (start point) = 0

Set  $dist(v_i) = \infty$  for all  $v_i \in V$ 

while  $V$  is not empty do
    Pick  $u$  from  $V$  such that  $dist(u) = \min\{dist(V)\}$ 
    remove  $u$  from  $V$ 
    for each neighbor  $v_i$  of  $u$  do
         $temp = dist(u) + cost(u, v_i)$ 
        if  $temp < dist(v_i)$  then
             $dist(v_i) \leftarrow temp$ 
             $v_i.parent = u$ 
        end
    end
end

return  $dist, parent$ 
  
```

---

As the mathematical formulation of this algorithm has already been studied in Chapter 2.1, this topic will only approach its computational method applied to the path planning problem in robotics. Dijkstra's algorithm is based on a specialized tree-based data structure called *heap*. The elements of this structure are ordered according to the total path cost  $f(n)$  for a given node  $n$ , followed by the expansion of the nodes beginning from the starting point, and a reordering of the nodes according to their value  $f(n)$ . In this way, the cheapest state in the element at the top after reordering (heap) is extracted and expanded until the goal node is expanded and there are no more nodes remaining in the heap, being able them to find the solution by tracing the route from the goal node to the start node. Due to reordering operations, the complexity of the algorithm will be  $O(n \log(n) + m)$ , where  $n$  is the number of nodes and  $m$  the number of edges.

The advantage of using the algorithm for robot path planning is that the best path and all the lowest cost paths for any initial position from the initial node to the end are computed,

allowing the robot to find the best route through its current position (the process is repeated without needing a replanning until the goal) [58].

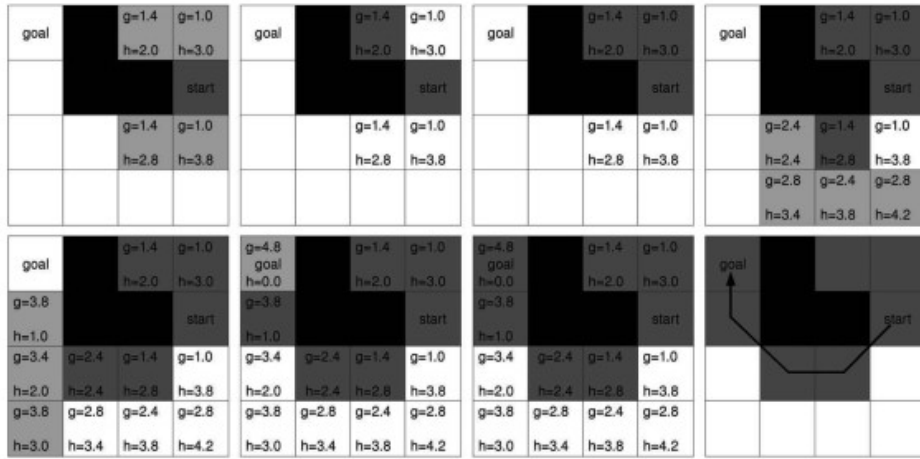
### A\* Algorithm

One of the best known and most studied algorithms for solving search problems is **A-star Search (A \*)**, which based on the best-first search, evaluates the nodes by combining the cost to reach the node and the cost to get from the node to the goal (see Equation 2.10).

$$f(n) = g(n) + h(n) \quad (2.10)$$

The A-Star algorithm ( $A^*$ ) is similar to the Algorithm 2, differing only in the inclusion of the  $h(n)$  function, which makes the graph search efficient for node by node single queries [58]. Generally, the  $A^*$  algorithm is implemented in an *grid* map representation, and the major advantage of this method is the reduction of the expanded nodes numbers needed to find a solution.

Figure 2.6: A-Star Algorithm



Considering an environment (Figure 2.6, adapted from [58]) where there is a discrete two-dimensional space ( $C - Space$ ), a starting point  $c_s$ , an ending point  $c_g$  and obstacles (black cells), the  $A^*$  starts the search by expanding the starting node and placing his neighbors nodes in a heap ordered according to the smallest value of  $f(n)$ , and including the heuristic function  $h(n)$ . The smallest state value is extracted and expanded until the *goal*

node is explored. Generally, it is not necessary to obtain an optimal solution as long the are suboptimal level guarantees [58]. The  $A^*$  pseudocode can be described as follows:

---

**Algorithm 3: A-Star Pseudo Code**


---

**Initialize** an open set  $OL$

**Initialize** a close set  $CL$

**Mark**  $c.g$  and  $c.f, c.parent$  the  $g, f$  values of a cell  $c$  and its parent cell

$OL \leftarrow c_s$

**Set**  $c_s.f$  to zero

**while**  $OL$  is not empty **do**

**Pick**  $q$  to be the node  $q \in OL$  with  $q, f$  minimal

**Drop**  $q$  from  $OL$

**Add**  $q$  to  $CL$

**if**  $q$  is the goal cell **then**

**return**

**end**

**for all** neighbors  $n_i$  or  $q$  check **do**

**if**  $n_i$  is an obstacle cell or is in  $CL$  **then**

            skip to the next neighbor

**end**

$g_{temp}, f_{temp} \leftarrow$  calculate  $g, f$  of  $n_i$

**if**  $n_i.g > g_{temp}$  **then**

            Update  $n_i.g \leftarrow g_{temp}$  and  $n_i.f \leftarrow f_{temp}$

            Set  $n_i.parent \leftarrow q$

**end**

**Add**  $n_i$  to  $OL$

**end**

**end**

**Construct** a path from  $c_g$  back to  $c_s$  by tracking the parent cells

---

Where:

- $OL$  is the set of cells that are currently under inspection;
- $CL$  is the set of cells that are already been inspected;

It is worth mentioning that to achieve smooth movements in the configuration space, it



is necessary to increase the grid density. However, this will cause an increase in the volume of data as the volume of C-Space increases exponentially with the size of the configuration space [18]. It is possible to perceive the similarity with Equation 2.6, that is, A\* uses a  $h(n)$  heuristic function that satisfies certain conditions. A\* is a complete and optimal algorithm, but for optimality to be achieved  $h(n)$  must be an admissible heuristic, that is, it should never overstate the cost to reach the goal. Another important feature for this algorithm to achieve optimality is the **consistency** (monotonicity, e.g.) of the heuristic function, where for each node  $n$  and each successor  $n'$  of  $n$  generated by an action  $a$ , the estimated cost of reaching the goal should not be greater than the step cost of getting to  $n'$  plus the estimated cost of reaching the goal from  $n'$ . This definition is also known as **triangle inequality**[51], or formally:

$$h(n) \geq c(n, a, n') + h(n') \quad (2.11)$$

Considering this, according to [51], the A\* has the following characteristics: *the tree-search is great if  $h(n)$  is permissible, the graph-search is great if  $h(n)$  is consistent*. For [61], the A\* is an elegant caste as the in a reweighted graph, where heuristics are incorporated into the weight function. Putting all these points together, we can perform the Theorems 2.1.3 and 2.1.4 [61]:

**Theorem 2.1.3** (A\* for Consistent Heuristics). Let  $h$  be consistent. If we set  $f(s) = h(s)$  for the initial node  $s$  and update  $f(v)$  with  $f(u) + (u, v)$  instead of  $f(u) + w(u, v)$ , at each time a node  $t \in T$  is selected, we have  $f(t) = \delta(s, t)$ .

**Theorem 2.1.4** (A\* for Admissible Heuristics). For weighted graphs  $F = (V, E, w)$  and admissible heuristics  $h$ , the A\* algorithm is complete and optimal.

### 2.1.3 The Multi-Agent Path Finding (MAPF) Problem

According to [62], the **Multi-Agent Path Finding (MAPF)** problem is a research area that addresses the problems and possible optimizations of path planning for multiple agents, being characterized by the non-collision key constraint while the agents perform the path concurrently.

Also, according to the authors, the classical MAPF problem with  $k$  agents is a tuple  $\langle G, s, t \rangle$  where  $G = (V, E)$  is an undirected graph,  $s : [1, \dots, k] \rightarrow V$  maps an agent to a source vertex, and  $t : [1, \dots, k] \rightarrow V$  maps an agent to a target vertex. For this type of system, time is discretized, and each agent can perform a single function action  $a : V \rightarrow V$  so that  $a(v) = v'$ . That is, if an agent is in a  $v$  vertex and executes a  $a$  action he will be in the  $v'$  vertex in the next time step. By definition, agents can perform two types of actions:

- **Wait:** The agent stays in its current vertex for another time step;
- **Move:** The agent moves from vertex  $v$  to  $v'$ .

Considering these assumptions, we can ask the following question: *How does a sequence of actions in Classical MAPF work?* For a sequence of actions  $\pi = (a_1, \dots, a_n)$  and an agent  $i$  where  $\pi_i[x]$  is the location of the agent after executing the first action  $x$  on  $\pi$  we will have [62]:

$$\pi_i[x] = a_x(a_{x-1}(\dots a_1(s(i)))) \quad (2.12)$$

Where  $s(i)$  is the agent's source, a sequence of actions  $\pi$  is a **single-agent plan** for agent  $i$  iff executing this sequence of actions in  $s(i)$  results  $t(i)$ , and a **solution** is the set of  $k$  single-agent plans for each agent. In this way, if we have a function  $f_{[a,i,j,t]}$ , which defines the agent movement, where the  $a$  is the agent,  $i$  the initial position,  $j$ , the final position, and  $t$  the time, the function  $f_{[a,i,j,t]} = 1$  indicates that agent  $a$  transact to state  $\lambda_{[t-1]}(a) = i$  to  $\lambda_{[t]}(a) = j$ . If  $i = j$  then the agent  $a$  stayed in his vertex position, and if  $t = 0$  we have the initial position of each agent.

### Problem Definition

Considering again  $G = (V, E)$  as an undirected graph, and a set of movable agents  $R = \{\bar{r}_1, \bar{r}_2, \dots, \bar{r}_v\}$  where  $v \leq |V|$ , the initial and goal robots arrangement will be defined by a uniquely invertible functions found at Equation 2.13 and Equation 2.14 respectively.

$$S_R^0 : R \rightarrow V \quad \text{i.e.} \quad S_r^0 \neq S_s^0 \quad \text{for} \quad \forall r, s \in R \quad \text{with} \quad r \neq s \quad (2.13)$$

$$S_R^+ : R \rightarrow V \quad \text{i.e.} \quad S_r^+ \neq S_s^+ \quad \text{for} \quad \forall r, s \in R \quad \text{with} \quad r \neq s \quad (2.14)$$

The multi-agent pathfinding problem is the task to find a number  $\zeta$  and a sequence  $S_R = [S_R^0, S_R^1, \dots, S_R^\zeta]$  where  $S_R^k : R \rightarrow V$  is a uniquely invertible function for every  $k = 1, 2, \dots, \zeta$ . For this happens, the following conditions must hold for the sequence  $S_R$  :

- i All agents reaches their destination vertices:

$$S_R^\zeta = S_R^+ \quad (2.15)$$

- ii A agent can either stay in a vertex or move to the neighboring vertex at each time step:

$$S_R^k(r) = S_R^{k+1}(r) \quad \forall \quad r \in R \wedge k = 1, 2, \dots, \zeta - 1 \quad (2.16)$$

$$\{S_R^k(r), S_R^{k+1}(r)\} \in E \quad \forall \quad r \in R \wedge k = 1, 2, \dots, \zeta - 1 \quad (2.17)$$

- iii The agent  $r$  moves between time steps  $k$  and  $k + 1$ :

$$S_R^k(r) \neq S_R^{k+1}(r) \quad (2.18)$$

- iv No robot  $s$  occupies the target vertex at same time step  $k$ :

$$S_s^k(r) \neq S_R^{k+1}(r) \quad \forall s \in R \quad (2.19)$$

- v If there's no other robot  $s$  on the target vertex at time step  $k$ , then the move of  $r$  at the time step  $k$  is allowed;

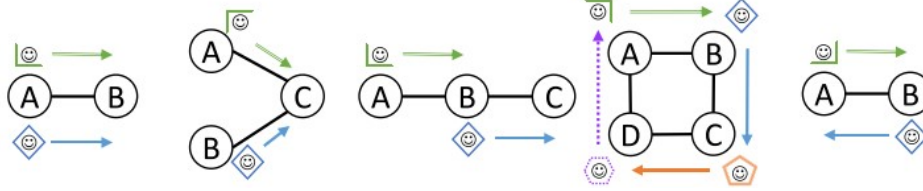
- vi If the robot  $r$  moves into a vertex that is being left by the robot  $s$ , and the move of  $s$  at the time step  $k$  is allowed, then the move of  $r$  at the time step  $k$  is also allowed. That is:

$$S_R^k(r) \neq S_R^{k+1}(r), \quad s \in R \quad (2.20)$$

$$s \neq r \wedge S_R^k(s) = S_R^{k+1}(r) \wedge S_R^k(s) \neq S_R^{k+1}(s) \quad (2.21)$$

## Conflicts, Agent Behavior and Objective Functions

Figure 2.7: Common Classical MAPF Conflicts



The objective of MAPF solvers is to find a solution and a plan for each agent without collisions. To achieve this, the proposed solver uses the concepts of *conflicts* (see Figure 2.7, adapted from [62]), where the solution is only considered valid iff there is no conflict between any set of two single-agent plans. Although the definition of these conflicts depends directly on the environment, some definitions of the most common conflicts for a single-agent pair can be seen below [62]:

**Definition 2.1.3.** (Vertex Conflict) A Vertex Conflict occurs iff the agents are planned to occupy the same vertex at the same time (1st illustration at Figure 2.7).

**Definition 2.1.4.** (Edge Conflict) A Edge Conflict occurs iff the agents are planned to transverse the same edge at the same time step and direction (2st illustration at Figure 2.7).

**Definition 2.1.5.** (Following Conflict) A Following Conflict occurs iff one agent is planned to occupy a vertex that was occupied in the previous time step (3st illustration at Figure 2.7);

**Definition 2.1.6.** (Cycle Conflict) A Cycle Conflict occurs iff the agents performs an rotating cycle pattern at same time step (4st illustration at Figure 2.7);

**Definition 2.1.7.** (Swapping Conflict or Edge Conflict) Occurs iff the agents are planned to swap location in a single time step (5st illustration at Figure 2.7);

Another important definition of a classical MAPF problem is how the agent behaves in after the goal is reached and before it reaches the goal. According to [62], the most common assumptions are:

**Definition 2.1.8.** (Stay at the Goal) The agent will wait in its goal until all agents have reached their goals (will cause a vertex conflict if any plan passes through its goal);

**Definition 2.1.9.** (Disappear at the Goal) The agent immediately disappears when it reaches its goal.

To evaluate a classical MAPF work, benchmarks and a **objective function** are usually used. For the Benchmarks part, the graph type and map representation selected must be analyzed first. As the MAPF application has a wide range of applications, we will focus only on maps based on Warehouse Grids (Figure 2.11), which are based in a way that is similar to real Warehouses and with long corridors. Considering the map in question, it is necessary to perform some methods for setting agent's sources and goals, the most common being:

- **Random:** Set the source and goal vertices by randomly choosing vertices in the graph (must be a path in the graph between the two);
- **Clustered:** Set the first agent's source and goal randomly, and all the other agents with distance  $r$  from the first agent's source and goal;
- **Designated:** Set the source of each agent randomly from a possible set of source vertices, and the goal in the same way;

Finally, the *objective functions* most used to evaluate a MAPF solution are the **Makespan** and **Sum of Costs**. The *Makespan* is the number of time steps required to all agents reach the goal, and for a MAPF solution  $\pi = \{\pi_1, \dots, \pi_k\}$ , the makespan of  $\pi$  is defined by Equation 2.22. The *Sum of Costs*, also known as *Flowtime*, is the sum of time steps required for each agent reach the goal, where the sum of costs of  $\pi$  is defined by Equation 2.23 [62].

$$makespan_{\pi} = \max_{1 \leq i \leq k} |\pi_i| \quad (2.22)$$

$$Costs_{\pi} = \sum_{i=1}^k |\pi_i| \quad (2.23)$$

## 2.2 Warehouses Systems, Designs and Typologies

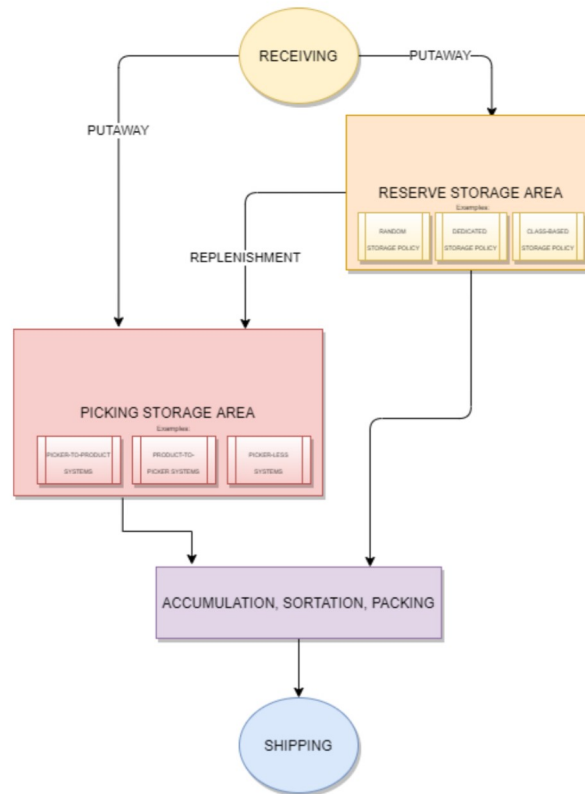
One of the first references to the storage of goods as a form of trade can be found in ancient Rome, where the increase in trade caused the need to construct buildings where this type of work was carried out, called *horreas* (for more information see [49]). Over the years, it was possible to notice the development of this system gaining strength with the creation of railroads in the United States and the great impulse and need for storage of goods during the consecutive Industrial Revolutions [1].

After the Second World War and with the new trend of technologies and research development in the field of Data Analysis, Artificial Intelligence, Machine Learning, and Robotics, the Warehouse system was divided into several Typologies, Layouts, and Systems. According to [57], a Warehouse is an industrial building, characterized as a facility primarily used for storage or distribution of materials. We can complement this technical definition through a more general view of its use today, [48] defines that the main function of a Warehouse is to satisfy the form of exponential consumption, efficiently meeting the needs of the consumer so that the product is delivered in the right time, quantity and quality.

The typical flow of goods in a Warehouse can be seen in Figure 2.8 [40], and according to [9], the whole process can be described as follows:

- The goods received are unloaded, where quantities are checked and quality checks are carried out;
- The approved goods are prepared for transport to storage, where a label, QR Code, Bar Code or Magnetic Label is added
- After transportation, the goods are stored in the Storage Area, following some storage policies [40]. If any material present at this location is requested, the process of *Order Picking* starts, where the products are listed, classified and sent to the Picking Storage Area;
- After that, the goods will be accumulated, assorted and packaged, thus proceeded to shipment.

Figure 2.8: An Typical Warehouse Flow and Operation



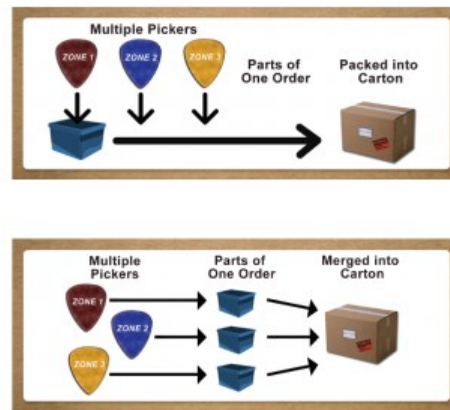
It is worth mentioning that more than 60% of all operational costs of a Warehouse are attributed to *Order Picking* [9], with *Packing* being the biggest throttle in the entire fulfillment process [47]. According to [53], the sub-processes of *Order Picking* that have a major impact on the time needed to perform the activity is the traveling and the search for items, totaling approximately 70% of the collection time, and becoming one of the most important factors for achieving better efficiency in the operational level.

Because a Warehouses normally has to deal with different quantities of SKUs, some fundamental concepts and taxonomies must be scored [9]. Some policies commonly used during a *order-picking* can be seen bellow [53]:

- When multiple orders need to be picked simultaneously in a single *order-picking* is called *batch-picking*, where picking is done in a certain zone (*zoning*);
- A *zone* can be created considering some variables such as product size, temperature and safety procedures [53];

- Although the *zoning* method has some advantages, such as reducing the distance covered by the *picker* [53] and consecutively reducing the collection time, the products will need to be sorted during or after the process [9], increasing the total process time;
- One of the ways to reduce the total process time in the *zoning* is to use methods like *sequential zoning/pick-and-pass* and *parallel zoning/pick-and-merge* [53]; As can be seen in Figure 2.9 [31], the *pick-and-pass* method is characterized by the collection of products from a *order*, made zone by zone sequentially until all parts of an order are reached. The *pick-and-merge* is the collection of products from a *order* from different zones at the same time, followed by the merge of all order parts;
- The product sorting can be done during the process (*sort-while-pick*) where the orders are separated into specific containers for each individual order, or after the execution of the same (*pick-and-sort*), where *wave picking* can be done in turns in the respective zones at the same time.

Figure 2.9: *Pick-and-Pass* and *Pick-and-Merge* Methods



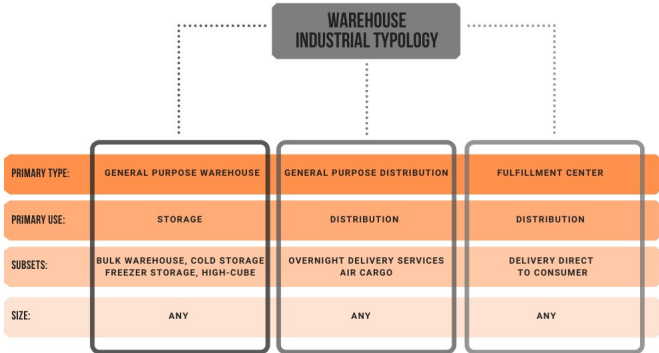
### 2.2.1 Warehouse Typologies

A Warehouse can be divided into several categories, as seen at Figure 2.10 [57]. Other typologies can be found in [9], which defines two more types of Warehouses, such as *Production Warehouses*, characterized by the storage of raw materials, semi-finished and finished products, and *Contract Warehouses*, responsible for the operation on behalf of one or more consumers. The Warehouse typology is directly linked to the type of activity developed, the



type of product or goods, and the form of logistics and product management. For this study in question, only two types are interesting to be analyzed, much of this due to the large volume of material and goods that pass through, requiring a more efficient and faster logistics: *Fulfillment Centers* and *Distribution Centers*.

Figure 2.10: Industrial Warehouse Typologies



**Fulfillment Centers**

According to [57], a Fulfillment Center (FC) (formerly called *Packing Warehouses*) has the distribution of goods as its primary use feature and can have different sizes. This type of facility can also be defined as an industrial property that allows an efficient movement and transport of goods directly to the consumer. The main focus is on delivering a final good to the consumer, executing order receipt, checking, labeling, packing, and shipment. Because this contact with the final consumer, can work with thousands to millions of Stock Keeping Unit (SKUs), it also has some characteristics as work in high speeds with good efficiency rates, individual piece pick or small parcel pick, and packing in corrugated boxes [47].

**Distribution Centers**

Although its definition is similar to a *FC* [57], a Distribution Center (DC), it is a Warehouse Facility with finished goods ready to be redistributed or distributed locally to internationally [37]. Like the *FC*, its primary use is the distribution of goods, but it differs in one special function, the *DC* is *demand-driven*. This works using an adaptable network focused on

a value-based outcome that changes according to the variables market share in near real-time. Other characteristics of this type of Warehouse that differs from the previous one can be described as picking on a larger scale through pallets and case quantities and a varied method of packing [47].

A distribution center also has some subdivisions and taxonomies, which are classified depending on their region location and activity. For example, according to [16], a *European Distribution Center (EDC)* is a Warehouse of Type *DC* that distributes to at least five different countries. The four most common types of EDCs and their taxonomies can be seen in the Table 2.3 adapted from [16].

Table 2.3: Taxonomies of *EDCs*

|                                    | SKUs Handling   | Warehouse Size | Objectives  | Additional Information   |
|------------------------------------|-----------------|----------------|---|--|
| Warehouse EDC (WHS)                | Medium, High    | Medium, Large  | Traditional Activities<br>(Storage, Picking, Transportation)  | Created as part of a Large System<br>Divided into Different Business Units |
| Warehouse/Office EDC (WHS/OFF)     | Low             | Small          | Traditional Activities and<br>Management-Related Activities   | Typically Distributes to a Global<br>Market.                               |
| Warehouse/Management EDC (WHS/MGT) | High, Very High | Large          | Traditional Activities and<br>Intensity Management-Related<br>Activities (Forecasting, Inventory, etc.) | Typically Focused on Local Market  |
| Warehouse/Factory EDC (WHS/FAC)    | Very High       | Large          | Different Activities<br>(Traditional, Technical, Management, etc.)                                      | Characterized by a High Level of Value<br>and High Intensity of Activities |

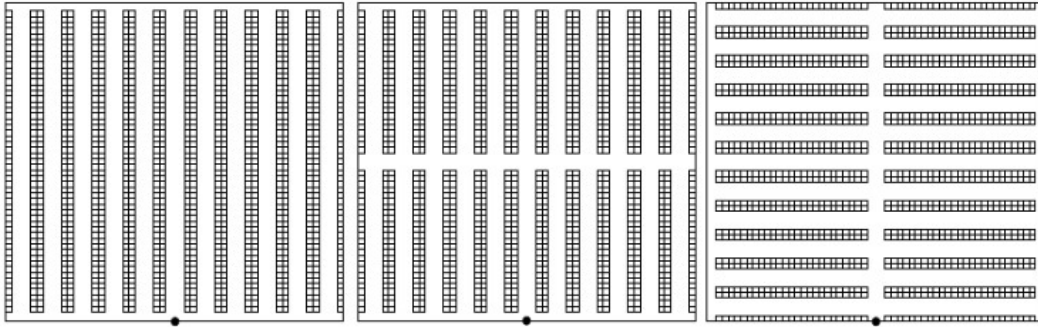
### 2.2.2 Warehouse Layouts

The warehouse layout design is one of the most important components to achieve optimization since it has a direct influence on the process of *order-picking*. To determine the ideal layout, it is necessary to consider the number of warehouse blocks and their sizes, the number of rack levels, and the locations of the *pickup and deposit (P&D)* of products [27]. According to [40], the types of Warehouses Layouts can be divided in *Traditional Layouts*, *Flying-V Layouts*, and *Fishbone Layouts*. However, some research has been carried out in the last years to discover new ways to optimize the layout of Warehouses, whether regular or irregular, new designs were proposed, like the *V-Shaped Layouts* [76], *Chevron aisles*, Layout Design from Multiple (P&D) Points [27], and *V-Shaped Layout* [76].

According to [78] Warehouses generally follow two basic design rules: *Picking aisles* must be straight and parallel to each other and *cross-aisles* must be straight. Considering

these rules, the layout of a Traditional Warehouse is similar to that found in Figure 2.11, adapted from [27].

Figure 2.11: Traditional Warehouse Layout Design

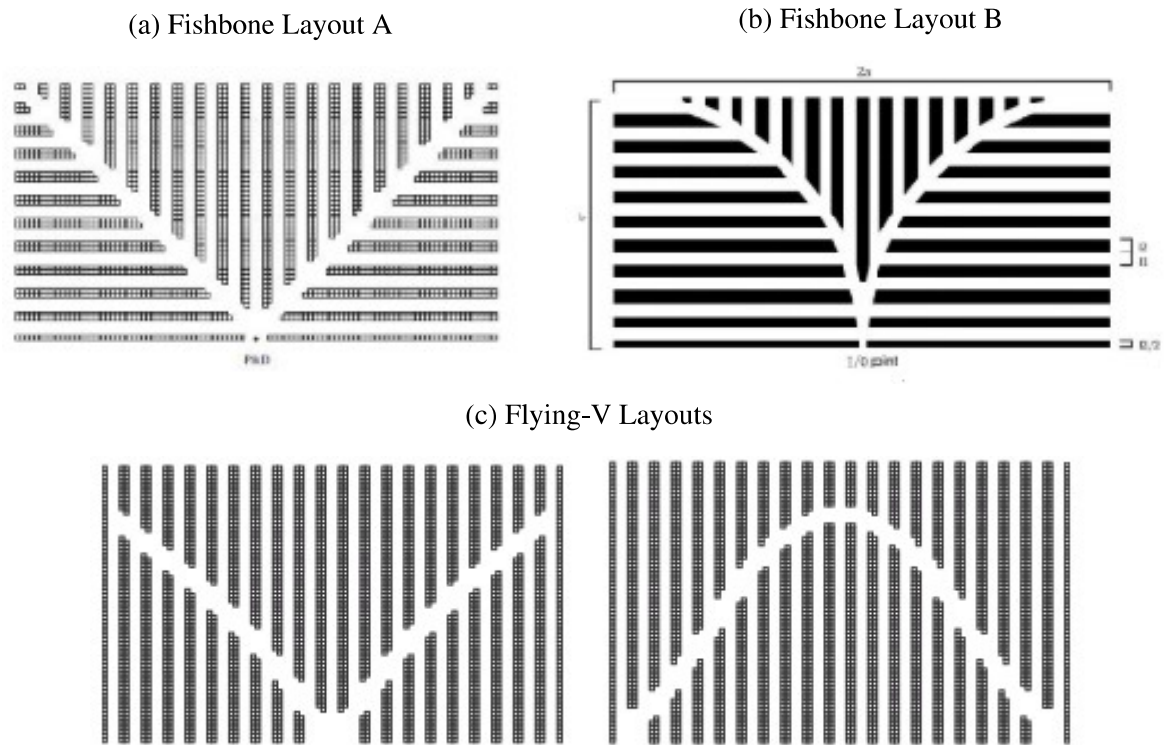


The selection between the layout designs will directly depend on the location of the P&D points. Considering the traditional design, the layout design of the Figure 2.11 (middle) and the Figure 2.11 (right) have greater efficiency (smaller traveled time) compared to the traditional format Figure 2.11 (left). It is also worth considering that the Figure 2.11 (right) is more efficient in several parameters regardless of the location of the P&D than the Figure 2.11 (middle), but the last one is more common in practice [41].

However, as seen at Figure 2.12 (adapted from [79], [76], and [27]) it was discovered that *Flying-V Layouts* (Figure 2.12a, 2.12b), and *Fishbone Layouts* (Figure 2.12c) can reduce up to 20% the traveled distance during the execution of *batch-picking* and 15% when it is necessary to carry out retrieval and storage in the same run (*dual-command*) [79] compared to traditional designs.

The *Fishbone Layout* design can reduce the time travel by up to 20% when compared to traditional warehouses, and up to 15 % when running *dual-commands*, however, it has the disadvantage of limited access to the storage space [41]. The *Flying-V Layouts* are less efficient than the *Fishbone*, reaching around 10% efficiency compared to traditional warehouses [41], but if the P&D points are located in front of each *picking aisle* the design of Figure 2.12 (left) can reduce traveling by about 3% to 6%, while the design of Figure 2.12 (right) can provide up to 2% reduction compared to a Traditional Warehouses with considerable size [27]. It is worth noting that one of the biggest drawbacks of these layouts concerning traditional design is that the facility must be 3-5% larger [41].

Figure 2.12: Irregular Warehouse Layouts Design



### 2.2.3 Warehouse Systems

According to [9], a *Warehouse System* can be defined as a set of equipment and policies applied to the collection, storage, and removal of items. Considering the level of automation applied to activities, been these physical, data-driven or decision-making, we can divide the system into the following: *Manual Warehouses*, *Automatic Warehouses*, *Automated and Robotized Warehouses*, *Multi-Shuttle Warehouses*, and *Smart Warehouses*. As the explanation of all the aforementioned systems would become exhaustive, only the **Automated and Robotized Warehouses** will be addressed, since it will be the systems proposed in this thesis.

#### Automated and Robotized Warehouses

An Automated Warehouse is the type of system characterized by a considerable degree of intelligence to collect and process items with minimum human interference, or when a robotic system (AGVs, SDVs, Cooperative Robots, etc.) is added to perform the *order-picking* ac-

tivities [9]. This type of system is generally used when you have items with high added value, small size or when the focus is high productivity and accuracy. The types of *picking equipment* most used in this type of system are *Layer Pickers*, *Dispensers* and *Robots* [53] (see Figure 2.13). For [15], although *Automated Warehouses* are characterized by complex modeling due to several components interacting with each other, this type of system has widely used *shuttle-based storage and retrieve system* for picking automation. In addition to these types of picking equipment, the recent field of *Automated Mobile Robots (AMRs)* applied to logistics and *Warehouses* has grown considerably in recent years, and could reach an investment of \$ 290 billion by 2040 [24].

Figure 2.13: Automated and Robotized Warehouse Robots

(a) Kiva Robot from Amazon Robotics



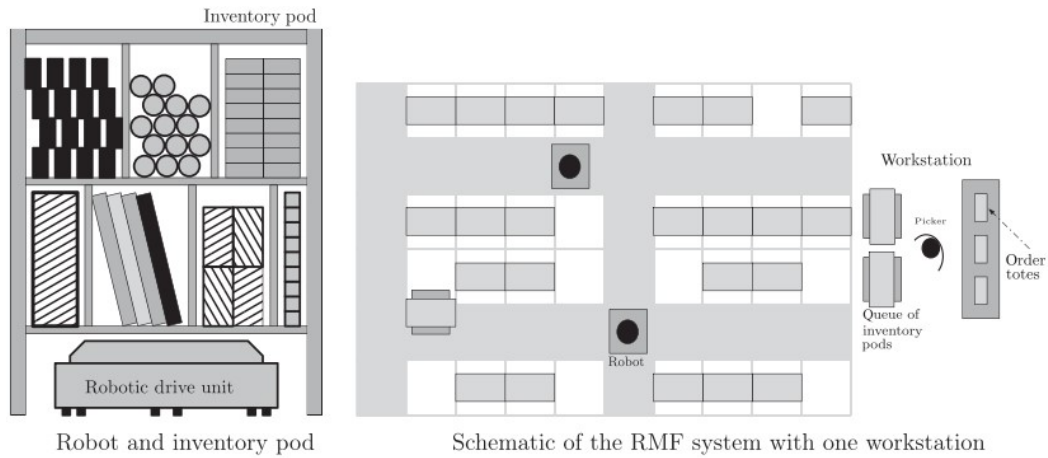
(b) Adapto Shuttle-based System



### 2.2.4 Robotic Mobile Fulfillment Systems (RMFS)

According to [4], an RMF system is based on three major components: Robots that communicate in a centralized or decentralized way (*Robot Drive Units*), the movable shelf racks that contain the stored products (*Inventory Pods*), and areas where workers perform replenishment, picking and packing *Workstations*. A representation of this system according to [4] can be seen in Figure 2.14.

Figure 2.14: An Generic RMF System



Although the concept of *RMF* is based on the characteristic of lift and transport movable shelf racks ([4], [39]), recently, new AMR technologies applied to the Distribution and Fulfillment Centers were launched for the *sorting*, and *compact picking* processes. In this way, we can define it as: **a *goods-to-person* AMRs based system, responsible to carry and transport small or large volumes of products to execute *order-picking* and *sorting* processes.**

### Autonomous Mobile Robots Applied to RMFS

A mobile robot can be defined as a set of elements that allow it to move around under its control [21]. If he can understand his environment without human interference, he can be considered autonomous [20]. Generally, his autonomous path planning is made through an array of sensors, cameras, and maps that allow materials to be transported without the need for coordination with other transport flows (asynchronous transport).

Although this type of solution has a similarity with the automatically guided vehicles (AGVs) and the automated guided carts (AGCs), they have some important differences as can be seen in Table 2.4. In 2012, after Amazon<sup>®</sup> bought Kiva Robotics, a new era of AMRs applied to Distribution Centers and Fulfillment Centers started. Although the conceptual idea of *Kiva Robots* was conceived in 1989 [4], the solution was implemented only in 2008 [25].

Table 2.4: Main Differences Between AMRs, AGVs and AGCs

|                                    | Automated Guided Vehicle (AGV) | Automated Guided Cart (AGC) | Autonomous Mobile Robot (AMR) |
|------------------------------------|--------------------------------|-----------------------------|-------------------------------|
| <i>Rigid Preset Routs</i>          | O                              | O                           |                               |
| <i>Fixed Infrastructure</i>        | O                              | O                           |                               |
| <i>Infrastructure Requirements</i> | O                              | O                           |                               |
| <i>Product Transport</i>           | O                              | O                           | O                             |
| <i>Picking Process Assistance</i>  | O                              | O                           | O                             |
| <i>Sortation Assistance</i>        |                                |                             | O                             |
| <i>Redeployable</i>                |                                |                             | O                             |
| <i>Intelligent</i>                 |                                |                             | O                             |
| <i>Modular Deployment</i>          |                                |                             | O                             |

## 2.2.5 System Architecture

One of the first articles found that proposed an AMR solution applied to the transport of objects can be found at [12]. The author uses sensors and an operational control center to better locate the robot, also adopting a global location path planning based on static data, a local sensor-based path planning for obstacle avoidance, Object and Pose Recognition to identify the pallet to be transported, and uses sensor fusion to obtain pose and object recognition. A more complete approach to a framework designed exclusively for warehouse systems compared to the previous can be found at [63]. The authors used the concept of distributed autonomous intelligent units (*cells*) (actually called AMRs) and studied the effect of some path planning algorithms. This framework is much more robust than the previous one, having a global memory, and a task manager in the highest control layer activity flow. More recent architectures can be found at [35] and [70]. The first addresses the creation of a cyber-physical system model for smart warehouses, which performs the development of a complete CPS-enabled framework, covering topics of data acquisition, path planning, wireless sensor network, collision detection with time window, and some robot strategies (avoidance, go-away, detour, and wait-before-startup). The second research features a modular robotic system with aisle-captive robots applied to small or medium-sized warehouses. It is worth mentioning that different from the previous frameworks, both studies propose a congestion-free system.

Regarding simulators, an framework proposal can be found at [72], characterized by an agent-based discrete-event simulation framework designed to study the context of an RMFS

while evaluating multiple decision problems jointly. The simulator also have an application programming interface (API) for integrating with ERP and other enterprise systems, and an implementation of the agent interface for robots and station apps. To validate the simulator, experiments were carried out with real robots on a 3x4 Grid Map and a location system based on a *QR Code* Grid Map.

### 2.2.6 Performance Improvements

A simpler and general review about this topic can be found in [28], were address the basic performance characteristics, the relationship between contexts and system design *RMF* with respect to performance, and some important taxonomies in this type of system. In [34] the first model of a *RMF* system is designed based on analytical models. System performance and robot usage values are estimated through parameters such as warehouse layouts and control policies. In addition, the article provides information on performance impacts regarding the length-to-width ratio, workstations locations, number of robots and number of orders that need to be completed per hour. In this way, it becomes an excellent reference for comparing results when creating new models or improvements.

Considering the previous impacts the following question can be made: What is the optimal speed and number of robot values in a Warehouse? The answer to this question can be found at [75]. The authors elaborate a study with numerical experiments and simulation of the analytical results using the Arena 14.0 software, describing the formulas that can be used to optimize the values of speed and number of robots for a mobile picking system. Another topic with great importance in the management of robotized warehouses and already studied previously on AGVs is the battery management problem. According to [77], a bad policy can cause an additional cost of 15% to the system. According to the authors, factors such as battery cost, cost of robots, and a small required retrieval transaction throughput time directly influence in the battery management policy selection.

Finally, some other variables can also directly influence the performance of a *RMF* system, the research found in [50] performs the study of robot assignment strategies in storage areas where any pod can be stored in any available location, and the effect of this on system performance. A study is about the energy efficiency of a *RMF* system based on the work performed by the robot can be found in [73], and in [67] the impact of not using safety



constraints is compared relative to operational inefficiency.

### 2.2.7 Discussion

Notably, the interest in this field of research received attention after the solution proposed by the Kiva Robotics in 2008, where the traditional AGVs were replaced by several autonomous mobile robots with a basic notion of intelligence. Although the first proposed solution uses only a traditional A\* algorithm, several studies can now be found promoting ways to improve system performance and efficiency. This field of study is relatively wide, as an RMFS addresses problems such as Network Design, Internet of Things, Cloud Computing, Scheduling, Path Planning, Database Implementations, Pick & Order, Delivery, Routing, Location Assignment, Aisle Computation & Performance, Order Batching, Packing, Ideal Velocity, Ideal Number of Robots, among others. And each of these areas is further expanded with the different types of Warehouses Design (Traditional, Flying-V, Fishbone), thus promoting different forms of analysis and solutions.

Considering the state-of-the-art reviewed, despite the considerable amount of studies found some research gaps were found. Regarding the robot localization, it is quite common to find articles with the proposal of *QR Codes* on the floor, ceiling, or natural landmarks. However, an increasing number of articles using LIDAR and other sensors for localization has also become present, despite its high cost considered with previous methods. Finally, regarding the performance and Improvements, although some articles are found demonstrating the impact of good scheduling for battery charging or Order & Picking on the robot battery and distance traveled, a large research gap is: **how to use the heavy data available in the environment (from the robots, sensors, networks) in a way that improves the system efficiency**. Some authors even propose new studies that use techniques of Data Analysis, Business Intelligence, or Machine Learning to improve decision making or creation of techniques and solutions to analyzes the robot's Health and Monitoring [14]. The latter topic can considerably increase the efficiency of the system through Forecasting techniques that can be attached to the heuristics of the robot's path planning.

# Chapter 3

## Related Researches

This chapter will present the latest state-of-the-art developments, addressing some problems, topics, and fields that will be implemented or elaborated in this dissertation. After the presentation of this summary, a brief discussion about the reason for choosing specific frameworks and platforms will be made.

### 3.1 Multi-Agent Path Finding

By definition, the path finding for multiple agents can be **Cooperative**, where the agents are aware of the others and their routes, **Non-Cooperative**, where the agents are not aware of the others and must predict their movements, or **Antagonist**, where each agent tries to reach his goal avoiding that the others reach theirs [59]. The Multi-Agent Path Finding, as previously mentioned in Chapter 2, is when multiple agents plan their route avoiding collisions with the others, thus being *an cooperative way to find a path*.

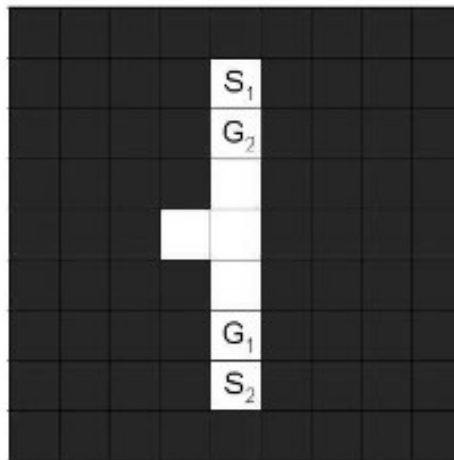
Although the first concepts that would initiate the field of Multi-Agent Path Finding date from 1987 [17], and that the traditional A\* can be adapted to make a new route if necessary [64], the real start of this field of research began to gain strength after the first publication of three A\* modifications that allowed a new way to approach the cooperative pathfinding [59]. One of the major challenges in trajectory planning for multi-agent systems is the possibility of breakdown due to bottlenecks, deadlocks, and cyclical repetitions, and to avoid this problem the authors developed new four A\* based algorithms, the **Local Repair A\* (LRA\*)**, **Cooperative A\***, **Hierarchical Cooperative A\* (HCA\*)**, and the **Windowed**

**Hierarchical Cooperative A\* (WHCA\*).**

According to [59], the operation of these algorithms is relatively similar in its foundation, being then differentiated in some specific changes. The *LRA\** algorithm is a family of algorithms widely used in the gaming industry, *where each agent performs the route search using a traditional A\* and ignoring other agents with the exception of neighbors*. If during his movement he will occupy a position that may cause a collision he will recalculate his route to the goal avoiding it. As can be imagined, this results in a higher computational cost, in addition to being unfeasible for very crowded regions.

The *CA\** (*Space Time A\**, e.g.) difference from the previous algorithm is the ability performs a search in the three-dimensional space  $(x, y, t)$  taking into account the route of the other agents, where the action of *wait* is used so that a given agent remains stationary if in the next instant of time it collides after the movement. In short, routes are added to a *reservation table* (implemented as a hash table) that will be avoided by the next agents when searching for a route. As a heuristic function, the distance from Manhattan is generally used, but any additional heuristics can be implemented in a way that reduces the computational cost. One of the shortcomings of this algorithm is that any decoupled, greedy algorithm that precalculates the optimal path will not solve some problems, like the one in Figure 3.1 (adapted from [59]). More information about the *CA\** and how to implement it can be found in [46].

Figure 3.1: Where the Cooperative A\* Fails



The *HCA\** algorithm is nothing more than an attempt to improve the heuristic used by

CA\*, where the abstract distance is computed according to demand, and the dimension of time and the reservation table are ignored. To achieve this the authors used the Reverse Resumable A\* (RRA\*) to calculate abstract distances. Finally, the last implementation by the authors is the WHCA\* algorithm, where through a "window" each agent will search for his partial route to the destination and start moving his path. Using continuous intervals of time the window is shifted forwards, and a new partial route is computed. In this way, an agent's goal is no longer to reach his destination but to complete the window via the terminal edge, also allowing the processed time to be shared with the other agents. Lastly, the authors developed 10 tests in a maze-life environment, where the environment is a 32x32 4-connected grid with obstacles randomly placed in 20% of the space available. The authors concluded that the LRA\* is highly influenced by crowded maps causing bottlenecks, being unfeasible in real situations, while the others managed to cope well with the situation, achieving only 2% failure for tests with 100 agents. Finally, CA\* was the algorithm with most computational cost, taking about 1s to calculate 100 routes, followed by HCA\* with 1ms per agent, and WHCA\* with 0.6ms per agent.

Although this is one of the main articles in the field, others have also been published over the years, either promoting a new solution or making new implementations and heuristics for the previously proposed algorithms. In [68] a tractable algorithm for multi-agent path planning (MAPP) based on grid maps is elaborated where costs are kept low by eliminating the need for replanning, and is characterized by low polynomial complexity in time, space, and quality of the solution. In [60] the RRA\* is used with two additions, the use of a Simple Independence Detection (SID), which assumes that the paths of all agents are independent and computes cooperatively the paths with conflict using an optimal algorithm, and Independence Detection (ID), which starts each agent in a group, searches and tries to find the most optimal path.

In [69], an improvement of the MAPP is made by adding target isolation, a buffer area to solve bottlenecks, and a new strategy that reduces the number of movements. These modifications allowed an efficiency of 92 to 99.7 % even in scenarios with 2000 agents, and reduced the distance traveled by more than 50 %. In [54] the first generalization of the Conflict-Based Search (CBS) is applied to MAPF, this being characterized as an algorithm divided into two levels, the highest level is responsible for executing a tree based on the

conflicts of the agents and the low level is responsible for research for a single agent at a time. New implementations and improvements for CBS are also made in the following years by [6], [10], [55], [3], and [19] consecutively.

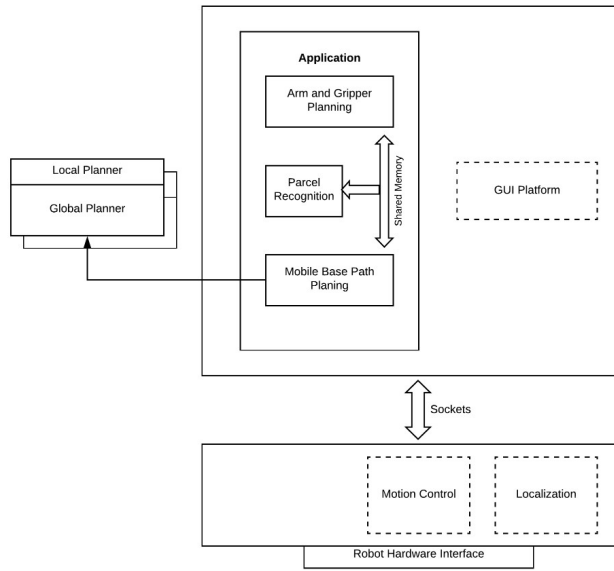
In [30] the first MAPF study is carried out with kinematic constraints and real tests with robots, where the MAPF-POST is elaborated, being responsible for the use of a temporal network that performs MAPF post-processing and allows the use in differential-drive robots taking into account their maximum speed. At [62] a summary of MAPF definitions and variations is made, presenting for example the *Online MAPF (Lifelong MAPF, eg)*, characterized by a sequence of MAPF problems that must be solved in the same graph. One of the research subtopics included here is the use of **Warehouse Models**, where an agent can be directed to other tasks, being this inspired by Automated and Robotized Warehouses. A more in-depth analysis of the most used algorithms in Online MAPF can be found at [65], who presents the Replan Single (RS), characterized by the search for an optimal path for each new agent one at a time, Replan Single Grouped (RSG), where we search for the optimal paths for all agents at once, and the Online Independence Detection (OID), based on the ID algorithm [60].

## 3.2 Path Planning Algorithms in RMFS

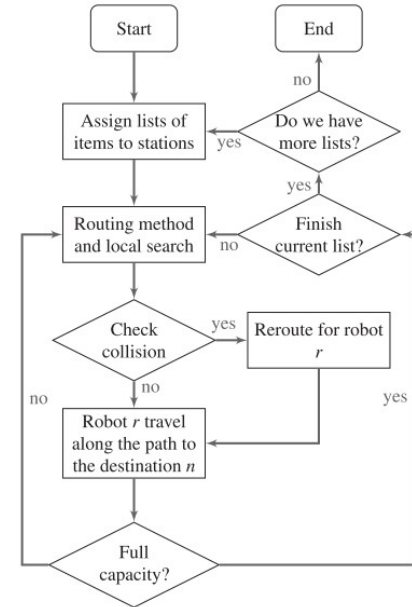
The architecture proposed by [12] can be seen in Figure 3.2a. The path planning algorithm was divided into two hierarchical levels: a global (lowest) and local (highest) planning. The global planning used the visibility graph approach in conjunction with the Dijkstra algorithm, and the local planning used the local criteria and heuristics. The two planners were combined together with the localization algorithm (*Extended Kalman Filter*) was used with reflectors placed in the workspace, making possible to move the robot around the environment. More recent and relatively similar work can be found at [45] (see 3.2b) and [5]. Both uses the *Dijkstra's algorithm*, however the first one uses the *Manhattan distance as heuristic method to find the shortest path*, the *breadth first search (BFS) with limit node depth as local search*, and the experiments were carried out with one agent. The last one uses the same algorithm for Path Planning (Dijkstra's Algorithm), however, the heuristic function used is the two-dimensional Euclidean space consisting of multiple tiles in a Warehouse Grid Map

with 100x100, using 5 to 25 robots randomly allocated. According to the authors, the time required to find a solution for each delivery task is 17.36 seconds for a system with 25 robots

Figure 3.2: Path Planning Methodologies generally used before the MAPF



(a) AMR System Architecture Proposed by [12]



(b) RMFS Path Planning Proposed by [45]

With the advancement of research on the Multi-Agent Path Finding Problem (which was more focused on AI and Games), new Path Planning algorithms were developed and applied to Warehouse Systems. The beginning of this research field received more attention after the work published by [71], characterized by the first RMFS implemented, where the first Path Planning version was a standard implementation of the A\* Algorithm to plan the robots paths to storage locations and inventory stations. One of the main features of this system is the delivery of a new pod every six seconds, processing in real time, expandability, a multi-agent architecture, and a two-dimensional Grid graph used as Map Representation. According to [72], the Multi-Agent Path Finding (MAPF) is a challenging problem that can be applied in robotics, and many of the MAPF algorithms do not consider kinematic constraints, such as maximum velocity limits, maximum acceleration, maximum deceleration, turning times, and the movement is only considered in discretized environments. Furthermore, those who consider kinematic constraints work only for a small number of agents and are very slow for

a massive search like an RMFS.

The first article found that compiles the MAPF algorithms in recent years, briefly explaining how the areas of AI, Robotics, and Computer Theory address this problem can be found at [38]. One of the important characteristics of this work is the analysis of how an Automated and Robotized Warehouse raises an interesting number of optimization problems, such as trajectory planning and where and when "pods" must be moved by the Warehouse. A feature shared in the latest state-of-the-art multi-agent path-finding algorithms applied to Warehouses **is the inability to find bounded-suboptimal solutions for 100 robots in small warehouses in real-time, since there's a direct relationship between a tight space and the runtime.**

In [42] there is an interesting compilation that addresses some of the most used algorithms in RMFS systems, these being: WHCA\*, FAR, BCP, OD&ID, and CBS. The authors perform complete tests, which analyze the idle time, Timeouts, Memory Average, Maximum Memory, and Handled Units per instance. According to the authors, WHCA\* was the one that had the best performance, but did not scale well. CBS was the one that suffered the most in long instances, but it becomes more efficient than WHCA\* for instances with the lower robot to station ration, and it was the one that used less memory compared to the others. Some recent state-of-the-art algorithms already address the Pick & Delivery issue, as in [26] and [36]. The first study proposes a new centralized heuristic for Online Multi-Agent Pickup and Delivery (MAPD, reducing the service time 43%, And the second addresses solutions for the deadlock and the salesman problem. In [29] a framework that can be added to the current MAPF planners solving the problems of the continuous movement of robots through the Action Dependency Graph (ADG).

The most recent work finally develops a framework that can produce a high quality of solutions for up to 1000 agents [66]. According to the authors, most of the methods that resolve the Lifelong MAPF include: **resolving the system as a whole, decomposing the system into a sequence of MAPF instances at each time-step, where one re-plans the path for all agents, or decompose the problem in a sequence of instances where planning is only done for agents with new goals.** Each method previously described has its defects, such as the need to know all the goals affecting the scalability, to re-plan all the paths at each time-step becoming time-consuming, or to need additional structures to guarantee completeness.

The authors then modify the Multi-Label A\* [26] for single-agent trajectory planning, where the framework (Windowed MAPF) is responsible for resolving collisions in a specific time window.

### 3.3 Discussion

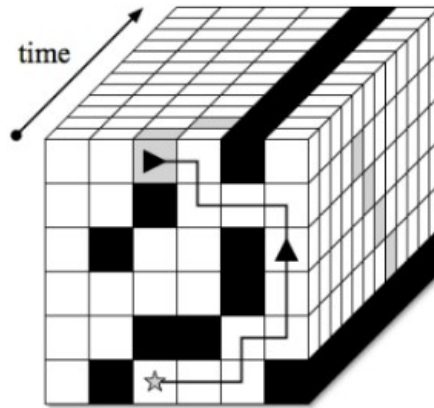
All studies reviewed in Path Planning the state-of-the-art for RMFS focused to solve two problems: how to achieve better efficiency of an already known algorithm or how to develop a framework (or algorithm) to solve some of the MAPF problems. In this way, most articles provided only theoretical bases that complement results from previous articles executing tests with around 100 agents. Few articles carried out real tests with robots, be it for solution analysis or hypothesis tests, and among those that tested only two had real implementations of planning algorithms working in real-time. Much of this is because of the NP-hard characteristic and the point that MAPF does not have a correct solution yet, becoming then more theoretical than experimental. However, the MAPF applied to RMFS is also not focused only on the issue of computational optimization, few simulators were found in a way that allows the assignment and testing of new algorithms for Warehouses of different layouts. This is another research gap found, all articles found promote MAPF solutions are only considering the Traditional Layout, not having performance analysis for unconventional layouts like Fishbone and Flying-V for example. Again, due to the complexity of MAPF, in addition to being a recent issue in Automated Warehouses where the best solution is unknown may be one of the reasons why other layouts have not yet been explored.

The MAPF is a well-known and well-studied problem in Artificial Intelligence that has gained attention in recent years, not only for researches in optimization, but also because migrated to the fields of Games, Robotics, Urban Traffic, and Networks. Despite the conceptual approach dating from 1987, it was only in 2005 that the first algorithm was developed trying to solve the path planning for multiple agents using a cooperative approach (CA\*). Although several algorithms have been developed since this date, one of the biggest difficulties still faced today is how to develop a search for  $n$  agents still allowing a robust, complete, and optimal solution. This is an interesting problem, because the complexity of the problem is polynomial in the grid size and max time, and is also influenced by the total number of



agents, becoming then NP-hard.

Figure 3.3: The Reservation Table



But why does this happen? Let's review some of the available algorithms. To solve this problem, the CA\*, LRA\* and WHCA\* algorithms use the interesting idea of reservation-based planning (see Figure 3.3), initially proposed in [59], which increases the search speed since it needs almost no coordination, however, it is incomplete and not optimal. Some complete algorithms can also be applied in MAPF such as the *Push & Swap* and Bibox, however, they are far from optimally and have a complex formulation. Other algorithms like M\* and CBS, although they can become optimal, complete, and efficient, are variable dependent, where the solution depends on the free space, bottlenecks, number of agents, graph size, heuristic accuracy, and conflicts.

Another research gap is also Lifelong Planning, characterized by MAPF planning for multiple goals. As it is an extremely recent issue, first presented in 2017 [42], some several questions and paths still need to be explored. One is how the Pick and battery recharge can be attached to the MAPF planner in a way he understands when and why a robot should recharge its battery considering its paths or future actions. As in the previous discussion, this theme opens new doors for emerging technologies such as the use of Machine Learning for Forecasting, or the determination of new heuristics based on the Health of the robot. It is possible to realize then that MAPF has a wide opening for new solutions, be they to demonstrate what works and what does not, thus allowing the growth and maturity of new implementations.

Regarding the Map Representation, the *Fixed-Size Cell Decomposition* will be used, and

the Map Representation will be a *Grid Map*. The reason for this choice is due to the considerable number of studies found in the state-of-the-art that uses this representation to run the path planning algorithms ([5], [33], [35], [45], [70], e.g.). Regarding the Path Planning Algorithms applied to RMFS, although research has intensified in recent years involving both MAPF and MAPD, there are still several fields of study open [52]. Some of them are: how to understand the "how" and "why" this problem is computationally hard, how adding "learn from experience" can decrease computational difficulty, how to work with the MAPF or MAPD distributed in multiple machines without losing completeness, how to do "lifelong" planning for automated warehouses, and how to handle heterogeneous robots.

Considering this, it is clear that this is an area that is still under development, much of it is relatively new and open to new solutions that try to promote a new way of solving the problem. In Table 3.1 it is possible to see a brief comparison of some state-of-the-art researches with the implementation made in this work.

Table 3.1: Related Researches Comparison

| Approaches | Number of Robots (Max) | Type of Environment | Type of Layout   | Size (Max.)   | Decentralization | Focus                                   |
|------------|------------------------|---------------------|--|---------------|------------------|---|
| [71]       | Undefined              | Grid                | Traditional Horizontal                                     | Undefined     | Undefined        | First Implementation of RMFS            |
| [45]       | 5                      | Graph               | Mixed (Horizontal + Vertical)                              | 45x25         | Not Possible     | MAPF                                    |
| [30]       | 4                      | Grid                | Layout for Validation Only                                 | 5x4           | Not Possible     | Implementation of Kinematic Constraints |
| [29]       | 50                     | Graph               | Traditional Horizontal                                     | 16x8          | Not Possible     | MAPF                                    |
| [35]       | 5                      | Grid                | Traditional Vertical                                       | 20x10         | Not Possible     | MAPF                                    |
| [5]        | 25                     | Grid                | Mixed (Horizontal + Vertical)                              | 100x100       | Not Possible     | Task Routing                            |
| ST-SPF     | 250                    | Grid Graph          | Traditional Vertical<br>Traditional Horizontal<br>Flying-V | Up to 100x50  | Possible         | MAPF                                    |
| STMS       | 48                     | Graph               | Traditional Vertical<br>Traditional Horizontal<br>Flying-V | Up to 50 x 14 | Not Possible     | MAPF                                    |

# Chapter 4

## The ST-SPF & STMS Algorithms

This chapter aims to describe the implementations and developments made for the path finding algorithms and the simulator. In Topic 4.1 the algorithms used in low-level search will be presented. Finally, in Topic 4.2 and 4.3, the proposed ST-SPF and STMS operating principle will be respectively presented.

### 4.1 Low-Level Algorithms

In the study of art, despite the fact that most algorithms use the A\* or its variants, was also possible to find some proposals with the Dijkstra's and the Breadth-First Search. Thus, it became necessary to implement and test these three algorithms to identify which one performs better for low-level search.

#### 4.1.1 The Breadth-First Search Algorithm

By definition, the Breadth-First Search traverses and search a tree or graph starting at the tree root. One of its main differences is the feature of *First-In, First-Out* when searching for the best path. The developed algorithm uses the Python 3+ *deque* module to perform the search operation, and can be viewed in the [8]. One of the reasons for using this algorithm is due to its ability to find the shortest path, making this a favorable point to possibly reduce Makespan when compared to Traditional A\*.

### 4.1.2 The Dijkstra's Algorithm

The Dijkstra algorithm is widely used in robotic applications to find the shortest path distance or minimum cost in a graph environment. Its main feature is the backwards search, going from the *goal* to the *start* position. The algorithm can be considered greedy and complete since there is a guarantee of finding the shortest path if there is a solution. Unlike the previous algorithm, the Python 3+ module *heapq* was used to elaborate the priority queue (heap) of the search algorithm and can be viewed at [8]. One of the reasons for using this algorithm in the tests is to evaluate the weighted graph developed by the simulator.

### 4.1.3 The A\* Algorithm

As mentioned in the state-of-the-art, most of the algorithms developed to solve the MAPF are A\* based due to its high performance and low branching factor. The Python 3+ *heapq* module was also used to build the priority queue (heap) of the search algorithm, and for the heuristic, the Manhattan distance was chosen as seen at [8]. One of the reasons for using this algorithm is that the A\* is the most commonly used in the state-of-the-art MAPF algorithms, being found and algorithms like the CA\* and CBS.

#### The Manhattan Distance

The reason for choosing this heuristic is its high efficiency in Grid Like Environments with four possible movements. This heuristic is also recommended when the dimension of the space may increase [2]. The software implementation can be seen at [8].

### 4.1.4 Low-Level Algorithms Comparison

The tests were run on a Notebook *ASUS Expert X23*, with an Intel I5 7200U 2.50 GHz Processor, an NVIDIA 920MX GPU, 8 GB DDR4 SDRAM, and an Ubuntu 18.04 operating system with PyGame v1.9.6 and Python 3.6. Before showing the algorithms results for different Warehouses Layouts, the following considerations need to be highlighted:

- The input graph is a 4-connected grid Bidirectional graph with some vertices removed to represent static obstacles (Pods, Workers, Treadmill) and positions that the robot

cannot move;

- For the Breadth-First Search a Simple Graph environment was used, and for the A\* and Dijkstra's Algorithms a Weighted Graph with weights in the recharge and pickup zones was used;
- The comparative tests between the algorithms have the same start and goal point;
- The memory measurement was done through the Python *Memory Profiler* library;
- The Warehouses Layouts Design are based on the layouts proposed in the state-of-the-art, as shown in Figures 2.11 and 2.12;
- The start and goal nodes are defined in the *Settings* file, which are modified according to the desired simulation experiments to new nodes locations;
- The goal is represented in green, the start position in red, the shortest path through the directional arrows, and in light gray it is possible see the nodes explored by the algorithm.

### **Traditional Horizontal Warehouse Layout**

A total of 9 tests were performed for each of the algorithms using the simulator proposed, where the location of the goal and start position were modified for each test. In Figure 4.1 is possible to review the results of the experimental evaluation where the Interactions are the number of interactions needed to find the solution, the Time is the algorithm Running Time, and the Depth is the number of nodes visited by each algorithm to find the solution.

Figure 4.1: Path Planning Results for the Traditional Layout

| Breadth-First Search (Traditional) |              |          |               |                  |       |
|------------------------------------|--------------|----------|---------------|------------------|-------|
| Test                               | Interactions |          | Time          |                  | Depth |
|                                    | While Loop   | For Loop | Wall Time (s) | Process Time (s) |       |
| 1                                  | 37           | 3        | 0.0037        | 0.0078           | 9     |
| 2                                  | 33           | 3        | 0.0011        | 0.0011           | 7     |
| 3                                  | 19           | 3        | 0.0006        | 0.0019           | 5     |
| 4                                  | 44           | 3        | 0.0014        | 0.0014           | 14    |
| 5                                  | 55           | 3        | 0.0012        | 0.0012           | 12    |
| 6                                  | 55           | 2        | 0.0015        | 0.0037           | 10    |
| 7                                  | 44           | 3        | 0.0013        | 0.0051           | 9     |
| 8                                  | 44           | 3        | 0.0018        | 0.0017           | 11    |
| 9                                  | 56           | 2        | 0.0016        | 0.0017           | 13    |

| Dijkstra's Algorithm |              |          |               |                  |       |
|----------------------|--------------|----------|---------------|------------------|-------|
| Test                 | Interactions |          | Time          |                  | Depth |
|                      | While Loop   | For Loop | Wall Time (s) | Process Time (s) |       |
| 1                    | 49           | 3        | 0.0007        | 0.0007           | 9     |
| 2                    | 48           | 3        | 0.0016        | 0.0015           | 7     |
| 3                    | 38           | 2        | 0.0010        | 0.0010           | 5     |
| 4                    | 55           | 2        | 0.0009        | 0.0034           | 14    |
| 5                    | 55           | 2        | 0.0008        | 0.0011           | 12    |
| 6                    | 56           | 2        | 0.0029        | 0.0070           | 10    |
| 7                    | 53           | 2        | 0.0007        | 0.0003           | 9     |
| 8                    | 48           | 3        | 0.0015        | 0.0045           | 11    |
| 9                    | 53           | 2        | 0.0008        | 0.0048           | 13    |

| A-Star Algorithm |              |          |               |                  |       |
|------------------|--------------|----------|---------------|------------------|-------|
| Test             | Interactions |          | Time          |                  | Depth |
|                  | While Loop   | For Loop | Wall Time (s) | Process Time (s) |       |
| 1                | 32           | 2        | 0.0005        | 0.0030           | 9     |
| 2                | 23           | 4        | 0.0006        | 0.0006           | 7     |
| 3                | 11           | 4        | 0.0002        | 0.0002           | 5     |
| 4                | 17           | 2        | 0.0004        | 0.0004           | 14    |
| 5                | 17           | 2        | 0.0007        | 0.0007           | 12    |
| 6                | 17           | 2        | 0.0003        | 0.0003           | 10    |
| 7                | 17           | 2        | 0.0009        | 0.0023           | 9     |
| 8                | 23           | 3        | 0.0007        | 0.0011           | 11    |
| 9                | 23           | 3        | 0.0009        | 0.0009           | 13    |

It is possible to notice that the A\* was the one with less running time than the others even having the same Depth, leading advantage also in the number of interactions in all tests. Meanwhile, Dijkstra's Algorithm fared similarly to Breadth-First Search but had to search a greater number of nodes to find the shortest path as seen in the Figure below. Regarding space complexity, all algorithms showed similar memory usage ( $\sim 75$  MB), one of the reasons may be the reduced size of the environment, resulting in a low memory requirement during search operations.

### Traditional Vertical Warehouse Layout

A total of 9 tests were performed for each of the algorithms using the simulator proposed, where the location of the goal and start position were modified for each test. In Figure 4.2 is possible to review the results of the experimental evaluation where the Interactions are the number of interactions needed to find the solution, the Time is the algorithm Running Time, and the Depth is the number of nodes visited by each algorithm to find the solution.

Figure 4.2: Path Planning Results for the Traditional Vertical Layout

| Breadth-First Search |              |          |               |                  |       |
|----------------------|--------------|----------|---------------|------------------|-------|
| Test                 | Interactions |          | Time          |                  | Depth |
|                      | While Loop   | For Loop | Wall Time (s) | Process Time (s) |       |
| 1                    | 34           | 3        | 0.0009        | 0.0035           | 9     |
| 2                    | 28           | 3        | 0.0012        | 0.0012           | 7     |
| 3                    | 21           | 3        | 0.0010        | 0.0042           | 5     |
| 4                    | 42           | 2        | 0.0012        | 0.0035           | 14    |
| 5                    | 53           | 2        | 0.0013        | 0.0013           | 12    |
| 6                    | 42           | 2        | 0.0007        | 0.0007           | 10    |
| 7                    | 42           | 2        | 0.0011        | 0.0045           | 9     |
| 8                    | 54           | 3        | 0.0013        | 0.0013           | 11    |
| 9                    | 54           | 3        | 0.0009        | 0.0015           | 13    |

| Dijkstra's Algorithm |              |          |               |                  |       |
|----------------------|--------------|----------|---------------|------------------|-------|
| Test                 | Interactions |          | Time          |                  | Depth |
|                      | While Loop   | For Loop | Wall Time (s) | Process Time (s) |       |
| 1                    | 47           | 4        | 0.0015        | 0.0055           | 9     |
| 2                    | 42           | 2        | 0.0014        | 0.0058           | 7     |
| 3                    | 36           | 2        | 0.0006        | 0.0006           | 5     |
| 4                    | 54           | 2        | 0.0008        | 0.0008           | 14    |
| 5                    | 54           | 2        | 0.0021        | 0.0047           | 12    |
| 6                    | 55           | 3        | 0.0008        | 0.0030           | 10    |
| 7                    | 31           | 3        | 0.0013        | 0.0021           | 9     |
| 8                    | 41           | 4        | 0.0007        | 0.0007           | 11    |
| 9                    | 51           | 2        | 0.0008        | 0.0008           | 13    |

| A-Star Algorithm |              |          |               |                  |       |
|------------------|--------------|----------|---------------|------------------|-------|
| Test             | Interactions |          | Time          |                  | Depth |
|                  | While Loop   | For Loop | Wall Time (s) | Process Time (s) |       |
| 1                | 35           | 2        | 0.0007        | 0.0016           | 9     |
| 2                | 23           | 4        | 0.0004        | 0.00045          | 7     |
| 3                | 13           | 4        | 0.0003        | 0.0003           | 5     |
| 4                | 20           | 2        | 0.0009        | 0.0009           | 14    |
| 5                | 20           | 2        | 0.0011        | 0.0011           | 12    |
| 6                | 20           | 2        | 0.0004        | 0.0004           | 10    |
| 7                | 10           | 3        | 0.0013        | 0.0011           | 9     |
| 8                | 16           | 3        | 0.0004        | 0.0004           | 11    |
| 9                | 22           | 3        | 0.0004        | 0.0004           | 13    |

It is possible to notice that the A\* was the one with less running time than the others even having the same Depth, leading advantage also in the number of interactions in all tests. Meanwhile, Dijkstra's Algorithm fared similarly to Breadth-First Search but had to search a greater number of nodes to find the shortest path as seen in the Figure below. Regarding space complexity all algorithms showed similar memory usage ( $\sim 75$  MB), one of the reasons may be the reduced size of the environment, resulting in a low memory requirement during search operations.

### Flying-V Warehouse Layout

A total of 9 tests were performed for each of the algorithms using the simulator proposed, where the location of the goal and start position were modified for each test. In Figure 4.3 is possible to review the results of the experimental evaluation where the Interactions are the number of interactions needed to find the solution, the Time is the algorithm Running Time, and the Depth is the number of nodes visited by each algorithm to find the solution.

Figure 4.3: Path Planning Results for the Flying-V Layout

| Breadth-First Search |              |          |               |                  |       |
|----------------------|--------------|----------|---------------|------------------|-------|
| Test                 | Interactions |          | Time          |                  | Depth |
|                      | While Loop   | For Loop | Wall Time (s) | Process Time (s) |       |
| 1                    | 30           | 3        | 0.0013        | 0.0033           | 9     |
| 2                    | 19           | 3        | 0.0006        | 0.0006           | 7     |
| 3                    | 45           | 3        | 0.0018        | 0.0056           | 5     |
| 4                    | 52           | 3        | 0.0020        | 0.0041           | 14    |
| 5                    | 45           | 3        | 0.0008        | 0.0008           | 12    |
| 6                    | 51           | 2        | 0.0008        | 0.0008           | 10    |
| 7                    | 57           | 2        | 0.0015        | 0.0048           | 15    |
| 8                    | 57           | 2        | 0.0013        | 0.0043           | 13    |
| 9                    | 51           | 2        | 0.0015        | 0.0015           | 11    |

| Dijkstra's Algorithm |              |          |               |                  |       |
|----------------------|--------------|----------|---------------|------------------|-------|
| Test                 | Interactions |          | Time          |                  | Depth |
|                      | While Loop   | For Loop | Wall Time (s) | Process Time (s) |       |
| 1                    | 46           | 2        | 0.0007        | 0.0007           | 9     |
| 2                    | 46           | 2        | 0.0006        | 0.0006           | 7     |
| 3                    | 39           | 2        | 0.0007        | 0.0048           | 5     |
| 4                    | 57           | 3        | 0.0017        | 0.0038           | 14    |
| 5                    | 57           | 3        | 0.0008        | 0.0008           | 12    |
| 6                    | 58           | 3        | 0.0026        | 0.0066           | 10    |
| 7                    | 46           | 2        | 0.0011        | 0.0035           | 15    |
| 8                    | 54           | 2        | 0.0008        | 0.0008           | 13    |
| 9                    | 46           | 2        | 0.0007        | 0.0007           | 11    |

| A-Star Algorithm |              |          |               |                  |       |
|------------------|--------------|----------|---------------|------------------|-------|
| Test             | Interactions |          | Time          |                  | Depth |
|                  | While Loop   | For Loop | Wall Time (s) | Process Time (s) |       |
| 1                | 31           | 2        | 0.0005        | 0.0027           | 9     |
| 2                | 25           | 2        | 0.0007        | 0.0009           | 7     |
| 3                | 13           | 2        | 0.0003        | 0.0011           | 5     |
| 4                | 19           | 2        | 0.0007        | 0.0007           | 14    |
| 5                | 34           | 2        | 0.0008        | 0.0008           | 12    |
| 6                | 19           | 2        | 0.0004        | 0.0004           | 10    |
| 7                | 31           | 3        | 0.0006        | 0.0006           | 15    |
| 8                | 25           | 3        | 0.0010        | 0.0010           | 13    |
| 9                | 19           | 3        | 0.0008        | 0.0026           | 11    |

It is possible to notice that the A\* was the one with less running time again. Meanwhile, Dijkstra's Algorithm fared similarly to Breadth-First Search but had to search a greater number of nodes to find the shortest path as seen in the Figure below. Regarding space complexity, all algorithms showed similar memory usage ( $\sim 75$  MB), one of the reasons may be the reduced size of the environment, resulting in a low memory requirement during search operations.

### Final Considerations

Through the tests, it is possible to see that the simulator has achieved the expected result for the Single Robot mode, where in all tests it was possible to find a way to the goal. The analysis also makes it clear that Dijkstra's algorithm is not feasible for the RFMS or even the MAPF problem since for being greedy it will perform a larger search before finding the shortest path. The A\* was the one that best performed, justifying the choice of this algorithm for the implementation in the MAPF solutions. The result differences between Breadth-First



and A-Star for the various types of Warehouse are more linked to the running time than to node search since the nodes searched by the Breadth-First search to find the shortest path came relatively close to A-Star, however, the algorithm had a much higher running time, making it unfeasible for the real RFMS scenario since the environment will be much larger than the one performed in these tests.

## 4.2 The Space-Time Swarm Path-Finder (ST-SPF) Algorithm

This chapter aims to present the Space-Time Swarm Path Finding (ST-SPF) proposal based on the concept of Swarms. Some rules will be defined for creating or appending the *Swarms* in the environment, as well as the high-level implementation of the algorithm and the reason for choosing low-level algorithms.

### 4.2.1 The ST-SPF Swarms

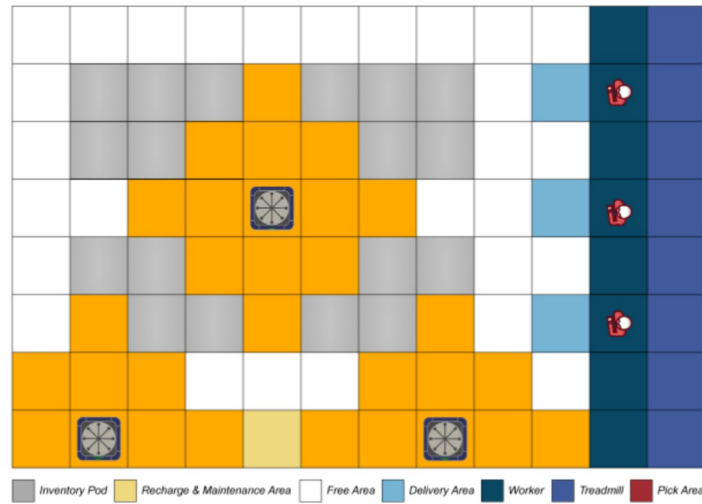
Firstly, let's define a **Swarm** as the joining of multiple agents with reduced intelligence, but that together can exhibit intelligent behavior. One of the main characteristics of the MAPF becoming so computationally heavy is the fact that it performs the pathfinding for all the robots in the environment. Our main hypothesis is that since the ST-SPF will distribute the planning by groups, the computational effort will be reduced since each group has a fewer number of agents, increasing the scalability as we bring more complexity, bigger instance sizes, and more agents. Considering the Definitions 2.1.3 and 2.1.3 that avoid conflicts in MAPF, we can make the following definitions for swarms creations or appending:

**Definition 4.2.1.** (Swarm Creation) A new swarm can only be created iff an agent is isolated and is not inside another agent's zone, where the swarm zone created must have the maximum size of the nodes around the agent plus a node cell for each direction where the movement is possible.

**Definition 4.2.2.** (Swarm Appending) If the agent is inside another agent zone, the previous Swarm will be appended and the robot will be inserted in this new group, where the Swarm Zone will become the sum of all swarms zones of the agents.

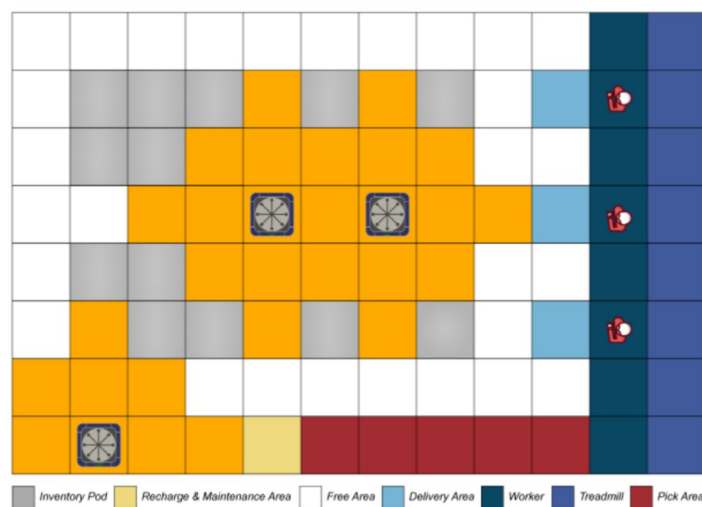
Considering the Definition 4.2.1, all agents when started in the environment will look for another agent in their vicinity looking at two nodes (or grid spaces) on the left, two on the right, two on the top, two on the bottom, and one on each diagonal. An example of the Swarm zone can be seen in Figure 4.4, where three robots are used in different environment locations.

Figure 4.4: The Swarm Zone Creation for Isolated Agents



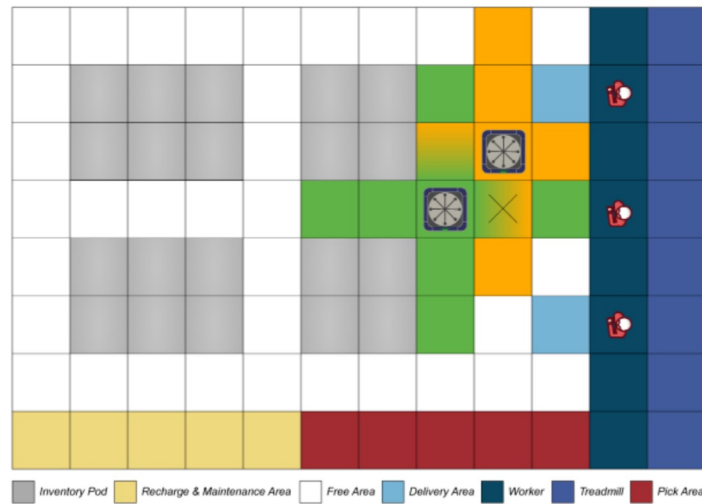
Considering the Definition 4.2.1, if an agent is started within a zone of some other agent, it must be inserted in the swarm of the previous one and increase the new zone of the swarm, this being the sum of the zones of the two agents. An example of the Swarm zone appending can be seen in Figure 4.5, where three robots are used in different environment locations.

Figure 4.5: The Swarm Zone Appending



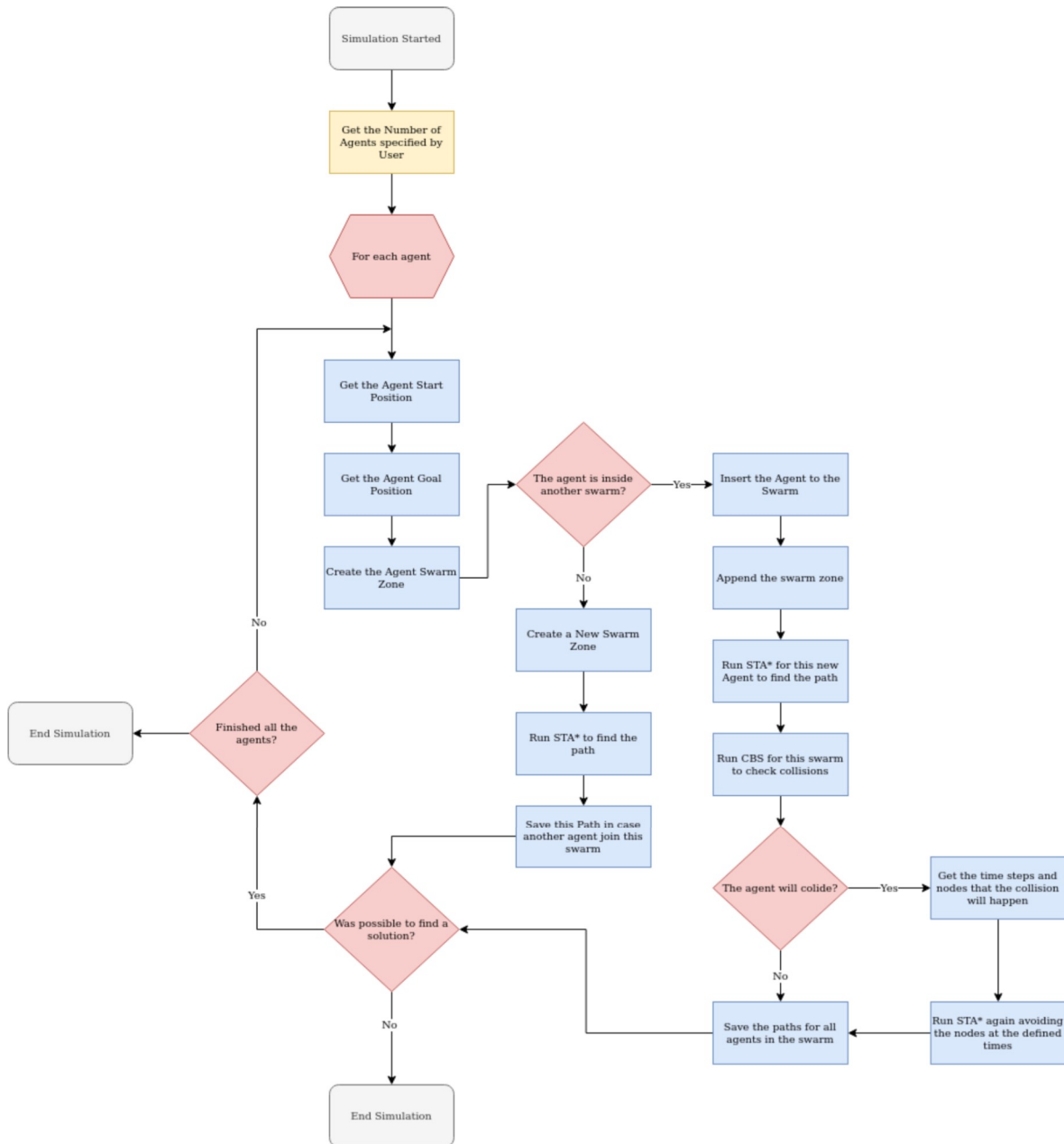
The need to include the diagonal zones to the agent swarm zone is that this prevents the agents from being in blind spaces, where Edge, Vertex or Cycle conflict can happen. In Figure 4.6 is possible to see an example of this problem if we don't use the diagonal zones for the agents. Considering that the "green" agent is going to right and the "orange" to bottom, we will probably have a collision at the next time-step.

Figure 4.6: Avoiding Blind Spots and Conflicts



The swarm's implementation in the simulator is done as the agents are added to the map. For the multi-robot simulation, the simulator starts one agent at a time in the location specified by the user at the *setting* file, followed by the function *create\_swarms\_area*, with is responsible for verify if there is already a zone present at the robot start location. After that, the path is calculated considering the free space available and also if the agent is isolated or in another swarm zone. Considering this rules, the ST-SPF algorithm principle of working can be seen at Figure 4.7, were code implementation can be seen in [8].

Figure 4.7: The ST-SPF Algorithm



For the ST-SPF implementation, as shown in the figure, two algorithms from the state-of-the-art were used: the Conflict Based-Search (CBS) and the Space-Time A\* (STA\*, also known as Cooperative A\*). In summary, the STA\* is used for the low-level search, while CBS is used to detect collisions between the paths found by STA\*. But how does STA\* work? The only difference from this to the traditional A\* is that it adds the concept of time, that is, instead of a map with a traditional location (X, Y), we will have a third variable (X, Y, t), being  $t$  the time. This allows algorithms like CBS to detect the location of the agent over

time and perform collision control. One of the reasons for choosing this specific algorithm is that A\* was the one that brought the best results for all types of layouts. The algorithm implementation is also simpler since it is only necessary to add one more variable to the traditional A\*. More details about STA \* can be viewed at [59].

The reason for choosing CBS is due to its extensive research in recent years and easy implementation. In summary, CBS is a high-level algorithm that detects a collision and adds it to a constraint tree, allowing the agents to avoid this location at the respective time  $t$ . More information about this algorithm can be found at cite Sharon2015CBS.

In this way, the ST-SPF can be considered as a high-level offline and decentralized algorithm, where through the concepts and rules of swarm represented previously it allows to execute the trajectory planning of multiple agents and the MAPF problem in a *distributed* way. Despite the choices of the aforementioned algorithms, any other algorithm could be used at a low-level and high-level, bringing more flexibility in its use.

#### 4.2.2 Discussion

One of the main drawbacks of this algorithm is that collision control is only performed within a swarm group with more than one agent, thus enabling the collision of an agent outside this group. However, because it has the characteristic of being decentralized, this problem can be easily solved by implementing the algorithm in an "online" environment, where agents check and update their location over time and can request entry into a swarm. In this way, we are replicating the same swarm appending process of the algorithm, thus avoiding future collisions and making the algorithm fail-proof.

One of the reasons for not implementing this add-on is that it would become relatively complicated to do this process through the PyGame library, in addition to the fact that CBS does not perform this type of online analysis since its execution is done one-time (since it is centralized). In this way, it was chosen to leave this implementation out of this work, since the online position detection method can be performed in several possible ways. Another detail that needs to be clarified is that during the comparative tests of CBS and ST-SPF in the Grid-Like environment, it was noticed that ST-SPF only achieved good running time results from a considerable number of robots. Thus the question arose: **How to reduce running time for a few robots?**.

We created a Graph-Like environment to answer this question with the hypothesis that since it has a simpler structure than a Grid-Like environment, we would considerably decrease the search, creation, and appending of the swarm algorithm. To also bring more complexity to this environment and understand how the algorithms will scale, the following restrictions (restrictions found in MAPF, as in 2.7) also added to the Graph-Like STA\* algorithm:

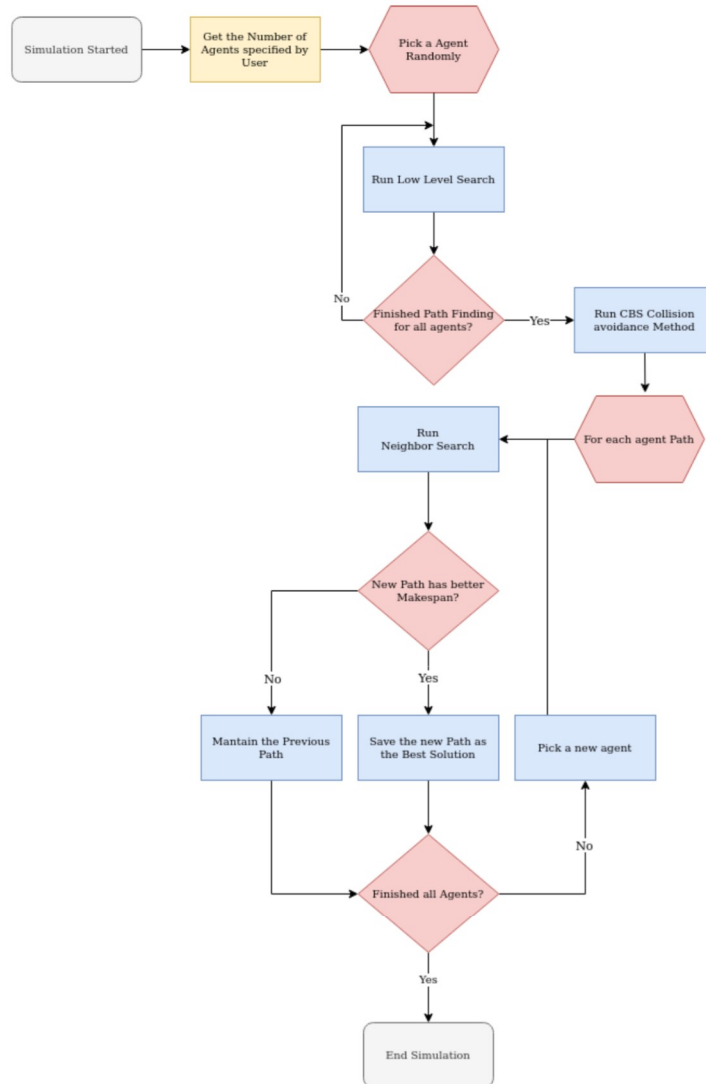
- Avoid Vertex Conflict;
- Avoid Edge Conflict;
- Avoid Following Conflict;
- Avoid Cycle Conflict;
- Avoid Swapping Conflict;

These changes allow a comparison to be made between a Grid-Like and Graph-Like environment, in addition to bringing a sense of which type of environment allows better scalability since if the Graph-Like environment has better results than the Grid-Like even with all retractions but not scales well, the Grid-Like will scale as well since have a heavier data structure.

### 4.3 The Space-Time Multi-Start Algorithm (STMS)

As explained earlier, CBS has an excellent performance in small environments, and during experimental evaluation the difference in performance of the ST-SPF in relation to the CBS in small spaces and with a reduced number of robots was notable. Thus the following question arose: **How to get better results in small and less crowded spaces maintaining collision control but bringing an advantage to the system?** To answer this question, Space-Time Multi-Start (STMS) was implemented, an centralized algorithm with focus on reducing the Makespan value. Unlike the ST-SPF, the STMS algorithm solves the path conflicts for all agents through three pillars: Randomness, Collision Avoidance, and Neighbors Search. An explanation of these definitions can be found below, while the algorithm flowchart can be seen in Figure 4.8.

Figure 4.8: The STMS Algorithm



**Definition 4.3.1.** (Randomness) To calculate the paths by the Low-Level Algorithm, each agent must be start randomly instead of following a certain order, but respecting the desired goal and start positions;

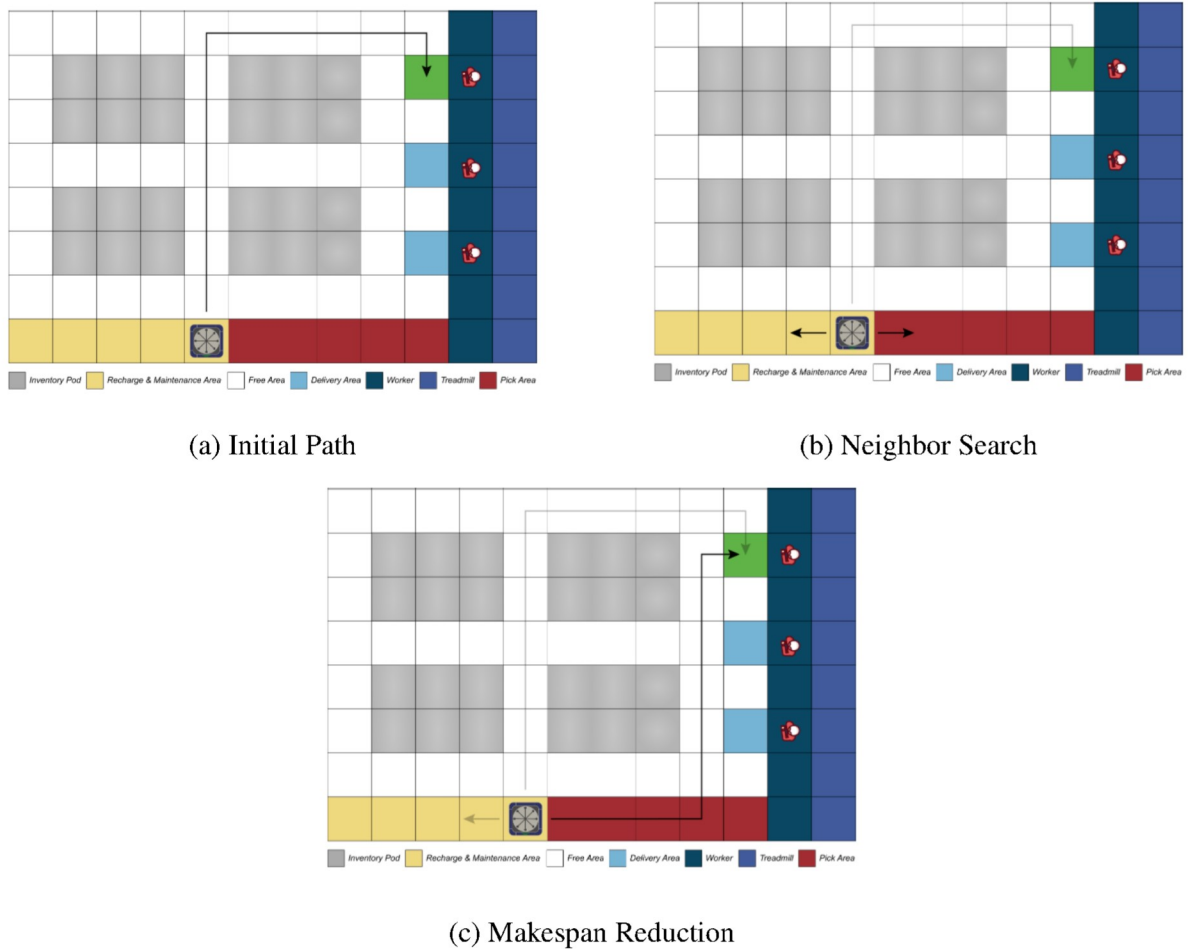
**Definition 4.3.2.** (CBS Collision Avoidance) Considering two agents A and B with position variant in time ( $t$ ), if the path of these agents has a conflict at  $t + n$ , the conflicting node must be placed as a constraint and the path of one of the agents must be recalculated to avoid this node.

**Definition 4.3.3.** (Neighbors Search) After finding a valid path through the low-level algorithm (initial solution), the agent must look at their neighboring nodes and calculate a new

path, if this new path is different from the initial solution and has a smaller Makespan this path should be considered as the best solution.

Let's approach these definitions with a practical example, considering the Definition 4.3, each agent is started randomly, and its path is resolved by the low-level algorithm. Assuming that the chosen agent was located at the recharge point and needed to be sent to one of the delivery points, after the execution of the low-level algorithm it was defined that the best path would be the black arrow as seen in Figure 4.9a. Since we are working with multiple agents we need to verify if the current path has any conflict with some other agent (Definition 4.3). Considering that there were no conflicts and the path remained the same, the following question can be made: **Would it be possible to have a better solution than the initial solution proposed by the low-level algorithm?**

Figure 4.9: Space-Time Multi-Start Algorithm





The answer to this question can be found by implementing the Definition 4.3 (Figure 4.9b). The agent can look at its neighboring nodes, and considering the initial path chosen as a constraint, it will look for a new solution starting from its neighbors. If he finds a solution that has a lower Makespan than the initial solution this new solution is the best solution (see Figure 4.9c). Like the ST-SPF, the STMS can be considered a high-level offline algorithm, since it is possible to modify the internal algorithms, however, it has the characteristic of being centralized like the CBS since it performs the Collision Avoidance based on the paths of all agents.

### 4.3.1 Discussion

One of the biggest drawbacks of this algorithm is the considerable increase in running time since we are calculating  $n$  new paths for each agent, focusing on achieve best Makespan value. Like the ST-SPF and CBS for the Graph-Like environment, the following restrictions (restrictions found in MAPF, as in 2.7) were also added to the algorithm:

- Avoid Vertex Conflict;
- Avoid Edge Conflict;
- Avoid Following Conflict;
- Avoid Cycle Conflict;
- Avoid Swapping Conflict;

The reason the choice behind reducing the makespan (and consequently the Sum of Costs) is that with a reduction in the number of robot movements in the environment, we considerably increase the possibility of solving more paths in crowd environments (since we will have more free spaces). Also, considering a real implementation, this would reduce the battery consumption of the robots in addition to the mechanical efforts, increasing the system life-cycle.

Finally, two algorithms were selected to perform the low-level search, the A-star ( $A^*$ ) and the Breadth-First Search (BFS). Thus, in the results two STMS algorithms will be analyzed, the STMS  $A^*$ -Based (STMS  $A^*$ ) and the STMS BFS-Based (STMS BFS). The reason to

include the BFS is that the algorithm will find the shortest path between two vertices instead of prioritizing the paths with the smallest weight (as in A\*). This feature is important since we want to reduce the Makespan as much as possible and will optimize the implementation of Definition 4.3. It is also worth mentioning that both algorithms needed to be implemented considering the time, so the correct nomenclature would be Space-Time A\* and Space-Time BFS, but for simplification, we will use the nomenclature used at the beginning of the paragraph.

# Chapter 5

## Experimental Evaluation

In this chapter, the proposed algorithms results will be presented. Despite the definition of the maximum number of agents, tests were executed in the tiny and large environments exceeding these limits to identify the robustness and scalability of the algorithms in populous and complex spaces. The reason for choosing only these sizes for robustness testing is that CBS performs best with a reduced number of agents (*no crowd*) in small and large spaces. Thus, for robustness analysis criteria, 15 robots were used in the Tiny environment and 250 in the Large environment. Is worth mentioning that this section will also cover the simulator's ability to run the multi-agent simulations in multiple warehouse layouts.

The chapter is divided as follows: In Topic 5.1, the algorithms results and discussion are presented regarding the Traditional Horizontal Layout using the Tiny, Small, Medium, and Large sizes. In Topic 5.2, the algorithms results and discussion are presented regarding the Traditional Vertical Layout using the Tiny, Small, Medium, and Large sizes. In Topic 5.3, the algorithms results and discussion are presented regarding the Traditional Horizontal Layout using the Tiny, Small, Medium, and Large sizes. Finally, in Topic an brief discussion is made to resume the results and discuss possible algorithms advantages and disadvantages.

The tests were run on a Notebook *ASUS Expert X23*, with an Intel I5 7200U 2.50 GHz Processor, an NVIDIA 920MX GPU, 8 GB DDR4 SDRAM, and an Ubuntu 18.04 operating system with PyGame v1.9.6 and Python 3.6. Before showing the algorithms results for different Warehouses Layouts, the following considerations need to be highlighted:

- The agents can perform movements in four directions: Top, Bottom, Left, and Right;

- The agents can perform only one movement per time step;
- The comparative tests between the algorithms have the same start and goal point for each layout;
- The Running Time measurement was done through the *timer* built-in function;
- The Warehouses Layouts Design are based on the layouts proposed in the state-of-the-art, as shown in Figures 2.11 and 2.12;
- The Makespan is the maximum number of movements (time-steps) done by an agent to reach the goal;
- The Sum of Costs is the sum of all agents movements (time-steps) to reach the goals;
- Five rounds of tests were performed for each robot number and each algorithm, so the values in the tables are the average;
- The algorithms can't exceed the 8 GB limit for memory usage, if this happens we declare as *memory overflow*;
- The maximum time limit defined to find a solution is 500 s;
- The green items in the tables are the best results for a evaluation criteria (Running Time, Makespan, Sum of Costs), while the red items are tests that exceeded the time limit.

## 5.1 Traditional Horizontal

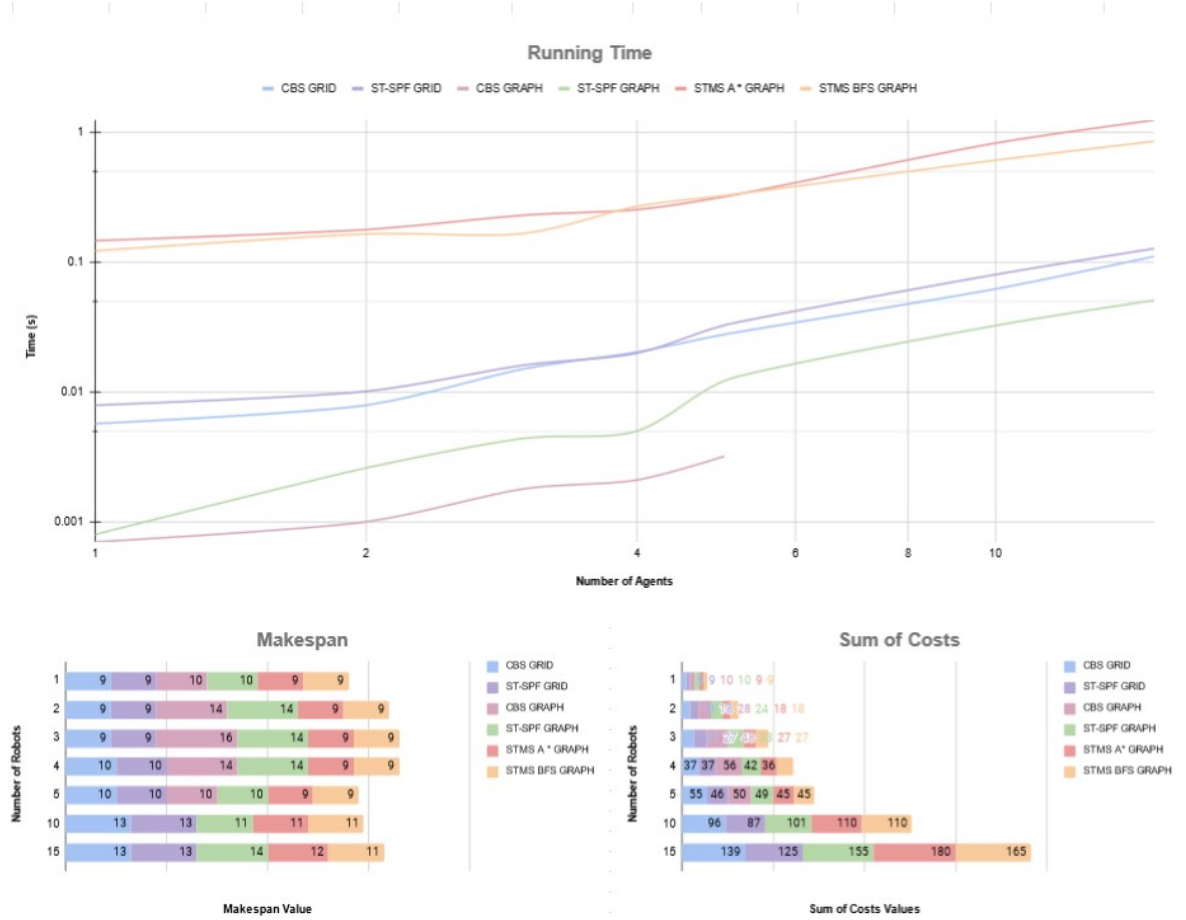
### 5.1.1 Tiny Layout

The Tiny environment has 18 x 8 (144 nodes/grid points) in size, however with the limitation of 128 nodes/grid points free for movement (128 states) due to the location of the Workers and Treadmill zones. Considering the minimum number of robots is 1 and the maximum is 10 since it has 9 pickup and recharge points. However, since the CBS works better in small spaces, we increased the maximum number of robots to 15 to compare which algorithm have more scalability in complex and crowded environment.

Table 5.1: Tiny Traditional Horizontal Results

| Number of Agents | Running Time (s) |        |        |            |           |            | Makespan  |        |     |            |           |            | Sum of Costs |        |     |            |           |            | Winner Algorithm (by Turn) |              |                |                                  |
|------------------|------------------|--------|--------|------------|-----------|------------|-----------|--------|-----|------------|-----------|------------|--------------|--------|-----|------------|-----------|------------|----------------------------|--------------|----------------|----------------------------------|
|                  | Grid-Like        |        |        | Graph-Like |           |            | Grid-Like |        |     | Graph-Like |           |            | Grid-Like    |        |     | Graph-Like |           |            | Running Time               | Makespan     | Sum of Costs   | Winner Algorithm (by Comparison) |
|                  | CBS              | ST-SPF | CBS    | ST-SPF     | STMS (A*) | STMS (BFS) | CBS       | ST-SPF | CBS | ST-SPF     | STMS (A*) | STMS (BFS) | CBS          | ST-SPF | CBS | ST-SPF     | STMS (A*) | STMS (BFS) |                            |              |                |                                  |
| 1                | 0.0057           | 0.0079 | 0.0007 | 0.0008     | 0.1459    | 0.1219     | 9         | 9      | 10  | 10         | 9         | 9          | 9            | 9      | 10  | 10         | 9         | 9          | CBS (Graph)                | —            | —              | —                                |
| 2                | 0.0079           | 0.0101 | 0.0010 | 0.0026     | 0.1776    | 0.1642     | 9         | 9      | 14  | 14         | 9         | 9          | 18           | 18     | 28  | 24         | 18        | 18         | CBS (Graph)                | —            | —              | —                                |
| 3                | 0.0151           | 0.0161 | 0.0018 | 0.0044     | 0.2293    | 0.1662     | 9         | 9      | 16  | 14         | 9         | 9          | 27           | 27     | 48  | 33         | 27        | 27         | CBS (Graph)                | —            | —              | —                                |
| 4                | 0.0203           | 0.0199 | 0.0021 | 0.0050     | 0.2531    | 0.2682     | 10        | 10     | 14  | 14         | 9         | 9          | 37           | 37     | 56  | 42         | 36        | 36         | CBS (Graph)                | STMS (Graph) | STMS (Graph)   | STMS (Graph)                     |
| 5                | 0.0277           | 0.0326 | 0.0032 | 0.0122     | 0.3180    | 0.3246     | 10        | 10     | 10  | 10         | 9         | 9          | 55           | 46     | 50  | 49         | 45        | 45         | CBS (Graph)                | STMS (Graph) | STMS (Graph)   | STMS (Graph)                     |
| 10               | 0.0620           | 0.0800 | —      | 0.0324     | 0.8222    | 0.6068     | 13        | 13     | —   | 11         | 11        | 11         | 96           | 87     | —   | 101        | 110       | 110        | ST-SPF (Graph)             | —            | ST-SPF (Graph) | ST-SPF (Graph)                   |
| 15               | 0.1103           | 0.1267 | —      | 0.0507     | 1.2339    | 0.8484     | 13        | 13     | —   | 14         | 12        | 11         | 139          | 125    | —   | 155        | 180       | 165        | ST-SPF (Graph)             | STMS (Graph) | ST-SPF (Graph) | ST-SPF (Graph)                   |

Figure 5.1: Tiny Traditional Horizontal Graph



As seen in Figure 5.1 and Table 5.1 the swarm algorithm of the ST-SPF brought a Running Time increase of approximately 30% for just one agent in the Grid-Like environment. This indicates that the swarm algorithm is responsible for this increase, however, the algorithm remained within the same time range as CBS ( $<0.01$  s). As we scale the number of agents, it is possible to see that ST-SPF maintains the same Running Time range as CBS, being able to win with 2 and 4 robots, providing a gain of 8% and 2% respectively. The ST-SPF also reduced the sum of costs by approximately 16%, 9%, and 10% for 5, 10, and

15 robots respectively.

Regarding the Graph-Like environment, CBS was the one that had the best result considering the time of execution, since the ST-SPF brought an additional gain of approximately 13% due to the swarm algorithm. This considerable reduction in the Running Time compared to the Grid-Like environment is because the data structure in the Graph-Like environment is already saved in memory as an integer type instead of an (X, Y) position. The STMS has a relatively longer execution time than the other two, but this is already expected considering that it is a Metaheuristic that will loop until it finds the best possible solution. Despite the time, STMS (A\*-Based & BFS-Based) stood out in the reduction of Makespan and Sum of Costs when compared to the other algorithms (Grid & Graph-like), reaching a gain of up to 56% when compared to CBS (Graph-Like), but falling behind ST-SPF after 10 robots. Comparing the two STMS algorithms, it is possible to notice that the BFS-Based brought up to 46 % time-complexity reduction when compared to the A\*-Based, becoming more viable for small and populous spaces using this particular layout.

Finally, a very interesting point to be highlighted in this environment is that CBS was unable to find a solution for the Graph-Like environment after 8 agents. This happens because of the additional MAPF and Collision rules that were implemented for this specific environment. Is worth mentioning that even though CBS failed to find the solution, the ST-SPF and both STMSs found a solution. This happens because the ST-SPF is not concerned with collisions between agents outside of swarms blocks, and the STMSs reduced the makespan and sum of costs, freeing up then spaces in the environment.

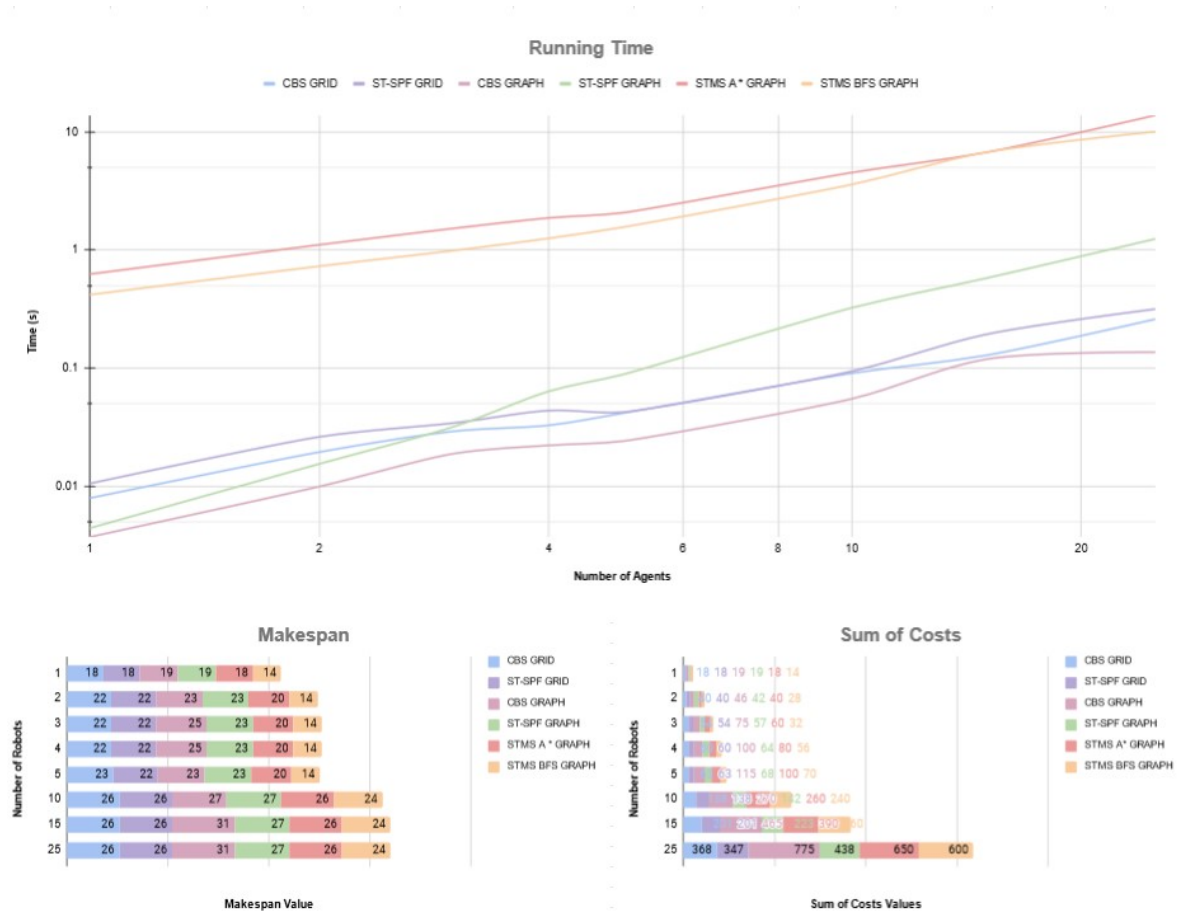
### 5.1.2 Small Layout

The Small environment has 28 x 10 (280 nodes/grid points) in size, however with the limitation of 260 nodes/grid points free for movement (128 states) due to the location of the Workers and Treadmill zones. Considering the minimum number of robots is 1 and the maximum is 25 since it has 24 pickup and recharge points.

Table 5.2: Small Traditional Horizontal Results

| Number of Agents | Running Time (s) |        |            |        |           |            | Makespan  |        |            |        |           |            | Sum of Costs |        |            |        |           |            | Winner Algorithm (by Turn) |              |              | Winner Algorithm (by Comparison) |
|------------------|------------------|--------|------------|--------|-----------|------------|-----------|--------|------------|--------|-----------|------------|--------------|--------|------------|--------|-----------|------------|----------------------------|--------------|--------------|----------------------------------|
|                  | Grid-Like        |        | Graph-Like |        |           |            | Grid-Like |        | Graph-Like |        |           |            | Grid-Like    |        | Graph-Like |        |           |            | Running Time (s)           | Makespan     | Sum of Costs |                                  |
|                  | CBS              | ST-SPF | CBS        | ST-SPF | STMS (A*) | STMS (BFS) | CBS       | ST-SPF | CBS        | ST-SPF | STMS (A*) | STMS (BFS) | CBS          | ST-SPF | CBS        | ST-SPF | STMS (A*) | STMS (BFS) |                            |              |              |                                  |
| 1                | 0.0079           | 0.0105 | 0.0037     | 0.0044 | 0.6252    | 0.4183     | 18        | 18     | 19         | 19     | 18        | 14         | 18           | 18     | 19         | 19     | 18        | 14         | CBS (Graph)                | STMS (Graph) | STMS (Graph) | STMS (Graph)                     |
| 2                | 0.0194           | 0.0261 | 0.0099     | 0.0154 | 1.1078    | 0.7296     | 22        | 22     | 23         | 23     | 20        | 14         | 40           | 40     | 46         | 42     | 40        | 28         | CBS (Graph)                | STMS (Graph) | STMS (Graph) | STMS (Graph)                     |
| 3                | 0.0291           | 0.0342 | 0.0188     | 0.0318 | 1.5252    | 0.9899     | 22        | 22     | 25         | 23     | 20        | 14         | 54           | 54     | 75         | 57     | 60        | 32         | CBS (Graph)                | STMS (Graph) | STMS (Graph) | STMS (Graph)                     |
| 4                | 0.0327           | 0.0436 | 0.0221     | 0.0635 | 1.8708    | 1.2569     | 22        | 22     | 25         | 23     | 20        | 14         | 60           | 60     | 100        | 64     | 80        | 56         | CBS (Graph)                | STMS (Graph) | STMS (Graph) | STMS (Graph)                     |
| 5                | 0.0414           | 0.0422 | 0.0240     | 0.0880 | 2.0684    | 1.5602     | 23        | 22     | 23         | 23     | 20        | 14         | 63           | 63     | 115        | 68     | 100       | 70         | CBS (Graph)                | STMS (Graph) | STMS (Graph) | STMS (Graph)                     |
| 10               | 0.0904           | 0.0940 | 0.0549     | 0.3242 | 4.5372    | 3.5983     | 26        | 26     | 27         | 27     | 26        | 24         | 138          | 138    | 270        | 142    | 260       | 240        | CBS (Graph)                | STMS (Graph) | STMS (Graph) | STMS (Graph)                     |
| 15               | 0.1281           | 0.1929 | 0.1189     | 0.5773 | 6.7240    | 6.7405     | 26        | 26     | 31         | 27     | 26        | 24         | 201          | 201    | 465        | 223    | 390       | 360        | CBS (Graph)                | STMS (Graph) | STMS (Graph) | STMS (Graph)                     |
| 25               | 0.2595           | 0.3158 | 0.1363     | 1.2427 | 13.8617   | 10.0719    | 26        | 26     | 31         | 27     | 26        | 24         | 368          | 347    | 775        | 438    | 650       | 600        | CBS (Graph)                | STMS (Graph) | STMS (Graph) | STMS (Graph)                     |

Figure 5.2: Small Traditional Horizontal Graph



As seen in Figure 5.2 and Table 5.2, the ST-SPF swarm creation algorithm brought a Running Time increase of approximately 28% for just one agent for the Grid-Like environment but was worse in Running Time when compared to the Tiny Traditional layout since it left the CBS time range ( $> 0.01$  s). As we scale the number of agents, it is possible to see that ST-SPF was worse than CBS considering the Running Time but had a sum of costs reduction by 6% for 25 robots.

Regarding the Graph-Like environment, the CBS achieved the lowest Running Time even

when compared to the Grid-Like results, since the ST-SPF and STMS increased exponentially when the number of agents was increased, where the ST-SPF also brought an additional gain of approximately 16% due to the swarm algorithm. The STMSs still have a relatively longer execution time but stood out in the Makespan reduction when compared to the CBS and ST-SPF algorithms (Grid & Graph-like). Comparing the two versions of the STMS, it is possible to notice that the BFS-Based brought a reduction of time-complexity of up to 38% in relation to the A\*-Based for 25 agents, also achieving 29% reduction of the Makespan & Sum of Costs values when compared to the CBS (Graph-like) and STMS A\*-Based. For this specific layout, despite its high time-complexity value, the STMS BFS-Based brought satisfactory results, since it achieved better Makespan and Sum of Costs values in all scenarios even when the number of agents was increased.

### 5.1.3 Medium Layout

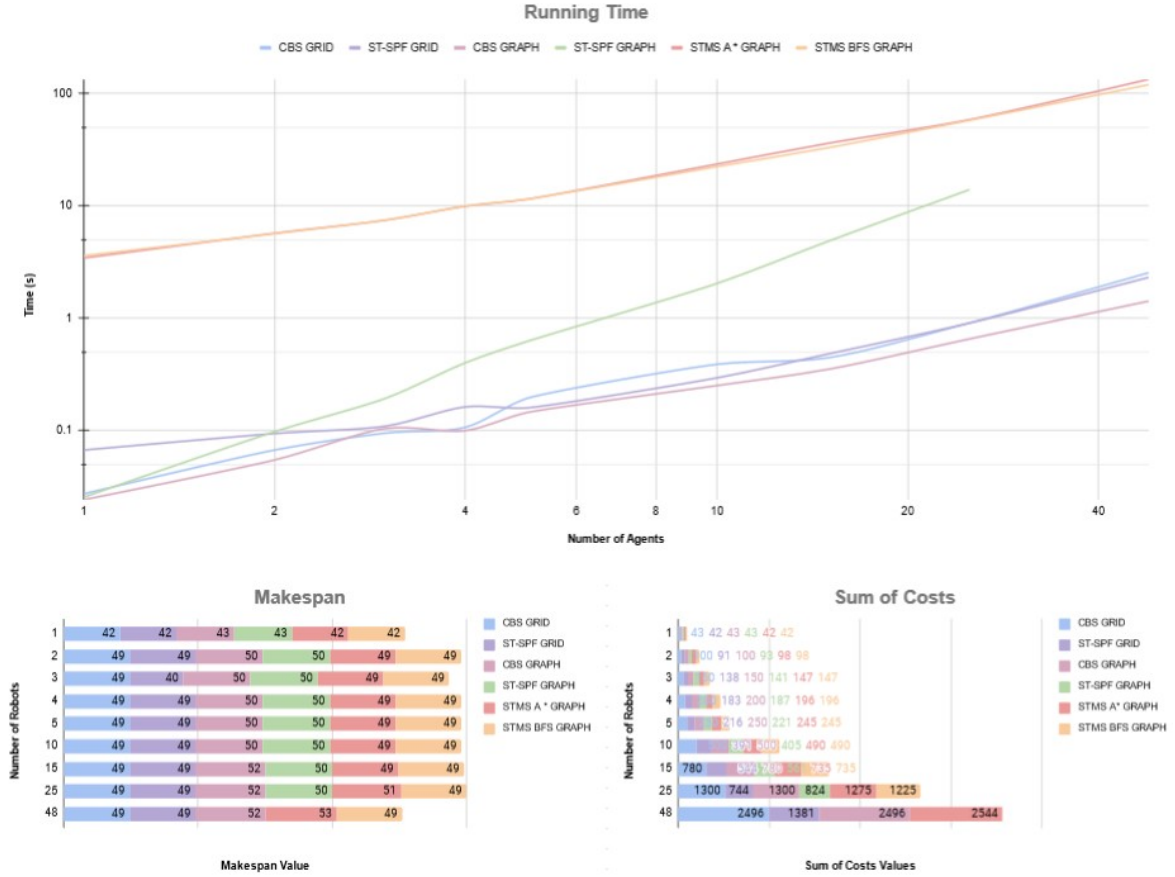
The Medium environment has 50 x 14 (700 nodes/grid points) in size, however with the limitation of 672 nodes/grid points free for movement (128 states) due to the location of the Workers and Treadmill zones. Considering the minimum number of robots is 1 and the maximum is 48 since it has 47 pickup and recharge points.

Table 5.3: Medium Traditional Horizontal Results

| Number of Agents | Running Time (s) |        |            |         |           | Makespan   |     |            |     |        | Sum of Costs |            |            |        |      | Winner Algorithm (by Turn) |          |              | Winner Algorithm<br>(by Comparison) |        |               |
|------------------|------------------|--------|------------|---------|-----------|------------|-----|------------|-----|--------|--------------|------------|------------|--------|------|----------------------------|----------|--------------|-------------------------------------|--------|---------------|
|                  | Grid-Like        |        | Graph-Like |         |           | Grid-Like  |     | Graph-Like |     |        | Grid-Like    |            | Graph-Like |        |      | Running Time (s)           | Makespan | Sum of Costs |                                     |        |               |
|                  | CBS              | ST-SPF | CBS        | ST-SPF  | STMS (A*) | STMS (BFS) | CBS | ST-SPF     | CBS | ST-SPF | STMS (A*)    | STMS (BFS) | CBS        | ST-SPF | CBS  |                            |          |              |                                     | ST-SPF | STMS (A*)     |
| 1                | 0.0275           | 0.0674 | 0.0244     | 0.0257  | 3.4391    | 3.6089     | 42  | 42         | 43  | 43     | 42           | 42         | 43         | 42     | 43   | 42                         | 42       | CBS (Graph)  | —                                   | —      | CBS           |
| 2                | 0.0674           | 0.0945 | 0.0550     | 0.0982  | 5.7109    | 5.6606     | 49  | 49         | 50  | 50     | 49           | 49         | 100        | 91     | 100  | 93                         | 98       | 98           | CBS (Graph)                         | —      | ST-SPF (Grid) |
| 3                | 0.0955           | 0.1101 | 0.1050     | 0.1952  | 7.4789    | 7.5031     | 49  | 40         | 50  | 50     | 49           | 49         | 150        | 138    | 150  | 141                        | 147      | 147          | CBS (Graph)                         | —      | ST-SPF (Grid) |
| 4                | 0.1072           | 0.1631 | 0.1002     | 0.4020  | 9.9233    | 9.9629     | 49  | 49         | 50  | 50     | 49           | 49         | 200        | 183    | 200  | 187                        | 196      | 196          | CBS (Graph)                         | —      | ST-SPF (Grid) |
| 5                | 0.1936           | 0.1597 | 0.1450     | 0.6213  | 11.4239   | 11.4508    | 49  | 49         | 50  | 50     | 49           | 49         | 250        | 216    | 250  | 221                        | 245      | 245          | CBS (Graph)                         | —      | ST-SPF (Grid) |
| 10               | 0.3902           | 0.2958 | 0.2527     | 2.0457  | 23.5316   | 22.3519    | 49  | 49         | 50  | 50     | 49           | 49         | 500        | 391    | 500  | 405                        | 490      | 490          | CBS (Graph)                         | —      | ST-SPF (Grid) |
| 15               | 0.4472           | 0.4833 | 0.3529     | 4.8635  | 36.0818   | 32.9674    | 49  | 49         | 52  | 50     | 49           | 49         | 780        | 544    | 780  | 561                        | 735      | 735          | CBS (Graph)                         | —      | ST-SPF (Grid) |
| 25               | 0.9082           | 0.9063 | 0.6550     | 13.9099 | 58.3748   | 57.5918    | 49  | 49         | 52  | 50     | 51           | 49         | 1300       | 744    | 1300 | 824                        | 1275     | 1225         | CBS (Graph)                         | —      | ST-SPF (Grid) |
| 48               | 2.5426           | 2.3128 | 1.4221     | —       | 133.5517  | 119.5090   | 49  | 49         | 52  | —      | 53           | 49         | 2496       | 1381   | 2496 | —                          | 2544     | 2352         | CBS (Graph)                         | —      | ST-SPF (Grid) |



Figure 5.3: Medium Traditional Horizontal Graph



As seen in Figure 5.3 and Table 5.3, the ST-SPF swarm creation algorithm brought a Running Time increase of approximately 84% for just one agent for the Grid-Like environment. However, as we scale the number of agents, it is possible to see that ST-SPF was better than CBS in Running Time for 5, 10, 25, and 48 agents. In this environment size, was possible to decrease by 27% the Running Time, and we also had better results for the ST-SPF sum of costs for almost all the number of agents, where the algorithm reduced the value by 10% for 48 robots.

Regarding the Graph-Like environment, the CBS achieved the lowest Running Time even when compared to the Grid-Like results, since the ST-SPF and STMS increased exponentially when the number of agents was increased. The ST-SPF also brought an additional gain of only 5% due to the swarm algorithm, becoming the best result when compared to the Tiny and Small sizes. Lastly, both STMS algorithms continues to increase the time-complexity, however the STMS BFS-Based still have better time-complexity values than the A\*-Based

(up to 12% difference).

In this specific size, it is possible to identify that the MAPF rules added to the algorithms in conjunction with the size of the environment started to affect the ST-SPF and STMS for the Graph-Like environment. The STMS appears to have a more negative impact since started to lose the ability to reduce the Makespan and Sum of Costs and considerably increase the Running Time. On the other hand, ST-SPF was unable to solve the problem from 48 robots due to memory overflow. One of the reasons that the algorithm has reached a high memory consumption since the algorithm implementation for the Graph-Like environment creates a class instance in memory with the values of the adjacency list, adjacency matrix, and swarm size for each robot or swarm.

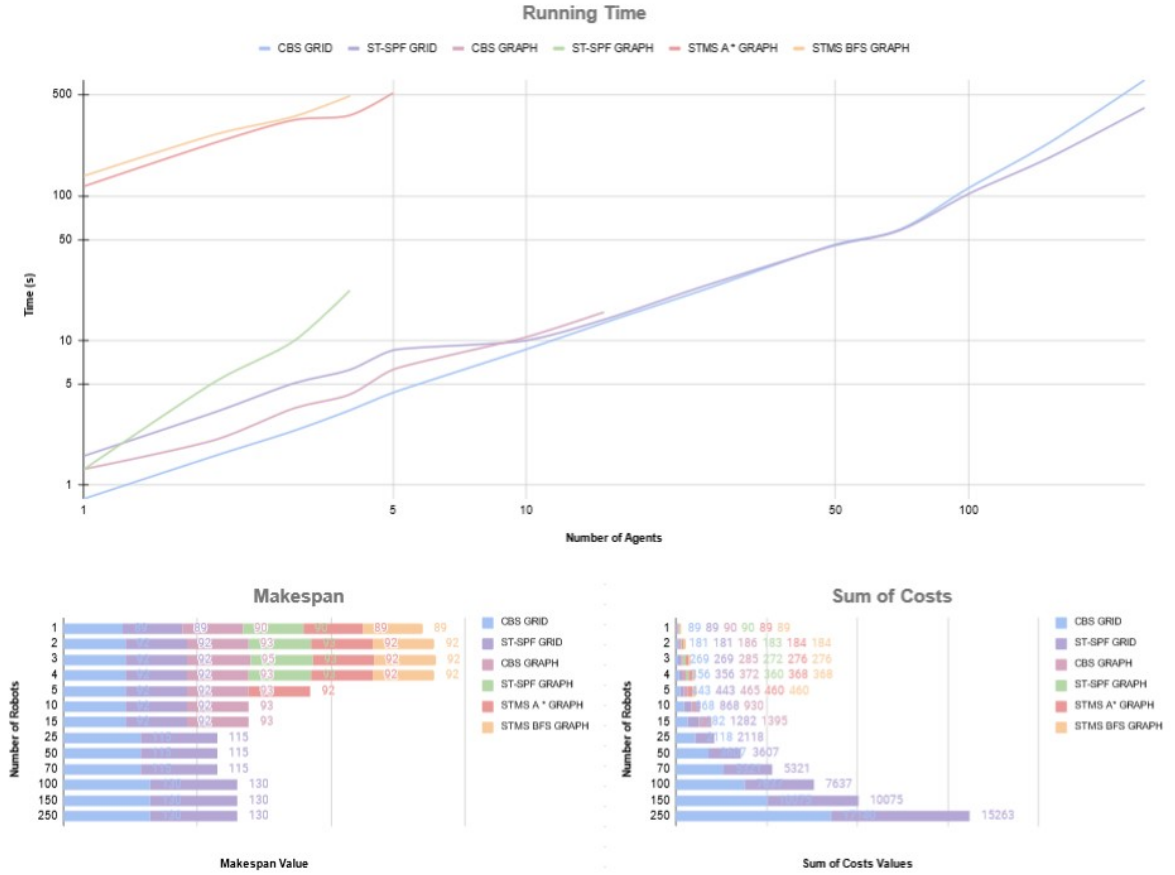
### 5.1.4 Large Layout

The Large environment has 100 x 50 (5000 nodes/grid points) in size, however with the limitation of 4900 nodes/grid points free for movement (128 states) due to the location of the Workers and Treadmill zones. Considering the minimum number of robots is 1 and the maximum is 98 since it has 97 pickup and recharge points. However, to test the complexity and scalability of the algorithms, the maximum number of robots was increased to 250.

Table 5.4: Large Traditional Horizontal Results

| Number of Agents | Running Time (s) |          |         |            |           |            | Makespan  |        |     |            |           |            | Sum of Costs |        |      |            |           |            | Winner Algorithm (by Turn) |          |               | Winner Algorithm<br>(by Comparison) |
|------------------|------------------|----------|---------|------------|-----------|------------|-----------|--------|-----|------------|-----------|------------|--------------|--------|------|------------|-----------|------------|----------------------------|----------|---------------|-------------------------------------|
|                  | Grid-Like        |          |         | Graph-Like |           |            | Grid-Like |        |     | Graph-Like |           |            | Grid-Like    |        |      | Graph-Like |           |            | Running Time (s)           | Makespan | Sum of Costs  |                                     |
|                  | CBS              | ST-SPF   | CBS     | ST-SPF     | STMS (A*) | STMS (BFS) | CBS       | ST-SPF | CBS | ST-SPF     | STMS (A*) | STMS (BFS) | CBS          | ST-SPF | CBS  | ST-SPF     | STMS (A*) | STMS (BFS) |                            |          |               |                                     |
| 1                | 0.8034           | 1.5891   | 1.2879  | 1.2821     | 116.4756  | 137.4416   | 89        | 89     | 90  | 90         | 89        | 89         | 89           | 89     | 90   | 90         | 89        | 89         | CBS (Grid)                 | —        | —             | CBS (Grid)                          |
| 2                | 1.6108           | 3.2402   | 2.0724  | 5.2760     | 235.9232  | 268.4970   | 92        | 92     | 93  | 93         | 92        | 92         | 181          | 181    | 186  | 183        | 184       | 184        | CBS (Grid)                 | —        | —             | CBS (Grid)                          |
| 3                | 2.3875           | 5.0833   | 3.4149  | 10.0257    | 336.3882  | 355.8104   | 92        | 92     | 95  | 93         | 92        | 92         | 269          | 269    | 285  | 272        | 276       | 276        | CBS (Grid)                 | —        | —             | CBS (Grid)                          |
| 4                | 3.3002           | 6.2885   | 4.2456  | 22.3273    | 361.6341  | 492.8279   | 92        | 92     | 93  | 93         | 92        | 92         | 356          | 356    | 372  | 360        | 368       | 368        | CBS (Grid)                 | —        | —             | CBS (Grid)                          |
| 5                | 4.3584           | 8.5693   | 6.2940  | —          | 515.6377  | —          | 92        | 92     | 93  | —          | 92        | —          | 443          | 443    | 465  | —          | 460       | 460        | CBS (Grid)                 | —        | —             | CBS (Grid)                          |
| 10               | 8.6963           | 10.0054  | 10.5594 | —          | —         | —          | 92        | 92     | 93  | —          | —         | —          | 868          | 868    | 930  | —          | —         | —          | CBS (Grid)                 | —        | —             | CBS (Grid)                          |
| 15               | 13.2858          | 13.9592  | 15.7366 | —          | —         | —          | 92        | 92     | 93  | —          | —         | —          | 1282         | 1282   | 1395 | —          | —         | —          | CBS (Grid)                 | —        | —             | CBS (Grid)                          |
| 25               | 22.2889          | 23.5798  | —       | —          | —         | —          | 115       | 115    | —   | —          | —         | —          | 2118         | 2118   | —    | —          | —         | —          | CBS (Grid)                 | —        | —             | CBS (Grid)                          |
| 50               | 46.3232          | 45.6554  | —       | —          | —         | —          | 115       | 115    | —   | —          | —         | —          | 3607         | 3607   | —    | —          | —         | —          | ST-SPF (Grid)              | —        | —             | ST-SPF (Grid)                       |
| 70               | 58.7702          | 57.9867  | —       | —          | —         | —          | 115       | 115    | —   | —          | —         | —          | 5321         | 5321   | —    | —          | —         | —          | ST-SPF (Grid)              | —        | —             | ST-SPF (Grid)                       |
| 100              | 113.6118         | 103.5556 | —       | —          | —         | —          | 130       | 130    | —   | —          | —         | —          | 7637         | 7637   | —    | —          | —         | —          | ST-SPF (Grid)              | —        | —             | ST-SPF (Grid)                       |
| 150              | 227.5796         | 180.2749 | —       | —          | —         | —          | 130       | 130    | —   | —          | —         | —          | 10075        | 10075  | —    | —          | —         | —          | ST-SPF (Grid)              | —        | —             | ST-SPF (Grid)                       |
| 250              | 635.4238         | 408.0578 | —       | —          | —         | —          | 130       | 130    | —   | —          | —         | —          | 17140        | 15263  | —    | —          | —         | —          | ST-SPF (Grid)              | —        | ST-SPF (Grid) | ST-SPF (Grid)                       |

Figure 5.4: Large Traditional Horizontal Graph



As seen in Figure 5.4 and Table 5.4, the ST-SPF swarm creation algorithm brought a Running Time increase of approximately 65% for just one agent for the Grid-Like environment. However, as we scale the number of agents, it is possible to see that ST-SPF was better than CBS in Running Time from 50 to 250 agents. In this environment size, was possible to decrease by 43% the Running Time, and we also had better results for the ST-SPF sum of costs, where the algorithm reduced the value by 10% for 250 robots. This result proves our previous hypothesis that the ST-SPF could bring more scalability as we increase the number of robots in instances with a considerable size. This happens because instead of calculating the collision for all the agents like CBS, we are clustering the problem in small swarms and making this calculation in a "decentralized" way. Finally, it is worth noting that CBS has also exceeded the maximum simulation time for 250 robots, achieving 135s above the limit.

Regarding the Graph-Like environment, neither the CBS, STMS (A\*-Based and BFS-Based), or ST-SPF performed well. As in the Medium size, it is possible to identify that

the MAPF rules added to the algorithms caused a considerable degree of complexity to the problem since all the algorithms reached memory overflow at some time. Since the ST-SPF is more memory-heavy for this specific environment, he reached the memory overflow first at 5 robots, followed by STMS A\*-Based at 5, STMS BFS-Based at 4, and CBS at 15.

This behavior raises the question of whether the use of a Graph-Like environment is good for complex environments with a large number of nodes since it is necessary to save the adjacency list and the adjacency matrix in memory. Despite this drawback, it is worth mentioning that the algorithm among the three that could circumvent this situation is the ST-SPF due to its characteristic of swarms that allows decentralizing the problem through an "Online" implementation, different from CBS and STMS that would need to be centralized.

## 5.2 Traditional Vertical

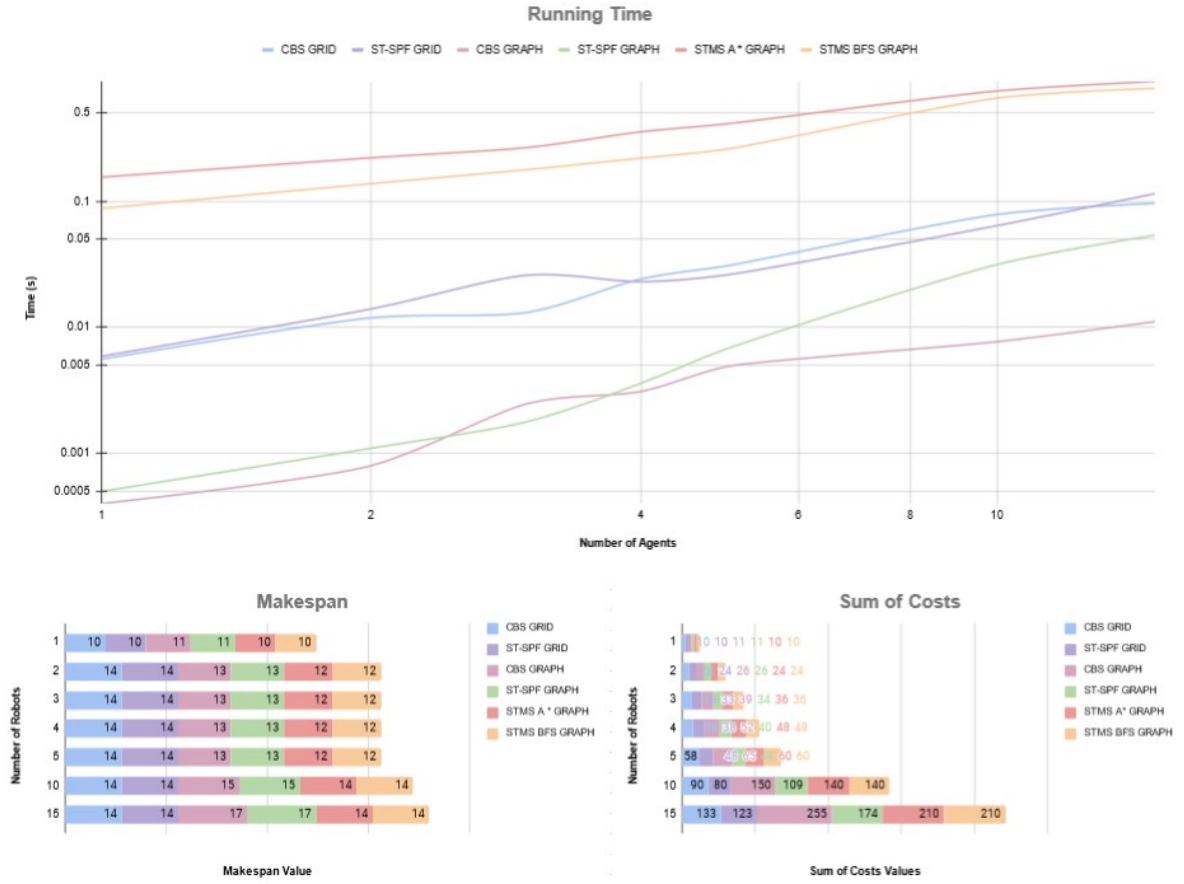
### 5.2.1 Tiny Layout

The Tiny environment has 18 x 8 (144 nodes/grid points) in size, however with the limitation of 128 nodes/grid points free for movement (128 states) due to the location of the Workers and Treadmill zones. Considering the minimum number of robots is 1 and the maximum is 10 since it has 9 pickup and recharge points. However, since the CBS works better in small spaces, we increased the maximum number of robots to 15 to compare which algorithm have more scalability in a complex and crowded environment.

Table 5.5: Tiny Traditional Vertical Results

| Number of Agents | Running Time (s) |        |        |            |           |            | Makespan  |        |     |            |           |            | Sum of Costs |        |     |            |           |            | Winner Algorithm (by Turn) |              |               | Winner Algorithm<br>(by Comparison) |
|------------------|------------------|--------|--------|------------|-----------|------------|-----------|--------|-----|------------|-----------|------------|--------------|--------|-----|------------|-----------|------------|----------------------------|--------------|---------------|-------------------------------------|
|                  | Grid-Like        |        |        | Graph-Like |           |            | Grid-Like |        |     | Graph-Like |           |            | Grid-Like    |        |     | Graph-Like |           |            | Running Time (s)           | Makespan     | Sum of Costs  |                                     |
|                  | CBS              | ST-SPF | CBS    | ST-SPF     | STMS (A*) | STMS (BFS) | CBS       | ST-SPF | CBS | ST-SPF     | STMS (A*) | STMS (BFS) | CBS          | ST-SPF | CBS | ST-SPF     | STMS (A*) | STMS (BFS) |                            |              |               |                                     |
| 1                | 0.0056           | 0.0059 | 0.0004 | 0.0005     | 0.1552    | 0.0877     | 10        | 10     | 11  | 11         | 10        | 10         | 10           | 10     | 11  | 11         | 10        | 10         | CBS (Graph)                | STMS (Graph) | —             | —                                   |
| 2                | 0.0119           | 0.0140 | 0.0008 | 0.0011     | 0.2211    | 0.1379     | 14        | 14     | 13  | 13         | 12        | 12         | 24           | 24     | 26  | 26         | 24        | 24         | CBS (Graph)                | STMS (Graph) | —             | —                                   |
| 3                | 0.0132           | 0.0260 | 0.0025 | 0.0018     | 0.2676    | 0.1787     | 14        | 14     | 13  | 13         | 12        | 12         | 33           | 33     | 39  | 34         | 36        | 36         | ST-SPF (Graph)             | STMS (Graph) | —             | —                                   |
| 4                | 0.0242           | 0.0230 | 0.0031 | 0.0036     | 0.3553    | 0.2199     | 14        | 14     | 13  | 13         | 12        | 12         | 38           | 38     | 52  | 40         | 48        | 48         | CBS (Graph)                | STMS (Graph) | —             | —                                   |
| 5                | 0.0308           | 0.0261 | 0.0049 | 0.0068     | 0.4115    | 0.2594     | 14        | 14     | 13  | 13         | 12        | 12         | 58           | 48     | 65  | 48         | 60        | 60         | CBS (Graph)                | STMS (Graph) | ST-SPF (Grid) | —                                   |
| 10               | 0.0785           | 0.0641 | 0.0077 | 0.0315     | 0.7492    | 0.6574     | 14        | 14     | 15  | 15         | 14        | 14         | 90           | 80     | 150 | 109        | 140       | 140        | CBS (Graph)                | —            | ST-SPF (Grid) | —                                   |
| 15               | 0.0963           | 0.1149 | 0.0111 | 0.0537     | 0.8908    | 0.7846     | 14        | 14     | 17  | 17         | 14        | 14         | 133          | 123    | 255 | 174        | 210       | 210        | ST-SPF (Graph)             | —            | ST-SPF (Grid) | —                                   |

Figure 5.5: Tiny Traditional Vertical Graph



As seen in Figure 5.5 and Table 5.5 the swarm algorithm of the ST-SPF brought a Running Time increase 5% for just one agent for the Grid-Like environment, but the algorithm remained within the same time range as CBS ( $<0.006$  s). As we scale the number of agents, it is possible to see that ST-SPF maintains the same Running Time range as CBS, being able to win with 4, 5, and 10 agents, providing a gain of 5%, 16%, and 20% respectively. The ST-SPF also reduced the sum of costs by approximately 17%, 11%, and 8% for 5, 10, and 15 agents respectively. Comparing this layout with the Tiny Traditional Horizontal, it was possible to reduce the computational complexity of CBS by 2 % and 12 % for 1 and 15 agents, and the ST-SPF to 26 %, 19 %, and 9% for 1, 10, and 15 agents. Despite this gain, this layout brought a considerable gain to Makespan for both algorithms reaching up to 55 % for 3 agents.

Regarding the Graph-Like environment, CBS was the one that had the best result considering the time of execution in general, since the ST-SPF brought an additional gain of

approximately 5% due to the swarm algorithm. However, the ST-SPF achieved better results achieving 25% of Running Time reduction with 3 agents. The STMS algorithms stood out in the reduction of Makespan from 1 to 5 robots when compared to the other algorithms (Grid & Graph-like), also reducing the Sum of Costs for 2 and 3 robots. It is interesting to mention that in this specific layout the STMSs algorithms achieved equal Makespan and Sum of Costs results. This happens since the layout is a little heavier than the Traditional Horizontal. Despite this, the STMS BFS-Based brought a time-complexity reduction of up to 15% when compared to A\*-Based.

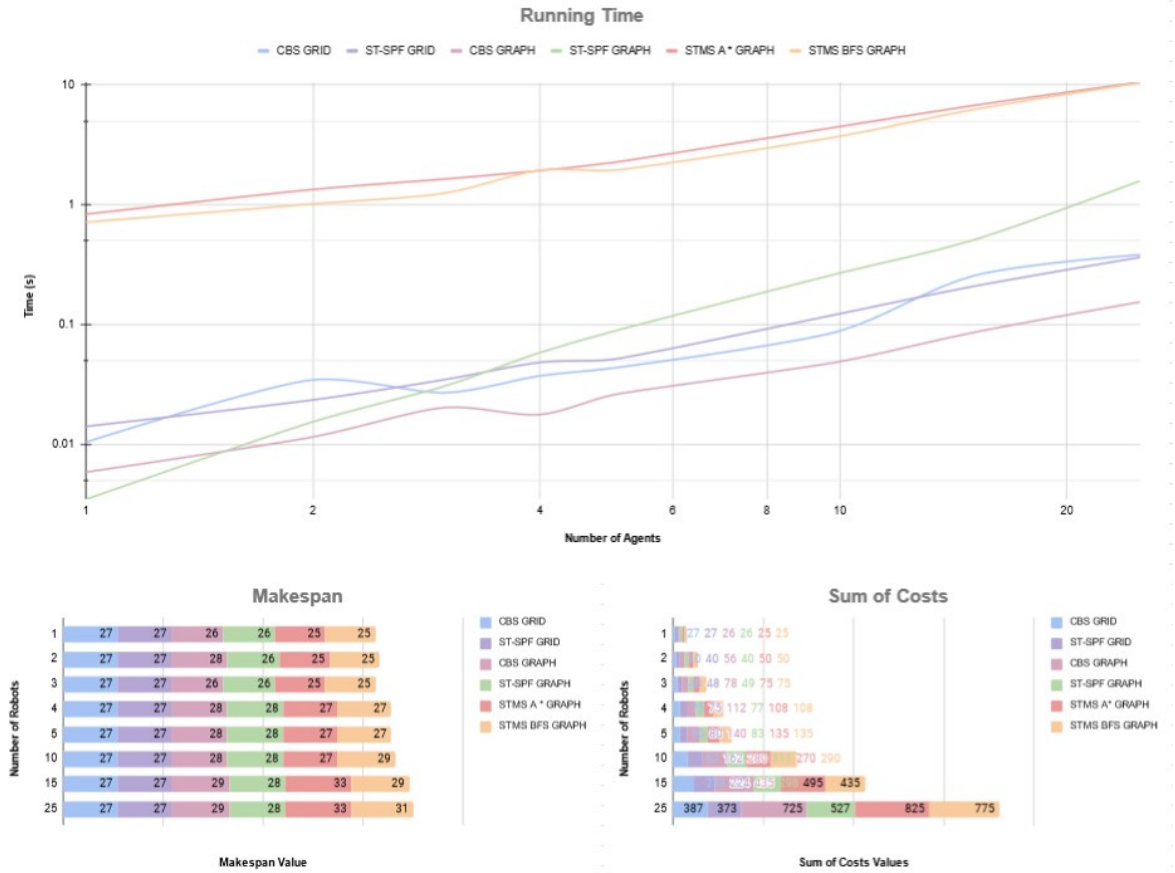
### 5.2.2 Small Layout

The Small environment has 28 x 10 (280 nodes/grid points) in size, however with the limitation of 260 nodes/grid points free for movement (128 states) due to the location of the Workers and Treadmill zones. Considering the minimum number of robots is 1 and the maximum is 25 since it has 24 pickup and recharge points.

Table 5.6: Small Traditional Vertical Results

| Number of Agents | Running Time (s) |        |        |            |           |            | Makespan  |        |     |            |           |            | Sum of Costs |        |     |            |           |            | Winner Algorithm (by Turn) |              |               |              | Winner Algorithm (by Comparison) |  |
|------------------|------------------|--------|--------|------------|-----------|------------|-----------|--------|-----|------------|-----------|------------|--------------|--------|-----|------------|-----------|------------|----------------------------|--------------|---------------|--------------|----------------------------------|--|
|                  | Grid-Like        |        |        | Graph-Like |           |            | Grid-Like |        |     | Graph-Like |           |            | Grid-Like    |        |     | Graph-Like |           |            | Running Time (s)           | Makespan     | Sum of Costs  |              |                                  |  |
|                  | CBS              | ST-SPF | CBS    | ST-SPF     | STMS (A*) | STMS (BFS) | CBS       | ST-SPF | CBS | ST-SPF     | STMS (A*) | STMS (BFS) | CBS          | ST-SPF | CBS | ST-SPF     | STMS (A*) | STMS (BFS) |                            |              |               |              |                                  |  |
| 1                | 0.0105           | 0.0142 | 0.0059 | 0.0035     | 0.8362    | 0.7175     | 27        | 27     | 26  | 26         | 25        | 25         | 27           | 27     | 26  | 26         | 25        | 25         | ST-SPF (Graph)             | STMS (Graph) | STMS (Graph)  | STMS (Graph) | —                                |  |
| 2                | 0.0347           | 0.0236 | 0.0116 | 0.0155     | 1.3446    | 1.0180     | 27        | 27     | 28  | 26         | 25        | 25         | 40           | 40     | 56  | 40         | 50        | 50         | CBS (Graph)                | STMS (Graph) | STMS (Graph)  | STMS (Graph) | —                                |  |
| 3                | 0.0271           | 0.0350 | 0.0204 | 0.0310     | 1.6439    | 1.2565     | 27        | 27     | 26  | 26         | 25        | 25         | 48           | 48     | 78  | 49         | 75        | 75         | CBS (Graph)                | STMS (Graph) | STMS (Graph)  | STMS (Graph) | —                                |  |
| 4                | 0.0375           | 0.0485 | 0.0178 | 0.0586     | 1.9310    | 1.9483     | 27        | 27     | 28  | 28         | 27        | 27         | 75           | 75     | 112 | 77         | 108       | 108        | CBS (Graph)                | —            | —             | —            | —                                |  |
| 5                | 0.0435           | 0.0515 | 0.0260 | 0.0880     | 2.2542    | 1.9411     | 27        | 27     | 28  | 28         | 27        | 27         | 80           | 80     | 140 | 83         | 135       | 135        | CBS (Graph)                | —            | —             | —            | —                                |  |
| 10               | 0.0889           | 0.1236 | 0.0493 | 0.2701     | 4.5005    | 3.7318     | 27        | 27     | 28  | 28         | 27        | 29         | 162          | 162    | 280 | 211        | 270       | 290        | CBS (Graph)                | —            | —             | —            | —                                |  |
| 15               | 0.2550           | 0.2086 | 0.0860 | 0.5065     | 6.7282    | 6.2201     | 27        | 27     | 29  | 28         | 33        | 29         | 238          | 224    | 435 | 298        | 495       | 435        | CBS (Graph)                | —            | ST-SPF (Grid) | STMS (Graph) | —                                |  |
| 25               | 0.3834           | 0.3654 | 0.1550 | 1.5831     | 10.5646   | 10.4116    | 27        | 27     | 29  | 28         | 33        | 31         | 387          | 373    | 725 | 527        | 825       | 775        | CBS (Graph)                | —            | ST-SPF (Grid) | STMS (Graph) | —                                |  |

Figure 5.6: Small Traditional Vertical Graph



As seen in Figure 5.6 and Table 5.6, the ST-SPF swarm creation algorithm brought a Running Time increase of approximately 29% for just one agent for the Grid-Like environment (1% higher than the Small Traditional Horizontal), and was also worse in Running Time when compared to the Tiny Traditional layout since it left the CBS time range ( $> 0.01$  s). As we scale the number of agents, it is possible to see that ST-SPF achieved a better Running Time for 2 and 25 agents, and also a better sum of costs, achieving a reduction by 5% and 4% for 15 and 25 robots.

Regarding the Graph-Like environment, the CBS achieved the lowest Running Time even when compared to the Grid-Like results, however, the ST-SPF performed well for 1 robot achieving a reduction in Running Time by 69%. Lastly, the STMS algorithms achieved a better Makespan from 1 to 3 agents, a gain of 4% when compared to the ST-SPF, and 8% when compared to CBS.



### 5.2.3 Medium Layout

The Medium environment has 50 x 14 (700 nodes/grid points) in size, however with the limitation of 672 nodes/grid points free for movement (128 states) due to the location of the Workers and Treadmill zones. Considering the minimum number of robots is 1 and the maximum is 48 since it has 47 pickup and recharge points.

Table 5.7: Medium Traditional Vertical Results

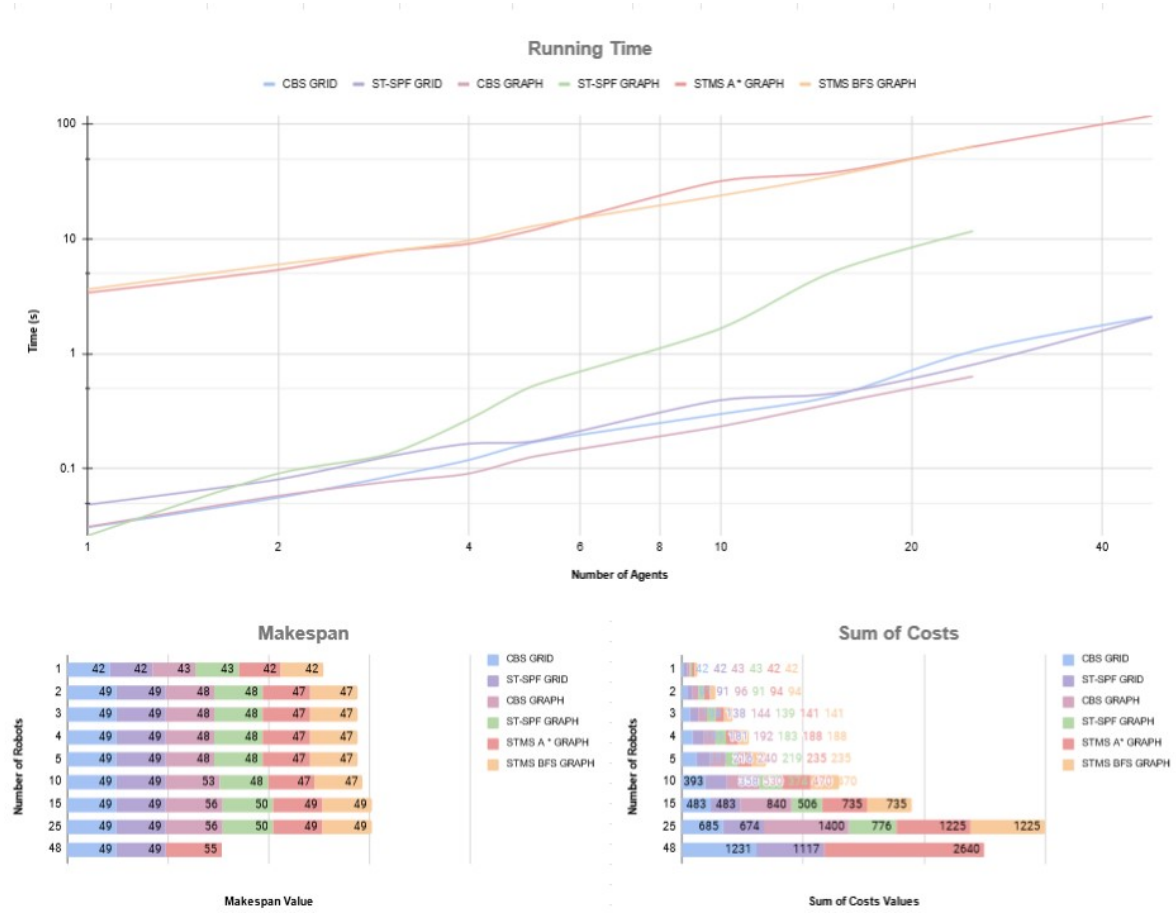
| Number of Agents | Running Time (s) |        |        |            |          |           | Makespan  |        |     |            |          |           | Sum of Costs |        |      |            |          |           | Winner Algorithm (by Turn) |              |               | Winner Algorithm<br>(by Comparison) |
|------------------|------------------|--------|--------|------------|----------|-----------|-----------|--------|-----|------------|----------|-----------|--------------|--------|------|------------|----------|-----------|----------------------------|--------------|---------------|-------------------------------------|
|                  | Grid-Like        |        |        | Graph-Like |          |           | Grid-Like |        |     | Graph-Like |          |           | Grid-Like    |        |      | Graph-Like |          |           | Running Time (s)           | Makespan     | Sum of Costs  |                                     |
|                  | CBS              | ST-SPF | CBS    | ST-SPF     | STMS(A*) | STMS(BFS) | CBS       | ST-SPF | CBS | ST-SPF     | STMS(A*) | STMS(BFS) | CBS          | ST-SPF | CBS  | ST-SPF     | STMS(A*) | STMS(BFS) |                            |              |               |                                     |
| 1                | 0.0307           | 0.0485 | 0.0311 | 0.0260     | 3.4102   | 3.6492    | 42        | 42     | 43  | 43         | 42       | 42        | 42           | 42     | 43   | 43         | 42       | 42        | ST-SPF (Graph)             | —            | —             | ST-SPF (Graph)                      |
| 2                | 0.0558           | 0.0805 | 0.0578 | 0.0906     | 5.3899   | 6.0056    | 49        | 49     | 48  | 48         | 47       | 47        | 91           | 91     | 96   | 91         | 94       | 94        | CBS (Grid)                 | STMS (Graph) | —             | —                                   |
| 3                | 0.0854           | 0.1280 | 0.0771 | 0.1349     | 7.7951   | 7.8502    | 49        | 49     | 48  | 48         | 47       | 47        | 138          | 138    | 144  | 139        | 141      | 141       | CBS (Graph)                | STMS (Graph) | —             | —                                   |
| 4                | 0.1189           | 0.1650 | 0.0905 | 0.2685     | 9.0925   | 9.6871    | 49        | 49     | 48  | 48         | 47       | 47        | 181          | 181    | 192  | 183        | 188      | 188       | CBS (Graph)                | STMS (Graph) | —             | —                                   |
| 5                | 0.1673           | 0.1713 | 0.1255 | 0.5135     | 11.8044  | 12.8095   | 49        | 49     | 48  | 48         | 47       | 47        | 251          | 216    | 240  | 219        | 235      | 235       | CBS (Graph)                | STMS (Graph) | ST-SPF (Grid) | —                                   |
| 10               | 0.2993           | 0.3952 | 0.2337 | 1.6674     | 32.0273  | 23.9989   | 49        | 49     | 53  | 48         | 47       | 47        | 393          | 358    | 530  | 374        | 470      | 470       | CBS (Graph)                | STMS (Graph) | ST-SPF (Grid) | —                                   |
| 15               | 0.4270           | 0.4506 | 0.3681 | 5.1619     | 37.9543  | 35.3796   | 49        | 49     | 56  | 50         | 49       | 49        | 483          | 483    | 840  | 506        | 735      | 735       | CBS (Graph)                | —            | —             | CBS (Graph)                         |
| 25               | 1.0554           | 0.8043 | 0.635  | 11.7222    | 63.6951  | 64.6201   | 49        | 49     | 56  | 50         | 49       | 49        | 685          | 674    | 1400 | 776        | 1225     | 1225      | CBS (Graph)                | —            | ST-SPF (Grid) | —                                   |
| 48               | 2.1159           | 2.0936 | —      | —          | 118.9849 | —         | 49        | 49     | —   | —          | 55       | —         | 1231         | 1117   | —    | —          | 2640     | —         | ST-SPF (Grid)              | —            | ST-SPF (Grid) | ST-SPF (Grid)                       |

As seen in Figure 5.7 and Table 5.7, the ST-SPF swarm creation algorithm brought a Running Time increase of approximately 44% for just one agent for the Grid-Like environment, almost the half when compared to Medium Traditional Horizontal. As we scale the number of agents it is possible to see that ST-SPF was better than CBS in Running Time for 25 and 48 agents, decreasing by 27% the Running Time. The ST-SPF also had better results for the sum of costs, where the algorithm reduced the value by 9% for 48 robots.

Regarding the Graph-Like environment, the CBS achieved the lowest Running Time even when compared to the Grid-Like results, however, the ST-SPF achieved the best Running Time results for 1 agent for both Grid and Graph-Like environments. The STMS algorithms achieved a better Makespan from 2 to 10 agents, a gain of 4% when compared to the ST-SPF, and 8% when compared to CBS. Finally, is worth mentioning that in this particular layout, the STMS BFS-Based achieved memory overflow for 48 agents while the CBS and ST-SPF didn't found a solution.



Figure 5.7: Medium Traditional Vertical Graph



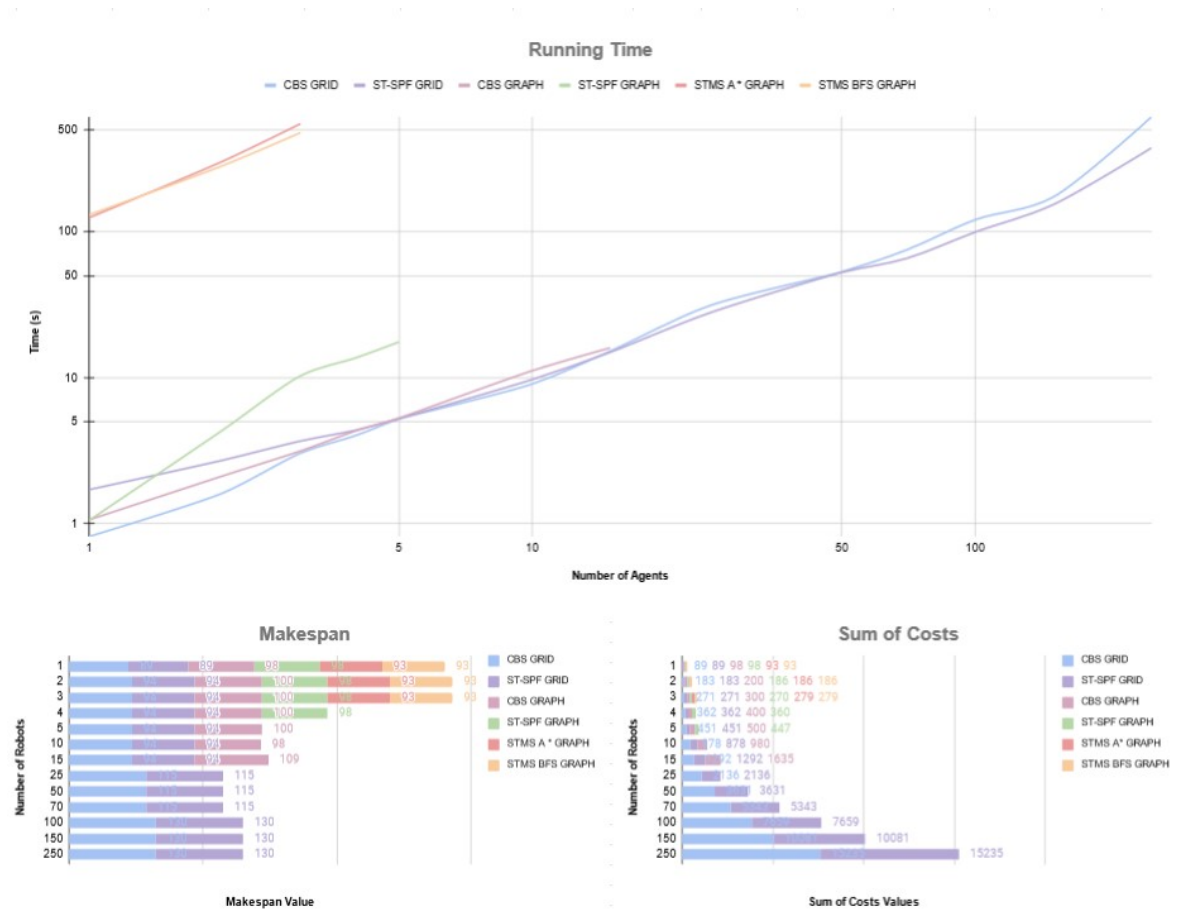
### 5.2.4 Large Layout

The Large environment has 100 x 50 (5000 nodes/grid points) in size, however with the limitation of 4900 nodes/grid points free for movement (128 states) due to the location of the Workers and Treadmill zones. Considering the minimum number of robots is 1 and the maximum is 98 since it has 97 pickup and recharge points. However, to test the complexity and scalability of the algorithms, the maximum number of robots was increased to 250.

Table 5.8: Large Traditional Vertical Results

| Number of Agents | Running Time (s) |          |         |            |           |            | Makespan  |        |     |            |           |            | Sum of Costs |        |      |            |           |            | Winner Algorithm (by Turn) |          |              | Winner Algorithm (by Comparison) |
|------------------|------------------|----------|---------|------------|-----------|------------|-----------|--------|-----|------------|-----------|------------|--------------|--------|------|------------|-----------|------------|----------------------------|----------|--------------|----------------------------------|
|                  | Grid-Like        |          |         | Graph-Like |           |            | Grid-Like |        |     | Graph-Like |           |            | Grid-Like    |        |      | Graph-Like |           |            | Running Time (s)           | Makespan | Sum of Costs |                                  |
|                  | CBS              | ST-SPF   | CBS     | ST-SPF     | STMS (A*) | STMS (BFS) | CBS       | ST-SPF | CBS | ST-SPF     | STMS (A*) | STMS (BFS) | CBS          | ST-SPF | CBS  | ST-SPF     | STMS (A*) | STMS (BFS) |                            |          |              |                                  |
| 1                | 0.8183           | 1.7135   | 1.0645  | 1.0557     | 125.0262  | 130.9634   | 89        | 89     | 98  | 98         | 93        | 93         | 89           | 89     | 98   | 98         | 93        | 93         | CBS (Grid)                 | —        | —            | CBS (Grid)                       |
| 2                | 1.6215           | 2.7236   | 2.1257  | 4.3762     | 304.0403  | 283.9042   | 94        | 94     | 100 | 98         | 93        | 93         | 183          | 183    | 200  | 186        | 186       | 186        | CBS (Grid)                 | —        | —            | CBS (Grid)                       |
| 3                | 3.0484           | 3.6892   | 3.1526  | 10.3219    | 550.9341  | 478.5309   | 94        | 94     | 100 | 98         | 93        | 93         | 271          | 271    | 300  | 270        | 279       | 279        | CBS (Grid)                 | —        | —            | CBS (Grid)                       |
| 4                | 4.0292           | 4.3808   | 4.3334  | 13.7135    | —         | —          | 94        | 94     | 100 | 98         | —         | —          | 362          | 362    | 400  | 360        | —         | —          | CBS (Grid)                 | —        | —            | CBS (Grid)                       |
| 5                | 5.2121           | 5.2489   | 5.2947  | 17.5941    | —         | —          | 94        | 94     | 100 | —          | —         | —          | 451          | 451    | 500  | 447        | —         | —          | CBS (Grid)                 | —        | —            | CBS (Grid)                       |
| 10               | 9.0773           | 9.7331   | 11.1878 | —          | —         | —          | 94        | 94     | 98  | —          | —         | —          | 878          | 878    | 980  | —          | —         | —          | CBS (Grid)                 | —        | —            | CBS (Grid)                       |
| 15               | 15.2324          | 14.9731  | 16.0462 | —          | —         | —          | 94        | 94     | 109 | —          | —         | —          | 1292         | 1292   | 1635 | —          | —         | —          | ST-SPF(Grid)               | —        | —            | ST-SPF(Grid)                     |
| 25               | 31.0096          | 27.6759  | —       | —          | —         | —          | 115       | 115    | —   | —          | —         | —          | 2136         | 2136   | —    | —          | —         | —          | ST-SPF(Grid)               | —        | —            | ST-SPF(Grid)                     |
| 50               | 53.2806          | 52.7702  | —       | —          | —         | —          | 115       | 115    | —   | —          | —         | —          | 3631         | 3631   | —    | —          | —         | —          | ST-SPF(Grid)               | —        | —            | ST-SPF(Grid)                     |
| 70               | 75.2212          | 65.8104  | —       | —          | —         | —          | 115       | 115    | —   | —          | —         | —          | 5343         | 5343   | —    | —          | —         | —          | ST-SPF(Grid)               | —        | —            | ST-SPF(Grid)                     |
| 100              | 121.0908         | 99.6331  | —       | —          | —         | —          | 130       | 130    | —   | —          | —         | —          | 7659         | 7659   | —    | —          | —         | —          | ST-SPF(Grid)               | —        | —            | ST-SPF(Grid)                     |
| 150              | 173.2315         | 153.7619 | —       | —          | —         | —          | 130       | 130    | —   | —          | —         | —          | 10081        | 10081  | —    | —          | —         | —          | ST-SPF(Grid)               | —        | —            | ST-SPF(Grid)                     |
| 250              | 614.2015         | 376.5263 | —       | —          | —         | —          | 130       | 130    | —   | —          | —         | —          | 15235        | 15235  | —    | —          | —         | —          | ST-SPF(Grid)               | —        | —            | ST-SPF(Grid)                     |

Figure 5.8: Large Traditional Vertical Graph



As seen in Figure 5.8 and Table 5.8, the ST-SPF swarm creation algorithm brought a Running Time increase of approximately 71% for just one agent for the Grid-Like environment. However, as we scale the number of agents, it is possible to see that ST-SPF was better than CBS in Running Time from 15 to 250 agents. In this environment size, was possible to decrease by 48% the Running Time. In some parts, this layout showed better results regarding Running Time than the Large Traditional Horizontal for both CBS and ST-SPF,

achieving up to 21% and 14% time reduction for 25 robots respectively.

Regarding the Graph-Like environment, neither the CBS, STMS (A\*-Based and BFS-Based), or ST-SPF performed well. As in the Medium size, it is possible to identify that the MAPF rules added to the algorithms caused a considerable degree of complexity to the problem since all the algorithms reached memory overflow or time limit at some time. Different from the Medium Traditional Horizontal Layout, the STMS algorithms reached the time limit with 4 agents, and the CBS and ST-SPF achieved memory overflow at 15 and 10 agents respectively, raising another flag to the use of a Graph-Like environment with a large number of nodes and agents.

## 5.3 Flying-V

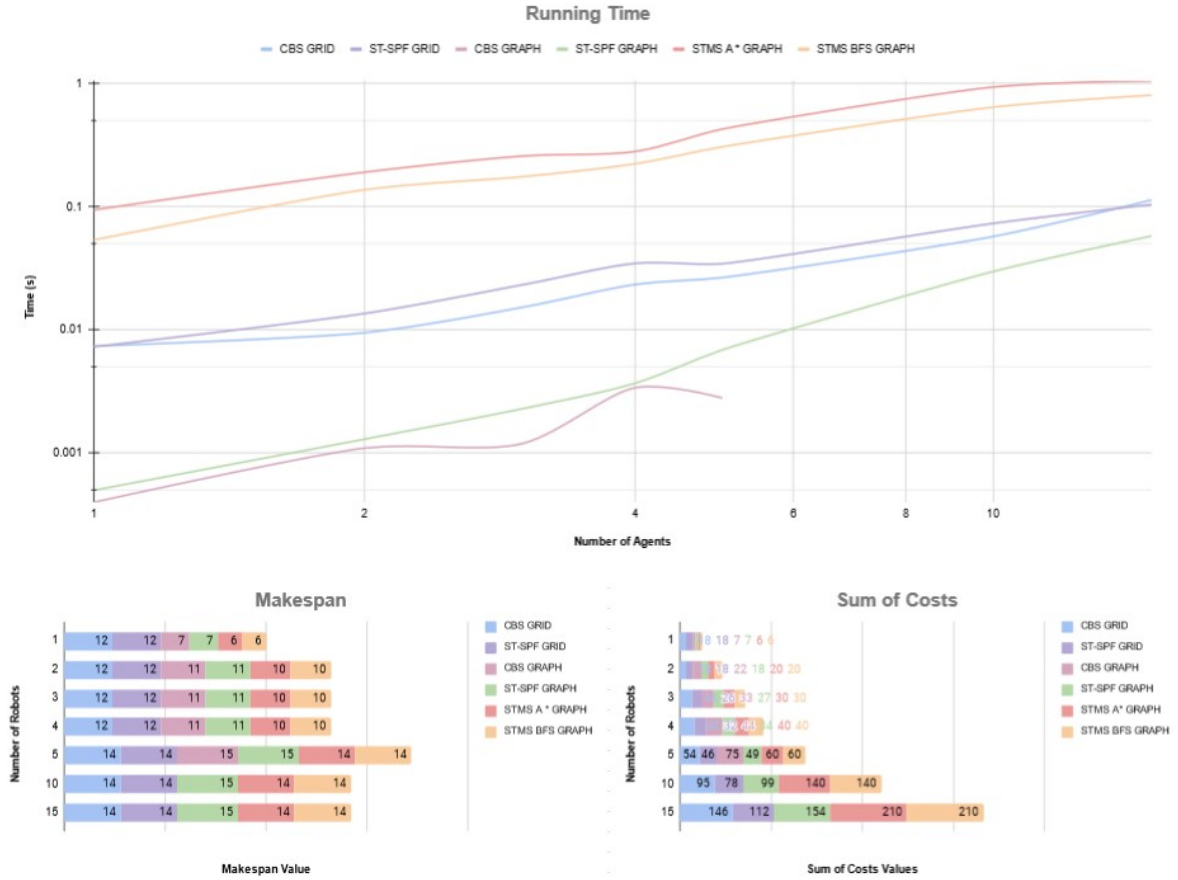
### 5.3.1 Tiny Layout

The Tiny environment has 18 x 8 (144 nodes/grid points) in size, however with the limitation of 128 nodes/grid points free for movement (128 states) due to the location of the Workers and Treadmill zones. Considering the minimum number of robots is 1 and the maximum is 10 since it has 9 pickup and recharge points. However, since the CBS works better in small spaces, we increased the maximum number of robots to 15 to compare which algorithm have more scalability in complex and crowded environment.

Table 5.9: Tiny Flying-V Results

| Number of Agents | Running Time (s) |        |            |        |           |            | Makespan  |        |            |        |           |            | Sum of Costs |        |            |        |           |            | Winner Algorithm (by Turn) |              |               | Winner Algorithm<br>(by Comparison) |
|------------------|------------------|--------|------------|--------|-----------|------------|-----------|--------|------------|--------|-----------|------------|--------------|--------|------------|--------|-----------|------------|----------------------------|--------------|---------------|-------------------------------------|
|                  | Grid-Like        |        | Graph-Like |        |           |            | Grid-Like |        | Graph-Like |        |           |            | Grid-Like    |        | Graph-Like |        |           |            | Running Time (s)           | Makespan     | Sum of Costs  |                                     |
|                  | CBS              | ST-SPF | CBS        | ST-SPF | STMS (A*) | STMS (BFS) | CBS       | ST-SPF | CBS        | ST-SPF | STMS (A*) | STMS (BFS) | CBS          | ST-SPF | CBS        | ST-SPF | STMS (A*) | STMS (BFS) |                            |              |               |                                     |
| 1                | 0.0074           | 0.0073 | 0.0004     | 0.0005 | 0.0944    | 0.0538     | 12        | 12     | 7          | 7      | 6         | 6          | 18           | 18     | 7          | 7      | 6         | 6          | CBS (Graph)                | STMS (Graph) | STMS (Graph)  | STMS (Graph)                        |
| 2                | 0.0095           | 0.0136 | 0.0011     | 0.0013 | 0.1906    | 0.1373     | 12        | 12     | 11         | 11     | 10        | 10         | 18           | 18     | 22         | 18     | 20        | 20         | CBS (Graph)                | STMS (Graph) | —             | —                                   |
| 3                | 0.0153           | 0.0233 | 0.0012     | 0.0023 | 0.2577    | 0.1753     | 12        | 12     | 11         | 11     | 10        | 10         | 34           | 26     | 33         | 27     | 30        | 30         | CBS (Graph)                | STMS (Graph) | ST-SPF (Grid) | —                                   |
| 4                | 0.0234           | 0.0347 | 0.0034     | 0.0037 | 0.2805    | 0.2230     | 12        | 12     | 11         | 11     | 10        | 10         | 40           | 32     | 44         | 34     | 40        | 40         | CBS (Graph)                | STMS (Graph) | ST-SPF (Grid) | —                                   |
| 5                | 0.0266           | 0.0345 | 0.0028     | 0.0069 | 0.4269    | 0.3064     | 14        | 14     | 15         | 15     | 14        | 14         | 54           | 46     | 75         | 49     | 60        | 60         | CBS (Graph)                | STMS (Graph) | ST-SPF (Grid) | —                                   |
| 10               | 0.0573           | 0.0732 | —          | 0.0298 | 0.9352    | 0.6422     | 14        | 14     | —          | 15     | 14        | 14         | 95           | 78     | —          | 99     | 140       | 140        | ST-SPF (Graph)             | STMS (Graph) | ST-SPF (Grid) | —                                   |
| 15               | 0.1135           | 0.1043 | —          | 0.0578 | 1.0514    | 0.8038     | 14        | 14     | —          | 15     | 14        | 14         | 146          | 112    | —          | 154    | 210       | 210        | ST-SPF (Graph)             | STMS (Graph) | ST-SPF (Grid) | —                                   |

Figure 5.9: Tiny Flying-V Graph



As seen in Figure 5.9 and Table 5.9 the swarm algorithm of the ST-SPF brought a Running Time decrease by approximately 1% for just one agent at the Grid-Like environment, but the CBS remained within the same time range ( $<0.008$  s). As we scale the number of agents, it is possible to see that CBS stayed ahead from 1 to 10 agents, falling behind at 15 agents, where the ST-SPF reduced the Running Time by approximately 8%. The ST-SPF had the same sum of costs values as CBS for 1 and 2 agents but reduced these values up to 24% from 3 to 15 agents. Comparing this layout with the Tiny Traditional Horizontal, it was possible to reduce the computational complexity of CBS by 7% for 10 agents, and the ST-SPF to 8%, and 17% for 10, and 15 agents. Comparing to the Tiny Traditional Vertical, it was possible to reduce the computational complexity of CBS by 27% for 10 agents, and the ST-SPF to 9% for 15 agents.

Regarding the Graph-Like environment, CBS was the one that had the best result considering the time of execution from 1 to 5 agents, since the ST-SPF brought an additional gain

of approximately 18% due to the swarm algorithm. However, the ST-SPF achieved better results for 10 and 15 robots, being better than the Grid-Like algorithms. The STMS stood out in the reduction of Makespan when compared to the other algorithms (Grid & Graph-like), however, the ST-SPF achieved a better Sum of Costs from 3 to 15 agents. Comparing the two STMS algorithms, it is possible to notice that the BFS-Based brought a reduction of time-complexity of up to 31% when compared to the A -Based, and both achieved up to 50% Makespan reduction when compared to the Grid-Like algorithms.

Finally, like the Tiny Traditional Vertical, the CBS and STMS were unable to find a solution for the Graph-Like environment after 8 and 12 agents respectively. However, the ST-SPF and STMS algorithms found a solution for all the number of agents.

### 5.3.2 Small Layout

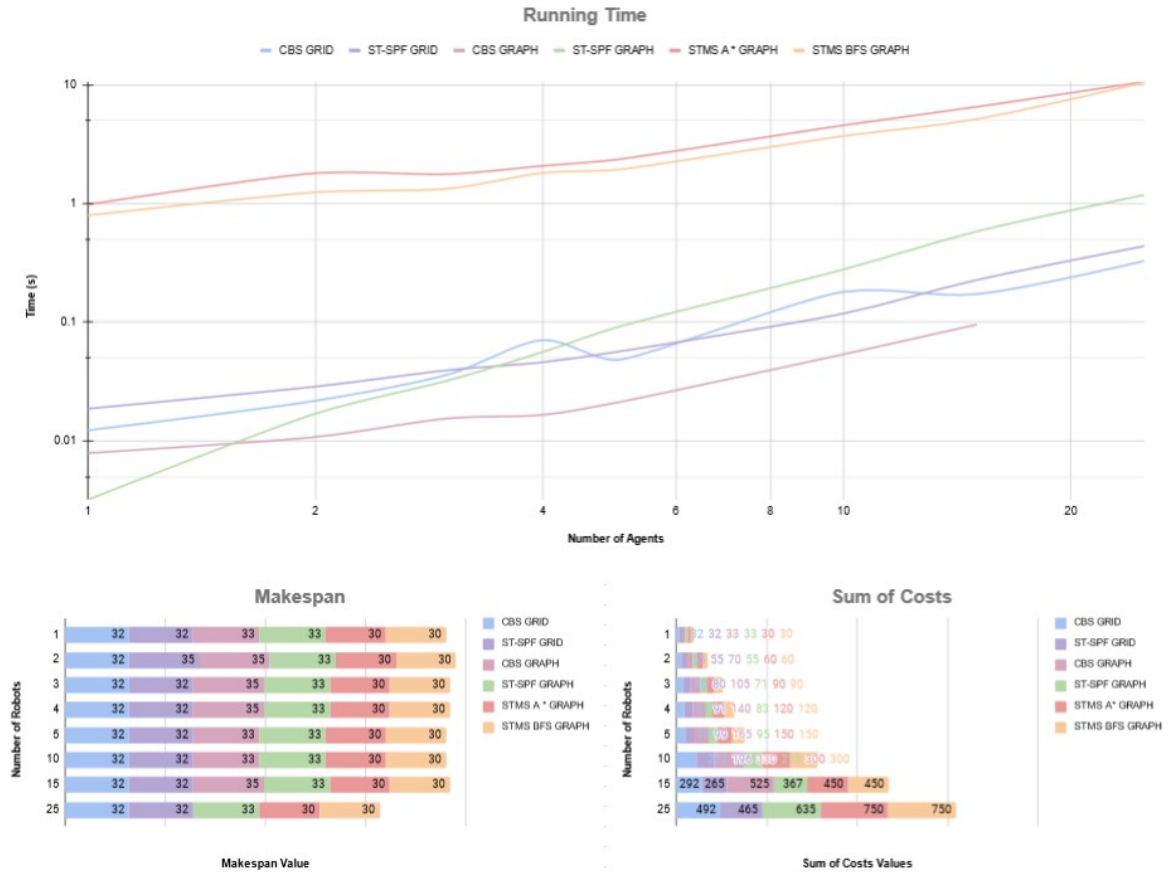
The Small environment has 28 x 10 (280 nodes/grid points) in size, however with the limitation of 260 nodes/grid points free for movement (128 states) due to the location of the Workers and Treadmill zones. Considering the minimum number of robots is 1 and the maximum is 25 since it has 24 pickup and recharge points.

Table 5.10: Small Flying-V Results

| Number of Agents | Running Time (s) |        |        |            |           |            | Makespan  |        |     |            |           |            | Sum of Costs |        |     |            |           |            | Winner Algorithm (by Turn) |              |               | Winner Algorithm<br>(by Comparison) |
|------------------|------------------|--------|--------|------------|-----------|------------|-----------|--------|-----|------------|-----------|------------|--------------|--------|-----|------------|-----------|------------|----------------------------|--------------|---------------|-------------------------------------|
|                  | Grid-Like        |        |        | Graph-Like |           |            | Grid-Like |        |     | Graph-Like |           |            | Grid-Like    |        |     | Graph-Like |           |            | Running Time (s)           | Makespan     | Sum of Costs  |                                     |
|                  | CBS              | ST-SPF | CBS    | ST-SPF     | STMS (A*) | STMS (BFS) | CBS       | ST-SPF | CBS | ST-SPF     | STMS (A*) | STMS (BFS) | CBS          | ST-SPF | CBS | ST-SPF     | STMS (A*) | STMS (BFS) |                            |              |               |                                     |
| 1                | 0.0123           | 0.0187 | 0.0079 | 0.0032     | 0.9837    | 0.7972     | 32        | 32     | 33  | 33         | 30        | 30         | 32           | 32     | 33  | 33         | 30        | 30         | CBS (Graph)                | STMS (Graph) | STMS (Graph)  | STMS (Graph)                        |
| 2                | 0.0218           | 0.0287 | 0.0108 | 0.0170     | 1.8070    | 1.2456     | 32        | 35     | 35  | 33         | 30        | 30         | 55           | 55     | 70  | 55         | 60        | 60         | CBS (Graph)                | STMS (Graph) | —             | —                                   |
| 3                | 0.0364           | 0.0395 | 0.0155 | 0.0324     | 1.7732    | 1.3375     | 32        | 32     | 35  | 33         | 30        | 30         | 80           | 80     | 105 | 71         | 90        | 90         | CBS (Graph)                | STMS (Graph) | —             | —                                   |
| 4                | 0.0707           | 0.0461 | 0.0166 | 0.0563     | 2.0827    | 1.8188     | 32        | 32     | 35  | 33         | 30        | 30         | 91           | 91     | 140 | 83         | 120       | 120        | CBS (Graph)                | STMS (Graph) | —             | —                                   |
| 5                | 0.0481           | 0.0562 | 0.0210 | 0.0905     | 2.3486    | 1.9272     | 32        | 32     | 33  | 33         | 30        | 30         | 99           | 99     | 165 | 95         | 150       | 150        | CBS (Graph)                | STMS (Graph) | —             | —                                   |
| 10               | 0.1805           | 0.1186 | 0.0538 | 0.2792     | 4.5756    | 3.7159     | 32        | 32     | 33  | 33         | 30        | 30         | 223          | 196    | 330 | 213        | 300       | 300        | CBS (Graph)                | STMS (Graph) | ST-SPF (Grid) | —                                   |
| 15               | 0.1728           | 0.2272 | 0.0957 | 0.5833     | 6.5447    | 5.1538     | 32        | 32     | 35  | 33         | 30        | 30         | 292          | 265    | 525 | 367        | 450       | 450        | CBS (Graph)                | STMS (Graph) | ST-SPF (Grid) | —                                   |
| 25               | 0.5289           | 0.4386 | —      | 1.1826     | 10.6447   | 10.5264    | 32        | 32     | —   | 33         | 30        | 30         | 492          | 465    | —   | 635        | 750       | 750        | ST-SPF (Grid)              | STMS (Graph) | ST-SPF (Grid) | ST-SPF (Grid)                       |

As seen in Figure 5.10 and Table 5.10, the ST-SPF swarm creation algorithm brought a Running Time increase of approximately 41% for just one agent for the Grid-Like environment. As we scale the number of agents, it is possible to see that ST-SPF achieved a better Running Time for 4 and 10 agents, achieving a reduction by 41% and 42%. Regarding the Sum of Costs, the ST-SPF reduced the values when compared to CBS from 10 to 25 robots up to 12%.

Figure 5.10: Small Flying-V Graph



Regarding the Graph-Like environment, the CBS achieved the lowest Running Time even when compared to the Grid-Like results. Regarding the STMS algorithms, it was possible to reduce the Makespan up to 17% compared to CBS (Graph-Like), and 10% compared to ST-SPF (Graph-Like). Finally, it is worth mentioning that CBS didn't find a solution starting with 24 agents.

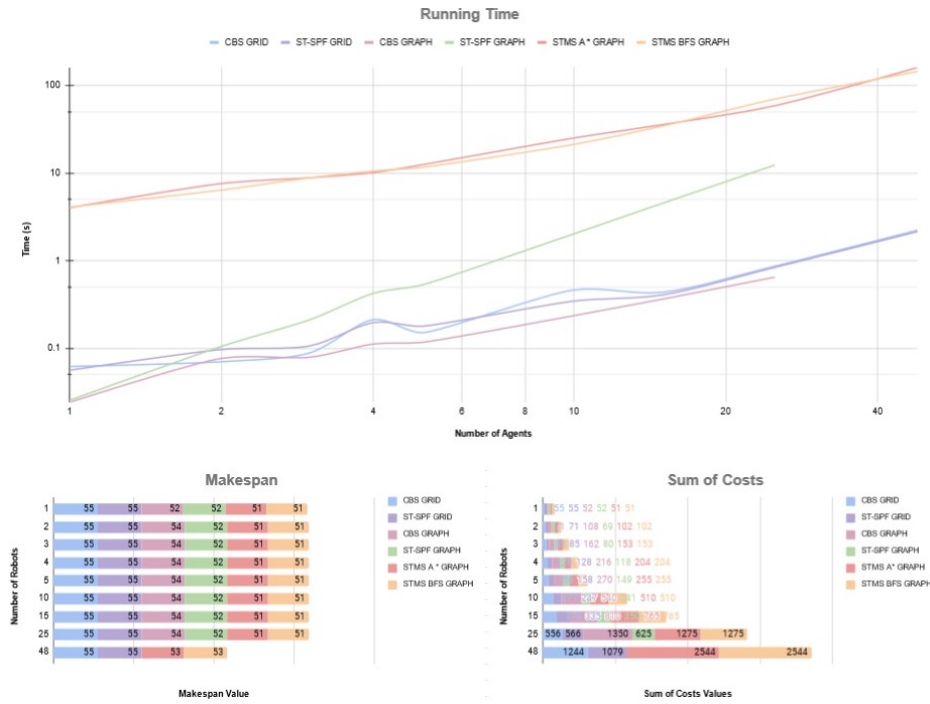
### 5.3.3 Medium Layout

The Medium environment has 50 x 14 (700 nodes/grid points) in size, however with the limitation of 672 nodes/grid points free for movement (128 states) due to the location of the Workers and Treadmill zones. Considering the minimum number of robots is 1 and the maximum is 48 since it has 47 pickup and recharge points.

Table 5.11: Medium Flying-V Results

| Number of Agents | Running Time (s) |        |        |            |          |           | Makespan   |     |        |            |     |        | Sum of Costs |            |      |            |      |      | Winner Algorithm (by Turn) |              |                | Winner Algorithm (by Comparison) |
|------------------|------------------|--------|--------|------------|----------|-----------|------------|-----|--------|------------|-----|--------|--------------|------------|------|------------|------|------|----------------------------|--------------|----------------|----------------------------------|
|                  | Grid-Like        |        |        | Graph-Like |          |           | Grid-Like  |     |        | Graph-Like |     |        | Grid-Like    |            |      | Graph-Like |      |      | Running Time (s)           | Makespan     | Sum of Costs   |                                  |
|                  | CBS              | ST-SPF |        | CBS        | ST-SPF   | STMS (A*) | STMS (BFS) | CBS | ST-SPF |            | CBS | ST-SPF | STMS (A*)    | STMS (BFS) | CBS  | ST-SPF     |      | CBS  |                            |              |                |                                  |
| 1                | 0.0620           | 0.0562 | 0.0242 | 0.0255     | 4.0251   | 4.0874    | 55         | 55  | 52     | 52         | 51  | 51     | 55           | 55         | 52   | 52         | 51   | 51   | CBS (Graph)                | STMS (Graph) | STMS (Graph)   | STMS (Graph)                     |
| 2                | 0.0703           | 0.0969 | 0.0767 | 0.1053     | 7.5914   | 6.4045    | 55         | 55  | 54     | 52         | 51  | 51     | 71           | 71         | 108  | 69         | 102  | 102  | CBS (Grid)                 | STMS (Graph) | ST-SPF (Graph) | —                                |
| 3                | 0.0890           | 0.1068 | 0.0792 | 0.2130     | 8.8422   | 8.8336    | 55         | 55  | 54     | 52         | 51  | 51     | 85           | 85         | 162  | 80         | 153  | 153  | CBS (Graph)                | STMS (Graph) | ST-SPF (Graph) | —                                |
| 4                | 0.2121           | 0.1968 | 0.1120 | 0.4241     | 10.1330  | 10.5600   | 55         | 55  | 54     | 52         | 51  | 51     | 128          | 128        | 216  | 118        | 204  | 204  | CBS (Graph)                | STMS (Graph) | ST-SPF (Graph) | —                                |
| 5                | 0.1506           | 0.1788 | 0.1169 | 0.5286     | 12.4301  | 11.5972   | 55         | 55  | 54     | 52         | 51  | 51     | 158          | 158        | 270  | 149        | 255  | 255  | CBS (Graph)                | STMS (Graph) | ST-SPF (Graph) | —                                |
| 10               | 0.4635           | 0.3461 | 0.2359 | 2.0258     | 25.1207  | 21.2579   | 55         | 55  | 54     | 52         | 51  | 51     | 277          | 247        | 540  | 241        | 510  | 510  | CBS (Graph)                | STMS (Graph) | ST-SPF (Graph) | —                                |
| 15               | 0.4364           | 0.4070 | 0.3649 | 4.5344     | 35.7144  | 34.4071   | 55         | 55  | 54     | 52         | 51  | 51     | 379          | 335        | 810  | 350        | 765  | 765  | CBS (Graph)                | STMS (Graph) | ST-SPF (Grid)  | —                                |
| 25               | 0.8644           | 0.8367 | 0.648  | 12.3249    | 58.6306  | 69.6978   | 55         | 55  | 54     | 52         | 51  | 51     | 556          | 566        | 1350 | 625        | 1275 | 1275 | CBS (Graph)                | STMS (Graph) | ST-SPF (Grid)  | —                                |
| 48               | 2.2329           | 2.1447 | —      | —          | 160.0627 | 143.9272  | 55         | 55  | —      | —          | 53  | 53     | 1244         | 1079       | —    | —          | 2544 | 2544 | ST - SPF (Grid)            | —            | ST-SPF (Grid)  | ST-SPF (Grid)                    |

Figure 5.11: Medium Flying-V Graph



As seen in Figure 5.11 and Table 5.11, the ST-SPF swarm creation algorithm brought a Running Time decrease of approximately 10% for just one agent for the Grid-Like environment. As we scale the number of agents it is possible to see that ST-SPF was better than CBS in Running Time for 4, 10, 15, and 25 agents, decreasing the Running Time up to 29%. The ST-SPF also had better results for the sum of costs, where the algorithm reduced the value by 13% for 48 robots.

Regarding the Graph-Like environment, the CBS achieved the lowest Running Time even when compared to the Grid-Like results (being behind only at 2 robots for the Grid-Like CBS). The STMS achieved a better makespan from 1 to 48 agents, also getting a better Sum of Costs for 1 agent. Finally, the ST-SPF achieved the best sum of costs from 2 to 25 agents when compared to the Grid-Like and Graph-Like algorithms.



In this specific size, we identify again that the MAPF rules added to the algorithms in conjunction with the size of the environment started to affect the ST-SPF and CBS. The CBS didn't found a solution starting with 31 agents, while the ST-SPF achieved memory overflow with 48 agents. The STMS algorithms then stands out in the Graph-Like environment since was possible to find a solution for all the number of agents, however, it remains behind the Grid-Like CBS and ST-SPF in Running Time, and the ST-SPF (Grid-Like) for the Sum of Costs.

### 5.3.4 Large Layout

The Large environment has 100 x 50 (5000 nodes/grid points) in size, however with the limitation of 4900 nodes/grid points free for movement (128 states) due to the location of the Workers and Treadmill. Considering the minimum number of robots is 1 and the maximum is 98 since it has 97 pickup and recharge points. However, to test the complexity and scalability of the algorithms, the maximum number of robots was increased to 250.

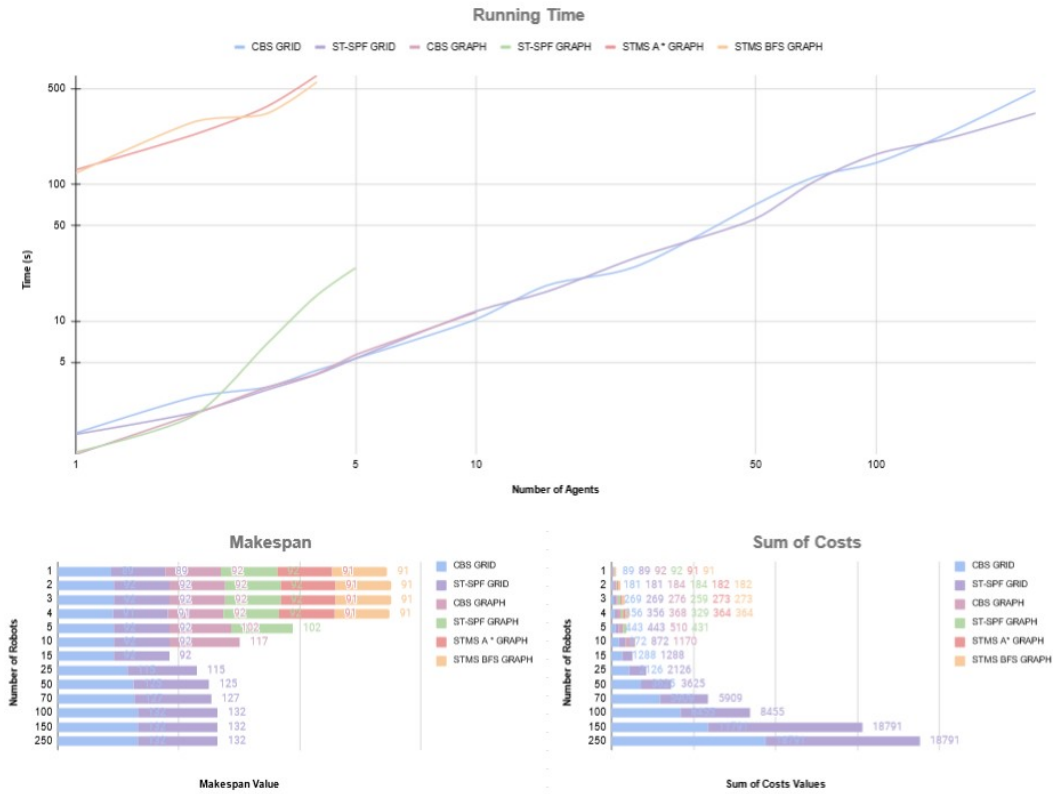
As seen in Figure 5.12 and Table 5.12, the ST-SPF swarm creation algorithm brought a Running Time decrease of approximately 2% for just one agent for the Grid-Like environment. However, as we scale the number of agents, it is possible to see that ST-SPF was better than CBS in Running Time for 15, 50, 70, 150, and 250 agents. In this environment size, was possible to decrease the Running Time up to 37%. In some parts, this layout showed better results regarding Running Time than the Large Traditional Vertical for both CBS and ST-SPF, achieving up to 21% and 11% time reduction for 250 agents respectively. This layout also showed better results than the Large Traditional Horizontal, achieving up to 23% and 18% time reduction for 250 agents respectively.

Table 5.12: Large Flying-V Results

| Number of Agents | Running Time (s) |          |            |         |           |            | Makespan  |        |            |        |           |            | Sum of Costs |        |            |        |           |            | Winner Algorithm (by Turn) |          |              | Winner Algorithm |
|------------------|------------------|----------|------------|---------|-----------|------------|-----------|--------|------------|--------|-----------|------------|--------------|--------|------------|--------|-----------|------------|----------------------------|----------|--------------|------------------|
|                  | Grid-Like        |          | Graph-Like |         |           |            | Grid-Like |        | Graph-Like |        |           |            | Grid-Like    |        | Graph-Like |        |           |            | Running Time (s)           | Makespan | Sum of Costs |                  |
|                  | CBS              | ST-SPF   | CBS        | ST-SPF  | STMS (A*) | STMS (BFS) | CBS       | ST-SPF | CBS        | ST-SPF | STMS (A*) | STMS (BFS) | CBS          | ST-SPF | CBS        | ST-SPF | STMS (A*) | STMS (BFS) |                            |          |              |                  |
| 1                | 1.5181           | 1.4882   | 1.0617     | 1.1034  | 127.9196  | 120.6299   | 89        | 89     | 92         | 92     | 91        | 91         | 89           | 89     | 92         | 92     | 91        | 91         | CBS (Grid)                 | —        | —            | CBS (Grid)       |
| 2                | 2.8069           | 2.1618   | 2.1155     | 2.0857  | 234.1463  | 289.3908   | 92        | 92     | 92         | 92     | 91        | 91         | 181          | 181    | 184        | 184    | 182       | 182        | CBS (Grid)                 | —        | —            | CBS (Grid)       |
| 3                | 3.2922           | 3.1421   | 3.2731     | 6.8194  | 370.4300  | 329.2146   | 92        | 92     | 92         | 92     | 91        | 91         | 269          | 269    | 276        | 259    | 273       | 273        | CBS (Grid)                 | —        | —            | CBS (Grid)       |
| 4                | 4.3643           | 4.0773   | 4.1275     | 15.4263 | 624.9622  | 559.9809   | 91        | 91     | 92         | 92     | 91        | 91         | 356          | 356    | 368        | 329    | 364       | 364        | CBS (Grid)                 | —        | —            | ST-SPF (Graph)   |
| 5                | 5.3238           | 5.3466   | 5.6680     | 24.4766 | —         | —          | 92        | 92     | 102        | 102    | —         | —          | 443          | 443    | 510        | 431    | —         | —          | CBS (Grid)                 | —        | —            | ST-SPF (Graph)   |
| 10               | 10.3259          | 11.8068  | 11.5833    | —       | —         | —          | 92        | 92     | 117        | —      | —         | —          | 872          | 872    | 1170       | —      | —         | —          | CBS (Grid)                 | —        | —            | CBS (Grid)       |
| 15               | 18.1873          | 16.3962  | —          | —       | —         | —          | 92        | 92     | —          | —      | —         | —          | 1288         | 1288   | —          | —      | —         | —          | ST-SPF (Grid)              | —        | —            | ST-SPF (Grid)    |
| 25               | 24.9604          | 28.9733  | —          | —       | —         | —          | 115       | 115    | —          | —      | —         | —          | 2126         | 2126   | —          | —      | —         | —          | CBS (Grid)                 | —        | —            | CBS (Grid)       |
| 50               | 71.3903          | 56.2227  | —          | —       | —         | —          | 125       | 125    | —          | —      | —         | —          | 3625         | 3625   | —          | —      | —         | —          | ST-SPF (Grid)              | —        | —            | ST-SPF (Grid)    |
| 70               | 113.0937         | 104.8742 | —          | —       | —         | —          | 127       | 127    | —          | —      | —         | —          | 5909         | 5909   | —          | —      | —         | —          | ST-SPF (Grid)              | —        | —            | ST-SPF (Grid)    |
| 100              | 143.8834         | 165.8952 | —          | —       | —         | —          | 132       | 132    | —          | —      | —         | —          | 8455         | 8455   | —          | —      | —         | —          | ST-SPF (Grid)              | —        | —            | ST-SPF (Grid)    |
| 150              | 234.6684         | 215.3288 | —          | —       | —         | —          | 132       | 132    | —          | —      | —         | —          | 11791        | 18791  | —          | —      | —         | —          | ST-SPF (Grid)              | —        | —            | ST-SPF (Grid)    |
| 250              | 484.5892         | 331.4795 | —          | —       | —         | —          | 132       | 132    | —          | —      | —         | —          | 18791        | 18791  | —          | —      | —         | —          | ST-SPF (Grid)              | —        | —            | ST-SPF (Grid)    |



Figure 5.12: Large Flying-V Graph



Regarding the Graph-Like environment, neither the CBS, STMS, or ST-SPF performed well. As in the other layouts with a larger size, it is possible to identify that the MAPF rules added to the algorithms caused a considerable degree of complexity to the problem since all the algorithms reached memory overflow at some time. Like the Traditional vertical Layout, the ST-STPF reached memory overflow at 10 agents, the STMS at 7, followed by CBS at 25.

# Chapter 6

## Conclusion and Future Directions

This work performed the implementation of two new algorithms with different focuses, the ST-SPF, focused on MAPF scalability in medium and large crowded instances, and the STMS algorithms focused on reducing Makespan in small and crowded instances. Tests were also carried out with low-level algorithms, exemplifying the efficiency of each in the state-of-the-art warehouse layouts. Finally, a simulator was developed that allows the execution of pathfinding algorithms for single or multi-agents, in addition to the analysis of MAPF problems, where the environment can be modified according to the user's needs.

It is noteworthy that the main objective of the comparative tests is to analyze three essential factors to the MAPF problem:

- The Running Time, that is, the time it took for the algorithm to reach the solution;
- The Makespan, that is, the number of movements required to all agents reach the goal;
- And the Sum of Costs, that is, the sum of movements steps required for each agent to reach the goal;

These factors are essential to achieve maximum throughput in the warehouse since with a lower Running Time, a higher amount of pickups and deliveries can be done since we will have less processing time. While a Smaller Makespan and Sum of Costs allow a better flow in the environment since we will have more spaces available (also avoiding possible "traffic" congestion).

In Chapter 4 the Low-Level algorithms are presented, and the implementations of the ST-SPF and STMS algorithms, explaining in detail the characteristics of each one and their respective advantages and disadvantages. The Chapter 5 presents the results for CBS, ST-SPF, and STMS algorithms in both Grid-Like and Graph-Like environments. In all, three types of layouts with four different sizes were tested, measuring then which type is more viable in a given scenario and which algorithm scaled better. As a result, it is possible to say that the simulator remained stable in all tests, whether in single or multi-agent mode.

The Grid-Like ST-SPF brought good scalability in large and crowded spaces compared to CBS in all scenarios. This result becomes a good implementation for large warehouses with a large number of robots. Despite the drawback of not performing the collision detection of agents outside a swarm, this problem is easily solved by bringing the system to an *online* environment, where agents update their position at any time and can make the request to enter into a swam (swarm appending), solving then this problem.

The Graph-Like STMS algorithms has become a good implementation for small and populous spaces, where even with a considerable number of restrictions it has managed to find a solution, differently from ST-SPF and CBS. This becomes a good implementation for small warehouses, whether those with few or many robots. Also, the STMS algorithms allows a greater Life-Cycle in these environments by reducing the Makespan. This also decreases the need for recharging and the efforts in the mechanical and electrical parts since they will make fewer movements. For spaces of small size using the Traditional Layout, it was also possible to notice that the STMS BFS-Based has the best results since it brings a reduction of the Running Time added with the possibility of lower Makespan when compared with the STMS A \* -Based.

Regarding the type of environment, the Graph-Like brought good results for small spaces, but it brought very high memory consumption for large and complex spaces. In contrast, Grid-Like loses performance considerably in small spaces but has better use of memory in large and complex spaces. In this way, a Graph-Like environment with all MAPF restrictions is best used in small environments (up to 280 nodes/grid points), while a Grid-Like environment with few restrictions becomes better in medium, large, and complex environments (> 700 nodes/grid points). Regarding the type of warehouse layout, it was noted that Traditional Horizontal has better scalability than the others for tiny and small sizes, while Traditional

Vertical found itself better in small and medium sizes. The Flying-V shows good results for the Large size when there are a lot of agents ( $> 150$ ). Was also possible to notice that the Traditional Horizontal has a lower computational cost, while Flying-V allows a considerable reduction in Makespan (in exchange for a small computational cost).

It is also worth mentioning that this work resulted in two publications. The first being the development of a graphical interface, which allows the monitoring of experiments in real-time (Human-Robot Interface) and manipulation of robot variables [7], which can be used later in the *online* ST-SPF for monitoring. The second was a survey that addresses the latest trends and research gaps in the field of Autonomous Mobile Robots applied to Robotic Mobile Fulfillment Systems [13].

Finally, several topics can be addressed in future works. Concerning the simulator, it would be interesting to have a better implementation of the user interaction process, as well as the possibility of integration with ROS or Gazebo, allowing a simulation considering some characteristics of the robot and the environment. Regarding the algorithms, a possible improvement of the ST-SPF is the implementation of it in a *online* environment, solving its current drawback. Besides, it is also interesting to perform tests with other state-of-the-art algorithms, such as RRT\*, and WHCA\* for example. About the STMS algorithms, a point of improvement is the reduction of Running Time, so that it becomes competitive when compared to CBS and ST-SPF.

# Bibliography

- [1] Kenneth B. Ackerman. *Warehousing: Origins, History and Development*. Springer US, Boston, MA, 1990.
- [2] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. pages 420–434. Springer, 2001.
- [3] Dor Atzmon, Roni Stern, Ariel Felner, Roman Barták, Glenn Wagner, and Neng Fa Zhou. Robust multi-agent path finding. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 3:1862–1864, 2018.
- [4] Kaveh Azadeh, René De Koster, and Debjit Roy. Robotized and automated warehouse systems: Review and recent developments. *Transportation Science*, 53(4):917–945, 2019.
- [5] Nikolaos Baras and Minas Dasygenis. An algorithm for routing heterogeneous vehicles in robotized warehouses. *2019 Panhellenic Conference on Electronics & Telecommunications (PACET)*, pages 1–4, 2020.
- [6] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. *Proceedings of the 7th Annual Symposium on Combinatorial Search, SoCS 2014*, 2014-Janua:19–27, 2014.
- [7] I. R. C. Barros, L. F. Costa, and T. P. Nascimento. Turtleui: A generic graphical user interface for robot control. In *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, pages 174–179. IEEE, 2019.

- 
- [8] Italo Barros. Warehousepy. [https://github.com/LASER-Robotics/MRS\\_Warehouse/tree/master/rfms\\_simulator](https://github.com/LASER-Robotics/MRS_Warehouse/tree/master/rfms_simulator), 2021. Online, Accessed: February 31, 2021.
- [9] J. P. Van Den Berg and W. H. M. Zijm. Models for warehouse management: Classification and examples. *International Journal of Production Economics*, 59(1):519–528, 1999.
- [10] Eli Boyarski, Ariel Feiner, Guni Sharon, and Roni Stern. Don’t split, try to work it out: Bypassing conflicts in multi-agent pathfinding. *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*, 2015-Janua:47–51, 2015.
- [11] H. Choset, K. Lynch, and S. et al Hutchinson. *Principles of Robot Motion*. Bradford Books, 2005.
- [12] C. Cosma, M. Confente, M. Governo, and P. Fiorini. An autonomous robot for indoor light logistics. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3:3003–3008, 2004.
- [13] Ítalo Renan da Costa Barros and Tiago Pereira Nascimento. Robotic mobile fulfillment systems: A survey on recent developments and research opportunities. *Robotics and Autonomous Systems*, 137:103729, March 2021.
- [14] Raffaello D’Andrea and Peter Wurman. Future challenges of coordinating hundreds of autonomous vehicles in distribution facilities. *BT - IEEE International Conference on Technologies for Practical Robot Applications, 2008. TePRA 2008*, pages 80–83, 2008.
- [15] René B. M. De Koster, Andrew L. Johnson, and Debjit Roy. Warehouse design and management. *International Journal of Production Research*, 55(21):6327–6330, 2017.
- [16] David Desmet, Robert Boute, and Ann Veerecke. A typology of european distribution centres, 2013. Online, Accessed: February 11, 2020.
- [17] Michael Erdmann and Tomás Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2(1-4):477–521, November 1987.

- 
- [18] Irad Ben-Gal Eugene Kagan, Nir Shvalb. *Autonomous Mobile Robots and Multi-Robot Systems*. John Wiley & Sons Ltd, 2020.
- [19] Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. Adding heuristics to conflict-based search for multi-agent path finding. *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*, 2018-June:83–87, 2018.
- [20] International Organization for Standardization. Robots and robotic devices – Vocabulary. Standard, International Organization for Standardization, August 2012.
- [21] International Organization for Standardization. Mobile robots - Vocabulary. Standard, International Organization for Standardization, March 2017.
- [22] Martin Gebser, Philipp Obermeier, Thomas Otto, Torsten Schaub, Orkunt Sabuncu, Van Nguyen, and Tran Cao Son. Experimenting with robotic intra-logistics domains. *CoRR*, abs/1804.10247, 2018.
- [23] Martin Gebser, Philipp Obermeier, Thomas Otto, Torsten Schaub, Orkunt Sabuncu, Van Nguyen, and Tran Cao Son. Experimenting with robotic intra-logistics domains. *TPLP*, 18(3-4):502–519, 2018.
- [24] Khasha Ghaffarzadeh and Na Jiao. Mobile robots, autonomous vehicles, and drones in logistics, warehousing, and delivery 2020-2040, 2019. Online, Accessed: February 19, 2020.
- [25] E. Giuzzo. Three engineers, hundred of robots, one warehouse. *IEEE Spectrum*, 45(7):26–34, 2008.
- [26] Florian Grenouilleau, Willem Jan Van Hoeve, and J. N. Hooker. A multi-label A\* algorithm for multi-agent pathfinding. *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*, pages 181–185, 2019.
- [27] K. R. Gue and R. D. Meller. A constructive aisle design model for unit-load warehouses with multiple pickup and deposit points. *European Journal of Operational Research*, 236(1):382–394, 2014.

- [28] Robin Hanson, Lars Medbo, and Mats I. Johansson. Performance Characteristics of Robotic Mobile Fulfillment Systems in Order Picking Applications. *IFAC-PapersOnLine*, 51(11):1493–1498, 2018.
- [29] Wolfgang Honig, Scott Kiesel, Andrew Tinka, Joseph W. Durham, and Nora Ayanian. Persistent and Robust Execution of MAPF Schedules in Warehouses. *IEEE Robotics and Automation Letters*, 4(2):1125–1131, 2019.
- [30] Wolfgang Honig, T. K. Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Summary: Multi-agent path finding with kinematic constraints. *IJ-CAI International Joint Conference on Artificial Intelligence*, pages 4869–4873, 2017.
- [31] iCepts Technology Group. Warehouse management basics-zone picking. <https://www.icepts.com/warehouse-management-basics-zone-picking/>, 2016. Online, Accessed: February 19, 2020.
- [32] Zool Hilmi Ismail and Nohaidda Sariff. A survey and analysis of cooperative multi-agent robot systems: Challenges and directions. March 2019.
- [33] N. Vimal Kumar and C. Selva Kumar. Development of collision free path planning algorithm for warehouse mobile robot. *Procedia Computer Science*, 133:456–463, 2018.
- [34] T. Lamballais, D. Roy, and M. B.M. De Koster. Estimating performance in a Robotic Mobile Fulfillment System. *European Journal of Operational Research*, 256(3):976–990, 2017.
- [35] C. K.M. Lee, Bingbing Lin, K. K.H. Ng, Yaqiong Lv, and W. C. Tai. Smart robotic mobile fulfillment system with dynamic conflict-free strategies considering cyber-physical integration. *Advanced Engineering Informatics*, 42(July), 2019.
- [36] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. Task and Path Planning for Multi-Agent Pickup and Delivery of the 18th International Conference on Autonomous Agents and Multiagent Systems. *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2019.
- [37] David Lowe. *The Dictionary of Transport and Logistics*. Kogan Page, feb 2002.



- 
- [38] Hang Ma and Sven Koenig. AI Buzzwords Explained: Multi-Agent Path Finding (MAPF). *AI Matters*, 3(3):15–19, 2017.
- [39] Kamran Mahroof. A human-centric perspective exploring the readiness towards smart warehousing: The case of a large retail distribution warehouse. *International Journal of Information Management*, 45(July 2018):176–190, 2019.
- [40] K V Manjunath and Dr B Ravishankar. Development of Algorithm and Flowchart for the Operation Optimization in Warehouse. *International Journal of Scientific Development and Research*, 1(7):296–313, 2016.
- [41] Riccardo Manzini. Warehousing in the global supply chain: Advanced models, tools and applications for storage systems. *Warehousing in the Global Supply Chain: Advanced Models, Tools and Applications for Storage Systems*, 9781447122746(March 2016):1–483, 2012.
- [42] Marius Merschformann, Lin Xie, and Daniel Erdmann. Path planning for Robotic Mobile Fulfillment Systems. *arXiv e-prints*, page arXiv:1706.09347, June 2017.
- [43] Van Nguyen, Philipp Obermeier, Tran Cao Son, Torsten Schaub, and William Yeoh. asprilo. <https://asprilo.github.io/>.
- [44] Van Nguyen, Philipp Obermeier, Tran Cao Son, Torsten Schaub, and William Yeoh. Generalized target assignment and path finding using answer set programming. In *IJCAI*, pages 1216–1223. [ijcai.org](http://ijcai.org), 2017.
- [45] Nantawat Pinkam, Francois Bonnet, and Nak Young Chong. Robot collaboration in warehouse. *International Conference on Control, Automation and Systems*, 0(Iccas):269–272, 2016.
- [46] Steve Rabin. *AI game programming wisdom 3*. Charles River Media, Boston, Mass, 2006.
- [47] K. Reed and D. Harmelink. What is the difference between distribution centers and fulfillment centers, 2013. Online, Accessed: February 9, 2020.

- 
- [48] Gwynne Richards. *Warehouse Management: A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse*. Kogan Page, nov 2017.
- [49] Geoffrey Rickman. *Roman Granaries and Store Buildings*. Cambridge University Press, apr 1971.
- [50] Debjit Roy, Shobhit Nigam, René de Koster, Ivo Adan, and Jacques Resing. Robot-storage zone assignment strategies in mobile fulfillment systems. *Transportation Research Part E: Logistics and Transportation Review*, 122(November 2018):119–142, 2019.
- [51] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach, Global Edition*. CL, may 2019.
- [52] Oren Salzman and Roni Stern. Research Challenges and Opportunities in Multi-Agent Path Finding and Multi-Agent Pickup and Delivery Problems. *Conference on Autonomous Agents and Multiagent System (AAMAS 2020)*, pages 3–7, 2020.
- [53] Bhavin Shah and Vivek Khanzode. A comprehensive review of warehouse operational issues. *International Journal of Logistics Systems and Management*, 26(3):346–378, 2017.
- [54] Guni Sharon, Roni Stern, Ariel Felner, and Nathan Sturtevant. Meta-agent conflict-based search for optimal multi-agent path finding. *Proceedings of the 5th Annual Symposium on Combinatorial Search, SoCS 2012*, pages 97–104, 2012.
- [55] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.
- [56] B. Siciliano and O. Khatib. *Handbook of Robotics*. Springer, Berlin, 2016.
- [57] Maria Sicola. Commercial Real Estate Terms and Definitions. *The NAIOP Research Foundation. San Francisco, California*, pages 28,29, 2017.
- [58] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza Angeles. *Introduction to Autonomous Mobile Robots*. The MIT Press, Massachusetts, EUA, 2011.

- [59] D. Silver. Cooperative Pathfinding. In *AIIDE Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005. Online, Accessed: July 1, 2020.
- [60] Trevor Standley. Finding optimal solutions to cooperative pathfinding problems. *Proceedings of the National Conference on Artificial Intelligence*, 1:173–178, 2010.
- [61] E. Stefan and S. Stefan. *Heuristic Search: Theory and Applications*. Elsevier, 2012.
- [62] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak. Multi-agent pathfinding: Definitions, variants, and benchmarks. <https://arxiv.org/abs/1906.08291>, 2019. Online, Accessed: June 06, 2020.
- [63] M. Stiefelhagen, K. Van Der Werff, B. Meijer, and T. Tomiyama. Distributed autonomous agents, navigation and cooperation with minimum intelligence in a dynamic warehouse application. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 6:5573–5578, 2004.
- [64] Bryan Stout. Smart moves: Intelligent pathfinding. In *Game Developer Magazine*, April 1998.
- [65] Jirí Švancara, Marek Vlk, Roni Stern, Dor Atzmon, and Roman Barták. Online Multi-Agent Pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:7732–7739, 2019.
- [66] Andrew Tinka, Joseph W Durham, and Sven Koenig. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses Extended Abstract. In *International Conference On Autonomous Agents and Multi-Agent Systems*, pages 1–3, 2020.
- [67] Teun van Gils, An Caris, Katrien Ramaekers, Kris Braeckers, and René B.M. de Koster. Designing efficient order picking systems: The effect of real-life features on the relationship among planning problems. *Transportation Research Part E: Logistics and Transportation Review*, 125(January):47–73, 2019.
- [68] Ko Hsin Cindy Wang and Adi Botea. Tractable multi-agent path planning on grid maps. *IJCAI International Joint Conference on Artificial Intelligence*, pages 1870–1875, 2009.

- [69] Ko Hsin Cindy Wang and Adi Botea. Scalable multi-agent pathfinding on grid maps with tractability and completeness guarantees. *Frontiers in Artificial Intelligence and Applications*, 215:977–978, 2010.
- [70] Wei Wang, Yaohua Wu, Jun Zheng, and Cheng Chi. A comprehensive framework for the design of modular robotic mobile fulfillment systems. *IEEE Access*, 8:13259–13269, 2020.
- [71] Peter R. Wurman, Raffaello D’Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–19, 2008.
- [72] Lin Xie, Hanyi Li, and Nils Thieme. From simulation to real-world robotic mobile fulfillment systems. *Logistics Research*, 12(1):1–13, 2019.
- [73] Tianfeng Xu, Peng Yang, and Huijie Guo. Energy Efficiency Analysis on Robotic Mobile Fulfillment System. *2019 IEEE 6th International Conference on Industrial Engineering and Applications, ICIEA 2019*, pages 145–149, 2019.
- [74] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10(12):399, January 2013.
- [75] Zhe Yuan and Yeming Yale Gong. Bot-in-time delivery for robotic mobile fulfillment systems. *IEEE Transactions on Engineering Management*, 64(1):83–93, 2017.
- [76] Li Zhou, Jing Liu, Xiani Fan, Dongjie Zhu, Pingyu Wu, and Ning Cao. Design of V-Type Warehouse Layout and Picking Path Model Based on Internet of Things. *IEEE Access*, 7(c):58419–58428, 2019.
- [77] Bipan Zou, Xianhao Xu, Yeming (Yale) Gong, and René De Koster. Evaluating battery charging and swapping strategies in a robotic mobile fulfillment system. *European Journal of Operational Research*, 267(2):733–753, 2018.
- [78] E. Zunic, A. Besirevic, S. Delalic, K. Hodzic, and H. Hasic. A generic approach for order picking optimization process in different warehouse layouts. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, May 2018.

- 
- [79] Emir Zunic, Admir Besirevic, Rijad Skrobo, Haris Hasic, Kerim Hodzic, and Almir Djedovic. Design of optimization system for warehouse order picking in real environment. In *2017 XXVI International Conference on Information, Communication and Automation Technologies (ICAT)*. IEEE, October 2017.

# Appendix A

## The WarehousePy Simulator

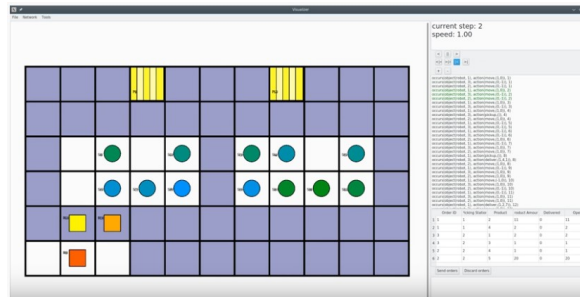
As commented in the contributions and the Chapter 3, one of the difficulties and questions encountered during the development of this framework was: How to perform the tests of Path Planning algorithms, and how to simulate the obstacles and zones of the Warehouse in a way that simulates the problem real? For this problem, several options could be used such as the use and creation of the map through frameworks such as *ROS Gazebo* or *V-REP*, however, these applications generally use *SLAM* to perform locomotion of the robot, making it difficult to create a graph network based on a Grid-type environment for the Path Planning and MAPF analysis. Also, because they are relatively heavy, we can be limited in the capacity of robots within the environment, since the number of robots added to the simulator will reduce the simulation efficiency depending on the computer specifications. A simple way to solve this problem is to simulate a *Grid* environment, where each cell of the map can be assigned to a real-world area, thus representing characteristics such as obstacles, starting point, and desired goal. This modification makes the simulation relatively light, allowing to be repeated on any other machine as long the necessary packages are installed.

In Chapter 2 was presented the most common types of map and environment representation, but let's review some important points:

- The map must be similar to the real environment;
- The accuracy of the map and its peculiarities must be represented;
- The Map Representation complexity has direct relation to the computational complexity of tasks such as location, navigation, and mapping.

With the review of the state-of-the-art carried out in Chapter 3 it was possible to notice that the articles that made some contribution in the area of Path Planning for robotic applications, or that simulated the behavior of the robot inside the Warehouse they also usually did it through a *Graph-like* or *Grid-like* environment, some examples were the contributions found in Section 3.3. Some of the available simulators work with a Grid-like environment, one example is *asprilo*, characterized as a benchmarking framework to study typical scenarios in intra-logistics and warehouse automation with multiple mobile robots (see Figure A.1). Despite the interesting proposal, the application is more focused on the concept of intra-logistics and TAPF (Combined Target Assignment and Path Finding), that is, **the simulator is focused on the steps involved in handling materials with multiple robots, such as the number of tasks that must be performed, and not in the implementation of a framework that performs the global or local planning of the robots** [22], [23], [44]. The simulator has the possibility to connect with other simulators, either to provide instantiating and order plans or to get an instance from the visualizer and returns a valid plan to the visualizer [43].

Figure A.1: The asprilo Simulator



Considering the gap both in the field of mobile robotics and in the MAPF problem, a simulator was developed through the *PyGame* framework, where it is possible to create the desired environments by defining obstacles, goals, start positions, workers locations, recharge zones, pickup zones, and treadmill zone. The simulator allows the creation of environments through simple graphs, weighted graphs, and grids, where the user can program his search algorithm and use the one that best benefits him. A detail that must be emphasized, however, is that since an RFMS robot can move only in 4 directions (Top, Bottom, Right, Left), the simulator uses this as a constraint, but the user can easily modify this on the source code,

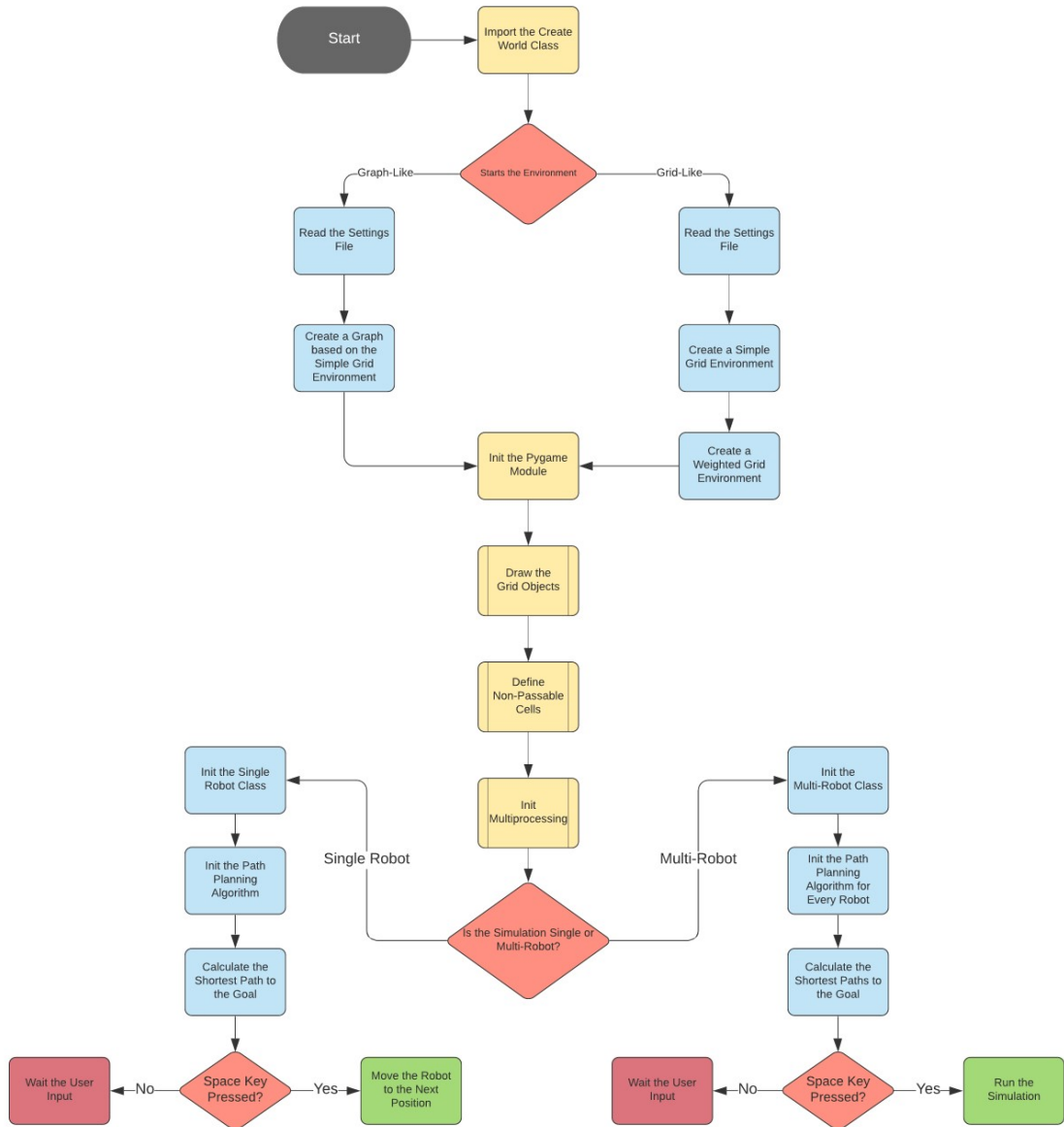
with is available at [8].

The desired environment settings can be made in the *Settings* file, where the user must define the object's locations with a matrix representation (a.k.a. Undirected Adjacency Matrix), the size of the desired nodes, and the maximum size of the grid (length and height). In this same file, it is also possible to create new environments, whether warehouse-like or not. The user can modify or add the robot's starts positions, their goals, the recharge locations, and all the other objects present in the simulator. It is also possible to add new objects if necessary, but this requires the addition of the object's function to the *createWorld* simulator class, which is responsible to translate the settings to a visible environment.

Through the *createWorld* simulator class, the user can select if he wants to run a Single Agent Path Planning Algorithm or a Multi-Agent Path Finding Algorithm calling the classes *MultiRobot* or *SingleRobot* and inputting the paths calculated by the user path planning algorithm. These classes will abstract the paths to the PyGame format and create the robot agents on the simulator. In Figure A.2 it is possible to view a flowchart that demonstrates how the simulator high-level principle of work.



Figure A.2: The WarehousePy Algorithm Flowchart



Using the methodology explained above, and considering the types of layouts covered in Subsection 2.2.2, the Traditional Horizontal, Traditional Vertical, and Flying-V environments were created in both Grid and Graph formats. Finally, for a better understanding of the scalability of the algorithms, four sizes were also implemented for each mentioned environment as shown in Figure A.3 and the details for each environment size can be viewed below:

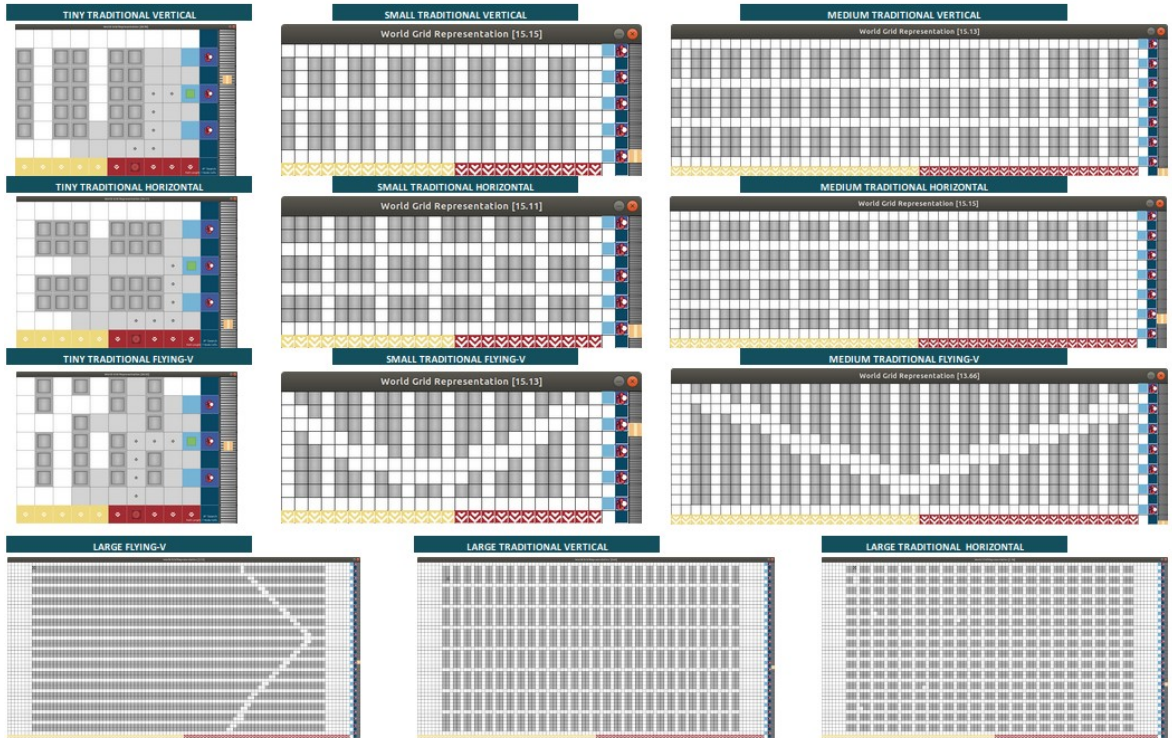
- **Tiny:** 18 W x 8 L in size, 128 states and maximum of 11 robots;

- **Small:** 28 W x 10 L in size, 260 states and maximum of 25 robots;
- **Medium:** 50 W x 14 L in size, 672 states and maximum of 48 robots;
- **Large:** 100 W x 50 L in size, 5000 states and maximum of 98 robots;

Where the maximum number of robots for each environment size was defined by the Equation A.1:

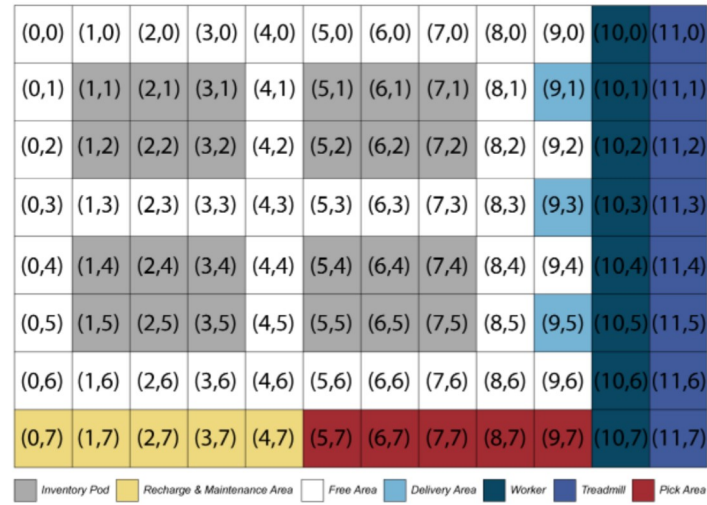
$$Agents = No.ofRechargeZones + No.ofPickupZones + 1 \quad (A.1)$$

Figure A.3: Warehouse Layouts implemented in WarehousePy



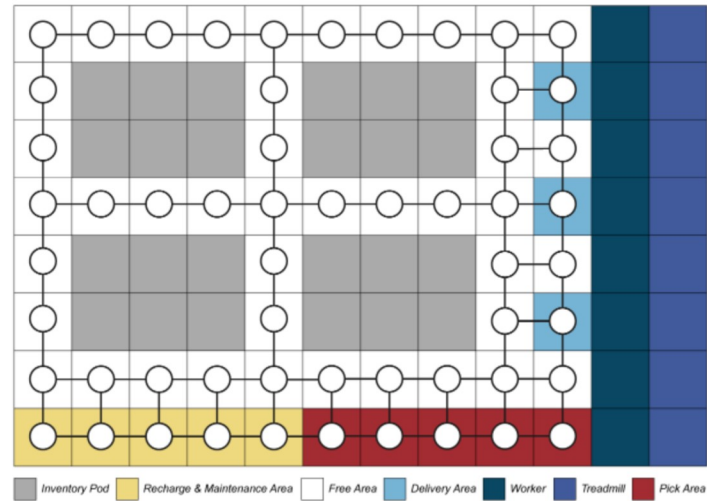
Let's unravel the environments above in a way that the simulator's operation can be better understood using the Tiny Horizontal environment (Figure A.4) as example. In our tiny layout we have a Grid-like environment where each cell has a specific matrix coordinate. In white, we have the coordinates with the free space available, in gray the static obstacles (in this case the Warehouse Pods), in yellow the cells responsible for the recharge zone, in red the cells responsible for the Pick zone, in light blue the delivery points, dark blue the location of the workers, and light purple the location of the treadmill.

Figure A.4: The Tiny Traditional Horizontal Grid Environment



Through this matrix the simulator will read the values and translate the desired environment, this is made identifying each cell as the desired Undirected Adjacency Matrix and transforming their in a connected grid environment as shown in Figure A.5.

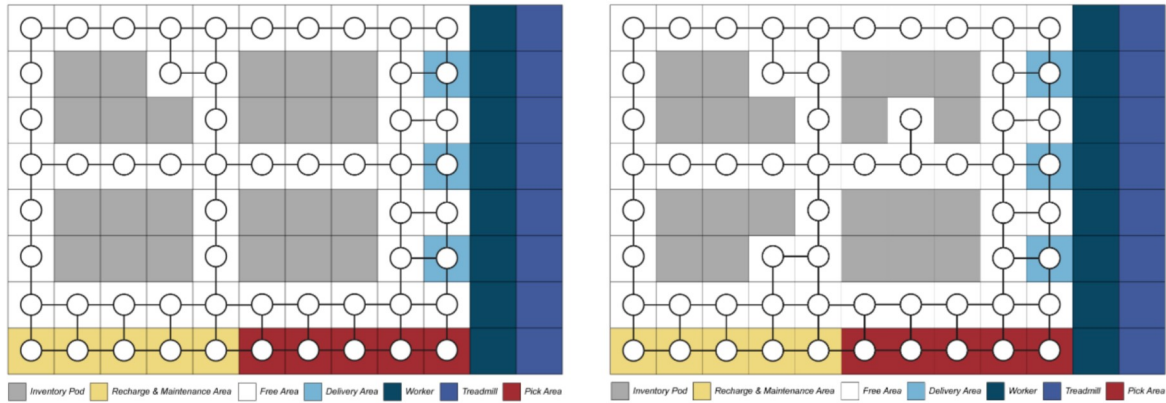
Figure A.5: The Tiny Traditional Horizontal Environment Connections



Through the figure above, the following question can be made: Why the workers and treadmill cells was not added to the graph by the simulator? One of the main reasons is the function **is\_passable** found in the file responsible for generating the world (see the source code at [8]). The function excludes the cells matrix that should not be visited by the robot, leading to the exclusion of the grid node. This is an important factor for the RFMS system

in particular since workers will not walk around the warehouse and will hold their positions waiting for the robot deliver the product, acting then like static obstacles (the same goes for the treadmill). A detail that should be emphasized is that in the Grid-like environment, the robot will avoid going through the pods (obstacles). However, it will go to the position if it is the final destination (goal), so for this reason, the grid connectivity for these respective nodes are not being represented. The obstacles (Pods) can also be removed after starting the simulator (Right Mouse Click), where the environment will be updated with the new free space available. This possibility becomes interesting for analyzing the desired Path Planning algorithm or planner since it can be programmed to recalculate the path (route) if a new obstacle appears. In Figure A.6 is possible to see the Grid connectivity change after removing some obstacles.

Figure A.6: Examples of Obstacle Removal after the Initialization

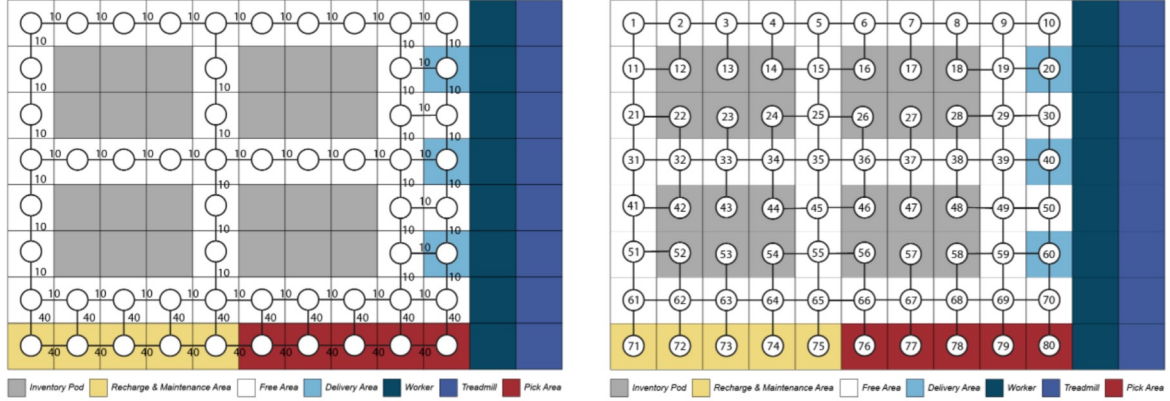


The simulator also allows the creation of a Weighted Grid Environment. This modification can be done through the main program function, where the user simply calls the type of graph he wants to create. The implementation of this graph in the software level is relatively simple, using **Object Orient Programming (OOP)** the program performs the *Class Inheritance* of the Simple Graph Class, receiving the values stored in the *Settings* file and the non-desired positions. After that, the program executes the path costs for every connected node being able to be used in Weighted Algorithms like the A\*. This type of graph was specifically chosen for our experiments, since the Recharge and Pickup zones must not be visited by the robot during the free search since the robot should only go to these zones when they are the destination. In Figure A.7 is possible to see the modifications made by the



Simulator when the Weighted Grid is used.

Figure A.7: Tiny Traditional Horizontal Graph & Grid Environments



(a) The Grid Environment

(b) The Graph Environment

Considering the previous explanations, the question may arise: what is the difference of this type of environment compared to the Graph-like? The only difference is the data structure that will be used by the path planning algorithm. For this type of environment (see Figure A.7), the simulator will perform the translation of the nodes in Grid format to a Graph of integers, and perform the calculation of the adjacency list and adjacency matrix. In this way, the algorithms will use these new structures to carry out the planning, which considerably reduces the Running Time.

## A.1 Discussion and Final Considerations

The same process used in the creation of the Tiny Traditional Horizontal environment can be replicated for the others at Figure A.3, the only variables that will change are: The Environment Length (integer), The Environment Width (integer), and the Pods Location (list of tuples). With the modifications of these values in the *settings* file, the recharge, pickup, treadmill, delivery and work positions zones will be modified automatically. The change process is manual, where the user only has to make the modification. It is also interesting to comment that the purpose of this simulator, besides is used here to simulate the RFMS system, is also to provide an Open Source simulator for other researches or applications that involve the study of path planning algorithms for single or multiple robots. The relation-

---

ship between the pixels and the real environment allows a more realistic simulation since the relation between *pixels*  $\times$  *real distance* is respected.