Tampere University

Robert Heikkilä

DISTRIBUTED MULTI-AGENT PATHFINDING IN HORIZONTAL TRANSPORTATION

Master's Thesis
Faculty of Engineering and Natural Sciences
Examiners: Asst. prof. Azwirman Gusrialdi
Prof. Matti Vilkko
April 2023

ABSTRACT

Robert Heikkilä: Distributed multi-agent pathfinding in horizontal transportation
Master's Thesis
Tampere University
Degree Programme in Automation Engineering
April 2023

Horizontal transportation in maritime container terminals plays a crucial role in ensuring safe, efficient, and cost-effective operations. Heavy working machines, such as straddle carriers, trucks, and automated guided vehicles, transport containers between cranes, creating complex routing problems known as multi-agent pathfinding (MAPF) problems. Existing solutions may not adequately address the unique challenges presented by container terminals, necessitating the development of new algorithms.

This thesis aims to develop and demonstrate a distributed MAPF algorithm for horizontal transportation in container terminals. The MAPF problem is first formulated as a binary linear programming (BLP) model by expressing the actions in the container terminal using a directed pseudograph. Optimal solutions are obtained using PYOMO, an open-source Python-based optimization software. The Augmented Lagrangian, a graph pathfinding algorithm, and a stochastic element are then employed to create a sub-optimal, distributed algorithm.

The developed algorithm is evaluated against an optimal solution and a reference method that prioritizes calculating the path for one agent at a time while taking into account previously calculated paths. A simulator is set up to emulate horizontal transportation in a maritime container terminal, by modeling the terminal as a graph in MATLAB. In the simulator, MAPF algorithms are applied in combination with a high-level coordinator assigning destinations.

The experimental part of this thesis investigates the trade-off between solution time (iterations) and solution quality by tuning algorithm parameters and evaluating the performance of the distributed algorithm in comparison to the priority-based method under two different map layouts, particularly addressing the presence of a bottleneck. The results demonstrate the need to adapt the algorithm's parameters and strategies according to specific environments and map layouts, to ensure good performance across various scenarios. The main contribution of this thesis lies in the development of a adaptable, distributed MAPF solution that can ultimately address diverse scenarios and environments.

Keywords: multi-agent pathfinding, container terminal, integer programming, graph theory

TIIVISTELMÄ

Robert Heikkilä: Useiden maassa liikkuvien työkoneiden hajautettu reitittäminen satamassa
Diplomityö
Tampereen yliopisto
Automaatiotekniikan DI-ohjelma
Huhtikuu 2023

Merikonttiterminaaleissa konttien vaakatasoinen kuljettaminen on keskeinen tekijä turvallisen, tehokkaan ja kustannustehokkaan toiminnan varmistamisessa. Raskaat työkoneet, kuten konttilukit, kuorma-autot ja automaattisti ohjatut ajoneuvot (AGV:t), kuljettavat kontteja nosturien välillä, luoden monimutkaisia reititysongelmia, joita kutsutaan monitoimija-reitinhaku (MAPF) -ongelmiksi. Olemassa olevat ratkaisut eivät riittävästi käsittele konttiterminaaleille asetettuja erityisiä haasteita, mikä edellyttää uusien hajautettujen MAPF-algoritmien kehittämistä.

Tämän diplomityön tavoitteena on kehittää ja esitellä hajautettu MAPF-algoritmi vaakasuuntaiseen kuljetukseen konttiterminaaleissa. MAPF-ongelma mallinnetaan ensin binääriseksi lineaariseksi ohjelmointimalliksi (BLP), jossa konttiterminaalissa liikkuminen mallinnetaan suunnattuna pseudograafina. Optimaalisia ratkaisuja tutkitaan käyttämällä PYOMO-ohjelmistoa, joka on avoimen lähdekoodin Python-pohjainen optimointiohjelmisto. Tämän jälkeen laajennettua Lagrangen kertoimien menetelmää, graafin polunetsintä algoritmia ja stokastististа elementtiä käytetään luomaan lähes optimaalinen, hajautettavissa oleva algoritmi.

Kehitettyä algoritmia arvioidaan vertaamalla sitä optimaaliseen ratkaisuun ja viite-menetelmään, joka priorisoi polkujen laskemista yhden agentin kerrallaan ottaen huomioon aiemmin lasketut polut. Simulaattori rakennetaan jäljittelemään vaakasuuntaista kuljetusta merikonttiterminaaleissa, mallintamalla terminaali graafina MATLABissa. Simulaattorissa MAPF-algoritmeja sovelletaan yhdessä korkean tason koordinaattorin kanssa, joka määrittelee määränpäät.

Diplomityön kokeellisessa osuudessa tarkastellaan ratkaisuajan (iteraatioiden) ja ratkaisun laadun välistä tasapainoa kokeilemalla erilaisia parametreja ja arvioimalla hajautetun algoritmin suorituskykyä verrattuna prioriteettipohjaiseen menetelmään kahden erilaisen karttapohjan päällä, keskittyen erityisesti pullonkaulan lisäämisen tuomiin vaikutuksiin. Tulokset osoittavat, että algoritmin parametrien ja strategioiden mukauttaminen erityisiin ympäristöihin ja karttapohjiin on tarpeen, jos halutaan varmistua tulosten olevan mahdollisimman lähellä optimaalisuutta erilaisissa skenaarioissa. Tämän diplomityön pääasiallinen kontribuutio on joustavan, hajautetun MAPF-ratkaisun kehittäminen, jolla voidaan tulevaisuudessa käsitellä erilaisia tilanteita ja ympäristöjä.

Avainsanat: reittioptimointi, konttisatama, kokonaislukuohjelmointi, graafiteoria

PREFACE

Writing this thesis has been a long, rewarding process. I have been fortunate enough to receive consistent guidance, which has helped me to stay on the right path. Specifically, I want to thank prof. Matti Vilkko for the research question, Asst. prof. Azwirman Gusrialdi for consistent feedback on the writing process and Msc. Teijo Juntunen for mentoring.

This work is part of the Business Finland Co-innovation research project FEMMa.

Tampere, 26th April 2023

Robert Heikkilä

CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

G	Graph (Graph theory)
V	A set of graph vertices
A*	An optimal pathfinding algorithm that finds the shortest path between two points in a graph using a heuristic function.
AGV	Automated guided vehicle
ALV	Automated lifting vehicle
ASC	Automated yard crane
ASP	Answer set programming
AStCs	Automated straddle carrier
BLP	Binary linear programming
CBS	Conflict based search
COP	Constraint optimization problem
CP	Constraint programming
FEU	Fourty-foot equivalent
HT	Horizontal transportation
ICTS	Increasing cost tree search
ID	Independence detection
ILP	Integer linear programming
KKT-conditions	Karush-Kuhn-Tucker conditions
A	A set of agents
E	A set of graph edges
MAPF	Multi-agent pathfinding
MPP	Multi-robot path planning
NP-hardness	A concept in computational complexity theory, which refers to the computational difficulty of solving problems in the complexity class NP.
QC	Quay crane
RMG	Rail-mounted gantry

RTG	Rubber-tyred gantry
SAT	Boolean satisfiability
SC	Straddle carrier
STS	Ship-to-shore
TEU	Twenty-foot equivalent
TOS	Terminal operating system
YC	Yard crane

1. INTRODUCTION

As a vital component of the container transportation network, *maritime container terminals* serve as a critical link in the global supply chain. With the ever-growing complexity of modern economies, the ability to efficiently export and import containerized goods has become increasingly important. From sophisticated machine parts and electronics to basic commodities such as agricultural produce, a wide variety of goods are transported in these ubiquitous metal containers. Often hailed as the driving force behind globalization, containerization has revolutionized international trade. For example, a recent study by (Michail, Melas, and Batzilis 2021) attempts to establish a connection between global gross domestic product (GDP) and containerization.

Rising container volumes have led to congestion issues in many terminals, prompting the adoption of automation as an economically viable solution. Automation can significantly enhance terminal efficiency and consistency while reducing labor costs. A recent study by (B. Kim, G. Kim, and Kang 2022) examined the throughput, number of ship arrivals, and berthing time at four ports (Rotterdam, Los Angeles/Long Beach, Qingdao, and Yangshan) operating both fully and non-fully automated container terminals during the COVID-19 pandemic. The results indicated better performance for fully automated terminals across all selected indexes. It is reasonable to assume that automation will continue to gain traction in ports worldwide, with the number of semi or fully automated terminals estimated to be 63 in 2022 (Knatz, Notteboom, and Pallis 2022).

In most ports, designated rubber wheel machines, such as straddle carriers, are used to transport container horizontally between cranes. Efficient and robust route planning for these machines represents a crucial sub-problem in port automation. Recent studies have directly addressed this issue, such as (Hu et al. 2021), which employed reinforcement learning to plan paths for automated guided vehicles (AGVs) in terminal environments, and (Tang et al. 2021), which introduced a graph-based geometric A-star algorithm for AGV pathing in terminals. Similar problems are often referred to as multi-agent pathfinding (MAPF) or multi-robot path planning (MPP) problems. Numerous general MAPF methods could be applicable in maritime container terminal settings, some of which are introduced in Section 3.3.

This thesis aims to develop and assess the use of a distributed MAPF algorithm to address the outlined problem. The hypothesis is that a distributed algorithm would offer better scalability and, consequently, faster computational times. The method for the development of a distributed MAPF algorithm is inspired by (Nishi, Ando, and Konishi 2005), where an augmented Lagrangian decomposition and coordination technique are employed to route AGVs in semiconductor fabrication bays in a distributed manner. In this approach, individual machines determine the optimal route by communicating with each other, enabling the use of parallel computing. Numerous other studies have investigated distributed approaches to MAPF problems in various settings, such as (Z. Ma, Luo, and H. Ma 2021), which employed deep Q-learning to develop a heuristic algorithm where agents cooperate via graph convolution. Overall, the topic of distributed MAPF remains highly relevant as no decisively effective solution has been presented thus far.

The structure of this thesis is organized into several chapters. Chapter 1 provides an introduction to maritime container terminals and their significance in the global supply chain. Chapter 2 delves into the details of intermodal container shipping, container handling equipment used in horizontal transportation, and the layout of maritime container terminals. Chapter 3 introduces the concept of multi-agent pathfinding (MAPF) and its relevance to container terminal operations, along with a brief survey of existing methods for solving the MAPF problem. Chapter 4 presents the formulation of the binary linear programming (BLP) model and discusses both optimal and suboptimal approaches to the MAPF problem, with a focus on the development of a distributed algorithm. In chapter 5 simulations are carried out to evaluate the performance of the proposed algorithm on a bottleneck-free and bottleneck map. Finally, Chapter 6 offers a conclusion, summarizing the key findings of the thesis and discussing potential avenues for future research.

2. MARITIME CONTAINER TERMINALS

A *maritime container terminal* is an important connection hub of land and water logistics. Vessels, trains and trucks arrive at the maritime container terminal, where they are unloaded and/or loaded with containers. The purpose of this chapter is to first provide an overview of container shipping in general, then emphasize the elements that are relevant to the thesis, such as horizontal transportation and the equipment involved in it. Lastly typical layouts for maritime container terminals are introduced. Most importantly, reading this chapter should provide sufficient information about horizontal transportation in maritime container terminals, the case for which the routing algorithms in this thesis are developed and tested.

2.1 Intermodal container shipping

Containers emerged in the 1960s and have become the unit of transport in integrated transportation systems. A *container* is essentially a standard size transportation cuboid. Maritime containers are often made of steel and are stackable. Container sizes are measured in *twenty-foot equivalent units* (TEU), which is the volume of one 20 ft long, 8.5 ft high and 8 ft wide container. In the maritime industry, a more popular container is a longer 40 ft container, with a volume of 2 TEU or 1 FEU (*fourty-foot equivalent unit*). The different attributes of the two container types are summarized in figure 2.1.

It is important to note that there are many types of containers that have the same external dimensions previously mentioned. The five main types are:

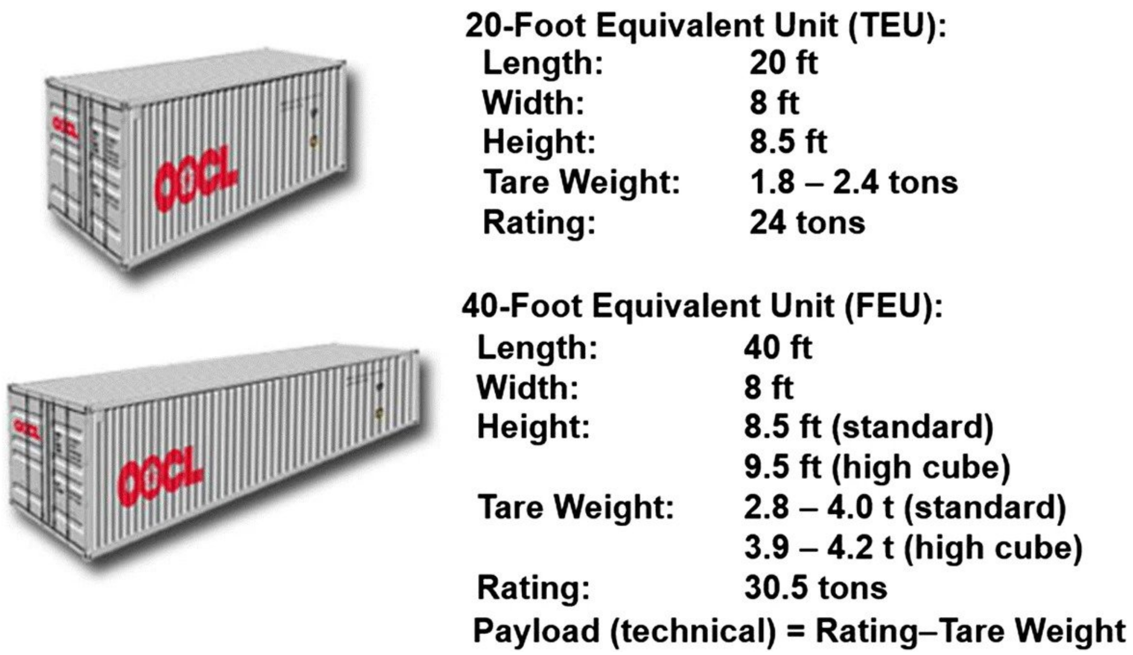
- **Standard container** - also known as general purpose containers. Standard containers are mainly used in 20 ft or 40 ft sizes and are loaded and unloaded through a double door.They are designed for non-specific dry cargo and can withstand harsh conditions. Standard containers are often watertight and can handle temperatures ranging from -40 °C to 70 °C. The fully enclosed rigid structure of the standard container makes it ideal for stacking operations.
- **Tank container** - a container designed to carry liquid goods such as fuel, vegetable oils and fruit juices. Usually either some sort of chemicals or foodstuff that are uneconomical to transport in bulk carriers due to small quantities or location. A tank container is essentially a tank capsule fitted inside a metal cuboid structure.

It is loaded from the top and has the same dimensions as the standard container. Modified versions of the tank container include insulation and temperature control.

- **Open top container** - a container that has no fixed top. Instead a removable tarpaulin is usually used as a roof. Open top containers are often used for dry cargo transportation that is too large to be fitted through a standard containers double doors. The open top also allows for faster loading of small goods. Most open top containers have the same dimensions as a standard container. They are also usually stackable unless cargo height exceeds container height. They are not, however, as structurally solid as standard containers.
- **Flat container.** - a container that has open top and sides. The floor-structure is reinforced for heavy cargo and end walls are either fixed or collapsible. Flat containers are designed to carry heavy or oversized cargo. The end walls are usually stable and strong enough to support the stacking and securing of multiple flat containers on top of each other. The dimensions also match standard containers. Unlike standard containers, however, the flat containers cargo is unprotected from the elements.
- **Refrigerated container** - also known as *reefers*. An insulated container equipped with a refrigeration unit. They are used to transport temperature-controlled cargo, often acting as a moving freezer. Reefers require outside power while in transport. Some reefers have integrated back-up generators for unexpected long cuts in supplied power. Special attention is required in operations as to not cut power for too long. The refrigerated container also have the outside dimensions of a standard containers and are stackable.

Containerization refers to the increasing use of containers as a means of transporting cargo. The main advantages of containerization are flexibility and scalability. A container is essentially a small warehouse that can be transshipped between cities, countries or even continents. A container will keep the cargo safe both from elements and outsiders. The unified ISO-standard size of most containers makes terminal, machine and vessel design significantly easier. Some challenges related to containerization include large land footprint of terminals due to limited stacking height, empty containers not being where the cargo is and high infrastructure and machine costs. (Rodrigue 2020)

In a maritime setting, most containers are transported by liner operators. Liner operators such as Maersk, MSC and COSCO operate a large fleet of vessels. These vessels mainly navigate between global trade hubs using increasingly well established international trade routes. Vessel sizes have been steadily increasing overtime. The better economics of large containers ships are mainly a result of increased fuel efficiency and reduced port calls. Maritime container terminals have to keep up in handling capacity to be able process large vessels efficiently. Both the terminal and liner operators aim for short transit



20-Foot Equivalent Unit (TEU):

Length:	20 ft
Width:	8 ft
Height:	8.5 ft
Tare Weight:	1.8 – 2.4 tons
Rating:	24 tons

40-Foot Equivalent Unit (FEU):

Length:	40 ft
Width:	8 ft
Height:	8.5 ft (standard)
	9.5 ft (high cube)
Tare Weight:	2.8 – 4.0 t (standard)
	3.9 – 4.2 t (high cube)
Rating:	30.5 tons

Payload (technical) = Rating–Tare Weight

Figure 2.1. *20- and 40-foot standard containers and their technical characteristics. (Haralambides 2019)*

times and maximal container flow. This requires co-operation between terminal and liner operators, well before the vessels arrival. (*Review of Maritime Transport* 2021)

Intermodal container shipping includes many modes of transport. The main mode of transport is the maritime container vessel, but in order to reach the hinterlands, trains and trucks are used. Between transportation modes, there are hubs, that transfer containers from one mode to another. For example, a maritime container terminal is tasked with the unloading and onloading of maritime vessels as well as receiving and sending off containers via trucks and trains. There are also different type of container terminals, such as the inland rail container terminal, where containers are transferred between trains and trucks. The overall flow of containers in intermodal container logistics is heavily dependent on the operation of all it's container terminals, as they are naturally the bottle necks of the transportation network. (Rodrigue 2020)

2.2 Container handling equipment in horizontal transportation

Containers are designed to be easily moved. For example, the corner locking points allow for firm lifting and carrying. There exists a multitude of equipment aiming to achieve these two basic tasks. In terminal setting, we can make a distinction based on direction of container movement. Cranes are mainly responsible for vertical handling and machines such as trucks handle most of the horizontal movement. There is also a hybrid category of machinery that can handle both directions with limitations (straddle carriers etc.). This section provides an introduction and some illustration on the equipment used.

2.2.1 Quay cranes

Arriving vessels moor at a quayside berth, where the ship is unloaded/loaded by *quay cranes* (QCs). A quay crane can also be referred to as a ship-to-shore crane (STS), but note that the container flow goes in both directions. Due to their heavy weight, QCs move on railtracks mounted on the quay wall. Sophisticated spreaders are used to grapple and hoist containers from the ship to quayside wall and vise versa. The productivity of a QC is indicated by the number of containers moved per hour. This productivity number also defines an upperbound for container flow through the terminal, and as a result quay cranes are one of the first things to consider in terminal planning. A schematic of a STS crane can be seen in figure 2.2. (Meisel 2009)

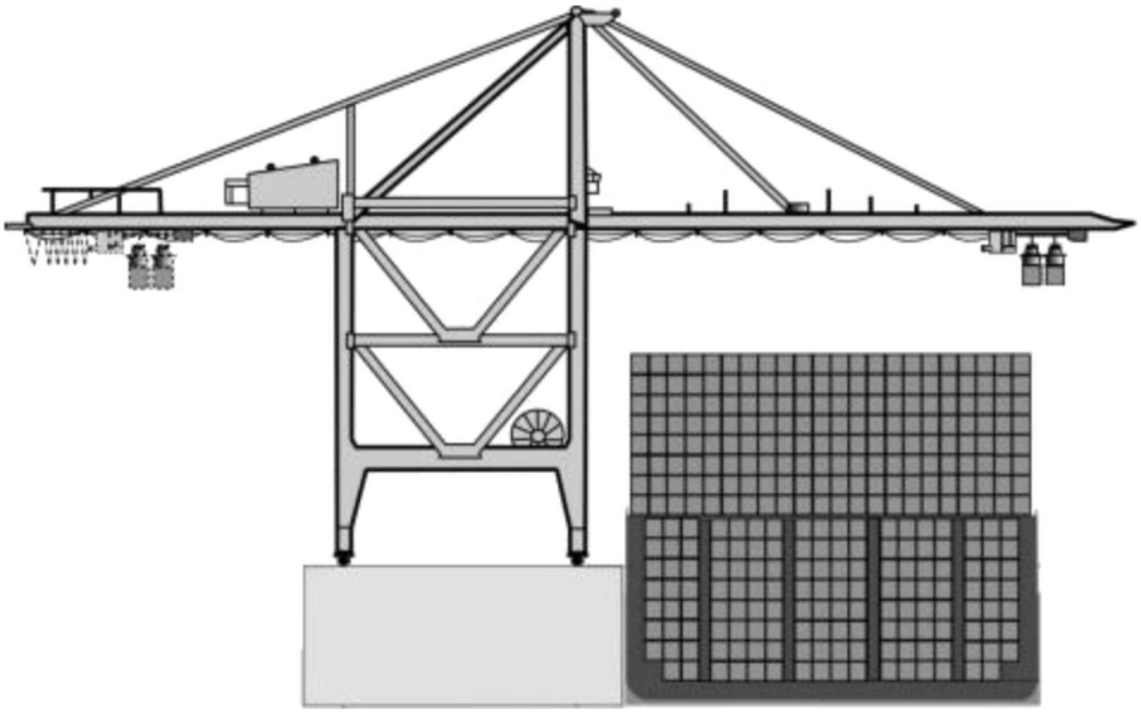


Figure 2.2. *STS/QC crane (Digiesi, Facchini, and Mummolo 2019)*

The size of QCs and ships go hand in hand. A large ship can only be served by a large QC, which in turn is a only a good investment for the terminal if there are going to be large ships to serve. It is natural to classify QCs based on the size of ship they can serve. There are 3 ocean ship categories, and subsequently 3 categories of QCs:

- **Panamax QC** - able to serve ships that fit through the Panama canal. This equates to ships that have a maximum storage width of 11 - 13 containers. A crane of this category has an outreach of 30-40m.
- **Post-Panamax QC** - able to serve ships that cannot fit though the Panama canal, limited to ships with a maximum storage width of 17-19 containers. A crane of this category has an outreach of 45-55m.

- **Super Post-Panamax QC** - able to serve current larges ships with a maximum storage width of 21-23 containers. A crane of this category has an outreach of 60-70m.

The categories fluctuate as vessel sizes keep growing, but the idea is that QCs are often categorized along with vessels. Besides size, quay cranes can differ by the number of hoists, spreader type and lifting capacity. In maritime container terminals, cranes typically have one or two hoists, that are lifting either a regular spreader or a tandem spreader. A speader is a device that is used to grapple containers. The mechanics involve the use of twistlocks, that are locked/unlocked to the top corner fittings of containers. A regular spreader can grapple either one 40 ft container or two adjacent 20 ft containers. A tandem spreader is essentially two regular spreaders attached together, and can grapple, for example, a 40 ft container and two 20 ft containers simultaneously. (Bartošek and Marek 2013)

2.2.2 Yard cranes

It is often necessary to stack containers on land. For this purpose a *yard crane* (YC) is needed. A storage block is usually associated with one or two yard cranes. The yard cranes are tasked with the intelligent reception and dispensation of containers in and out of the blocks. The most prevalent yard cranes are *gantry cranes* such as rubber-tyred gantry (RTG) crane and rail-mounted gantry (RMG) crane. The structure of yard cranes is very different from quay cranes, as they are designed to arc over the containers, as apposed to reaching for them via an appendage. A RTG can span up to 8 container rows and a RMG even more. Both can usually stack piles up to 6 containers high, although the last layer usually remains empty, to allow the passage of containers. A schematic of a yard crane can be seen in 2.3. (Meisel 2009)

Given a large enough number of containers, the effectiveness of RTGs and RMGs depends heavily on scheduling. The cranes have to scheduled in a way as to minimize unnecessary shuffling of containers. Scheduling is also a kind of prerequisite to yard crane automation. Automated yard cranes are referred to as automated stacking cranes (ASC). ASCs are currently almost exclusively RMGs, as rails provide a higher level of predictability. The decision as to where to stack the container has to be made within seconds of arrival. It is often not possible to calculate optimal solutions, but rather some heuristic is used. Some yard planners also specifically prefer straightforward stacking strategies that might be made by the terminal operating system (TOS) but can be easily supervised. State-of-the-art yard crane scheduling and stacking is was recently examined in (Kemme 2020).

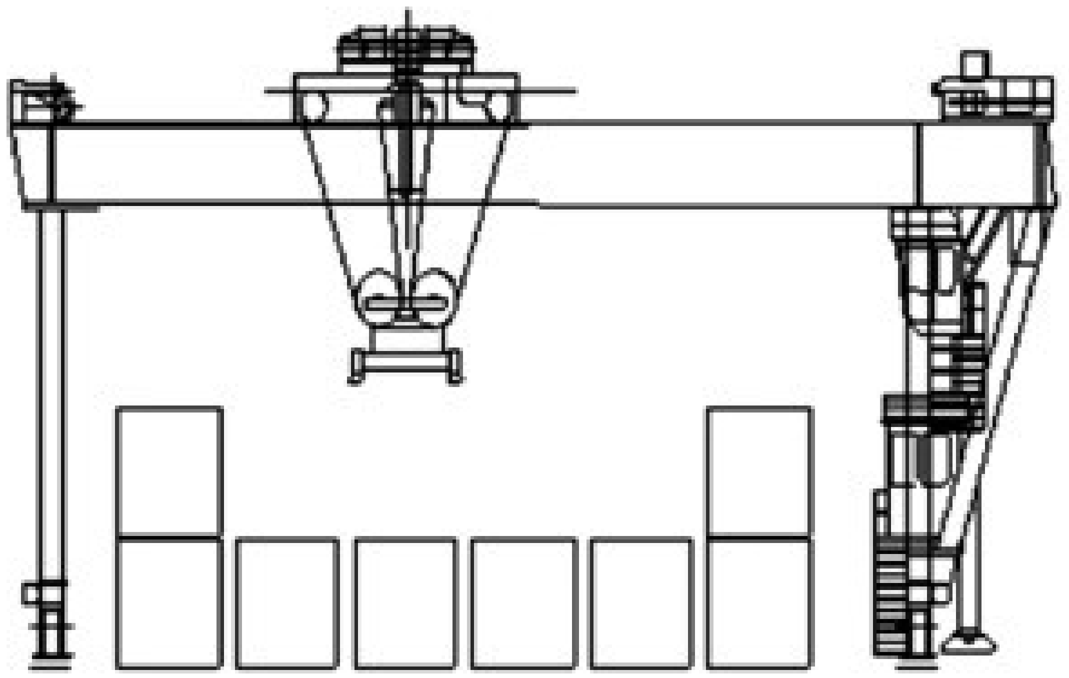


Figure 2.3. *Schematics of a yard crane. (Carlo, Vis, and Roodbergen 2014)*

2.2.3 Straddle carriers and AGVs

Horizontal transportation is needed to move containers from quayside cranes to yard cranes and vise versa. This can be achieved with or without the help of cranes. *Straddle carriers* (SC) are an example of a machine that can independently lift a container from the ground and move it anywhere on the pavement. *Automated guided vehicles* (AGVs) on the other hand require crane assistance on both ends of the journey. A variation of an AGV called the *lift*-AGV can lift the container independently with the use of *rack*. Schematics of a straddle carrier and AGV can be seen in 2.4. Internal trucks are not introduced, as the simplifications in the thesis would reduce them into AGVs.

In (Anvari et al. 2020), it is observed that straddle carrier-type machines, both manned and unmanned, are referred to by various names, such as *Shuttle Carrier/AutoShuttle* (Kalmar) and *BoxRunner/A-Sprinter* (Konecranes). Within automated terminal research, three primary categories of machines capable of independently lifting containers are identified: *automated lifting vehicles* (ALVs), *automated straddle carriers* (AStCs), and *lift*-AGVs.

It might seem logical to use ALV as a comprehensive term for all these machines; however, this would be *incorrect*. Research in this area may also present contradictions. In (H. Yu et al. 2022), lift-AGVs are likely classified as ALVs, whereas in (Kumawat and Roy 2020), they are distinctly considered as a separate type of machinery. For the purpose of this thesis, we will assume that ALVs encompass straddle carrier-type machines capable of lifting containers from above.

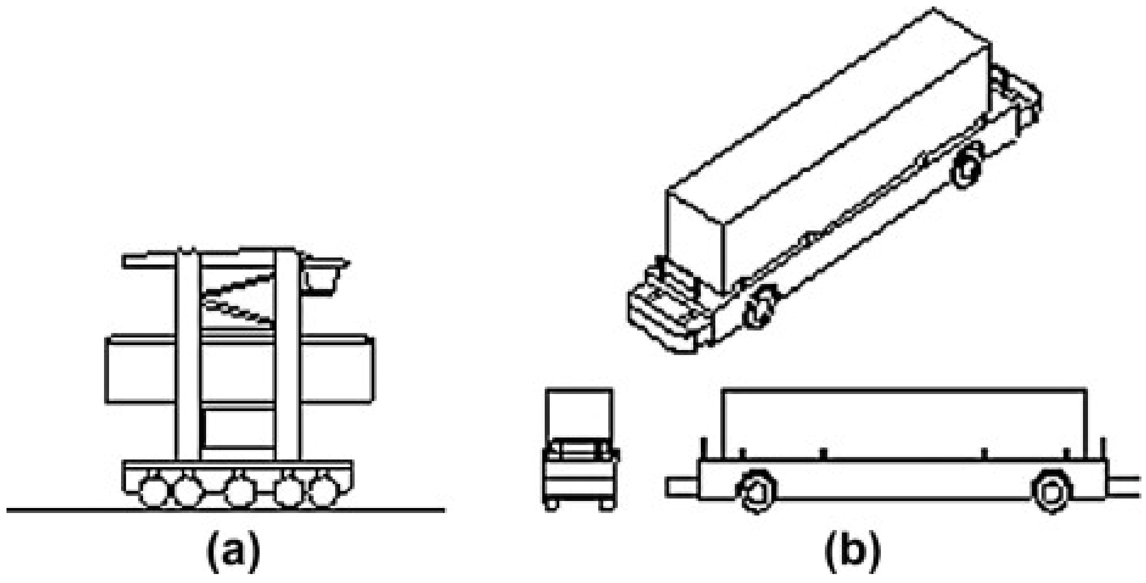


Figure 2.4. *Straddle carrier (a) and AGV (b). (Carlo, Vis, and Roodbergen 2014)*

The decision between crane-*dependent* and crane-*independent* equipment is more nuanced than one might initially assume. In (Bae et al. 2011), AGVs were compared to ALVs, revealing that AGVs struggle to meet the demands of highly productive QCs, even when additional vehicles are employed. This issue may arise due to increased waiting times at the ACS as more vehicles are added. To address this, the ASC throughput is also increased. When the throughput of QCs and ASC is balanced, it is discovered that, with a sufficient number of vehicles, AGVs become more effective. This advantage is likely attributed to their smaller size and increased agility, allowing them to better manage congestion.

In (Kumawat and Roy 2020), AGVs and lift-AGVs are compared, with the study identifying improved throughput performance in the simulated terminal when using lift-AGVs. The enhancement is so significant that the higher unit cost of lift-AGVs can be justified. However, according to a recent review of yard operation and management in container terminals (H. Yu et al. 2022), no single type of horizontal transportation machinery has emerged as the dominant choice in real automated ports thus far.

ACSs and their counterparts can also play a role in stacking operations. Historically, manual straddle carriers were a popular choice for managing yard storage. The pros and cons of utilizing straddle carriers in stacking operations, as opposed to yard cranes, have been examined in (Vis 2006) and (CHU and HUANG 2005). In essence, yard cranes offer higher container density and stacking height, along with easier automation. However, they may not be as fast as straddle carriers.

2.3 Layout of maritime container terminals

Maritime container terminals generally comprise a land interface, a water interface, and storage in between. Although terminal size, layout, and equipment can vary significantly, these terminals are typically divided into three distinct areas: the *quayside* (water interface), the *landside* (land interface), and the *yard* (storage), as illustrated in Figure 2.6. This thesis concentrates on the horizontal routing of vehicles between the quayside and container stacks.

Container terminals can be further categorized based on the orientation of container stacks in the yard, as depicted in Figure 2.5. In the parallel layout, containers are arranged parallel to the waterline, while in the perpendicular layout, containers are positioned perpendicular to the waterline. In both cases, container stacks are typically placed as close as possible to the quay side cranes to minimize horizontal transportation distances. Currently, most newly constructed automated container terminals prefer the perpendicular layout (H. Yu et al. 2022).

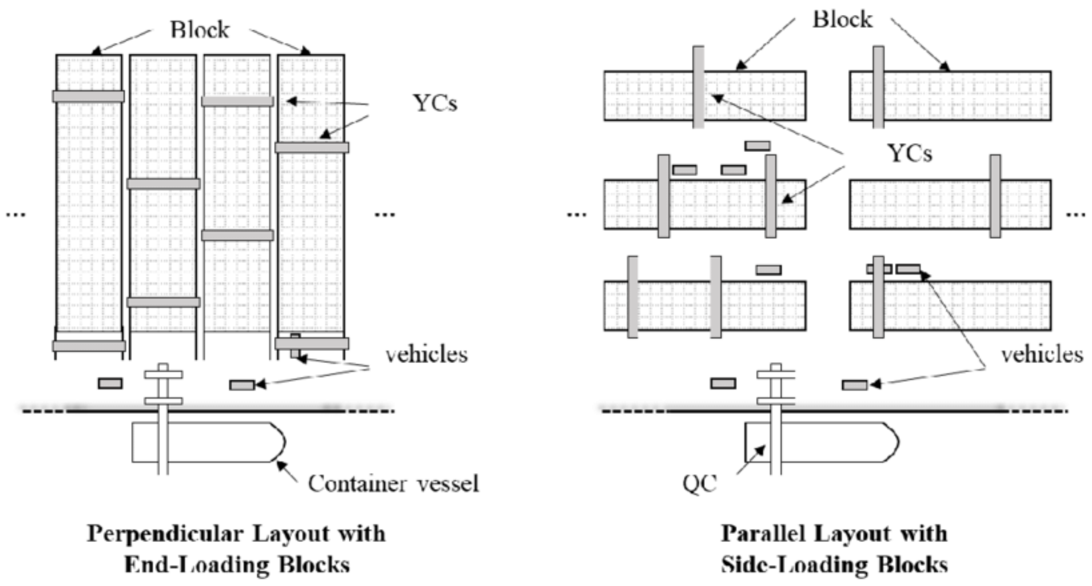

Block
YCs
...
...
vehicles
Container vessel
Perpendicular Layout with End-Loading Blocks
Block
YCs
...
...
vehicles
QC
Parallel Layout with Side-Loading Blocks

Figure 2.5. *Paraller vs perpendicular terminal layout. (Zhou, Lee, and Li 2020)*

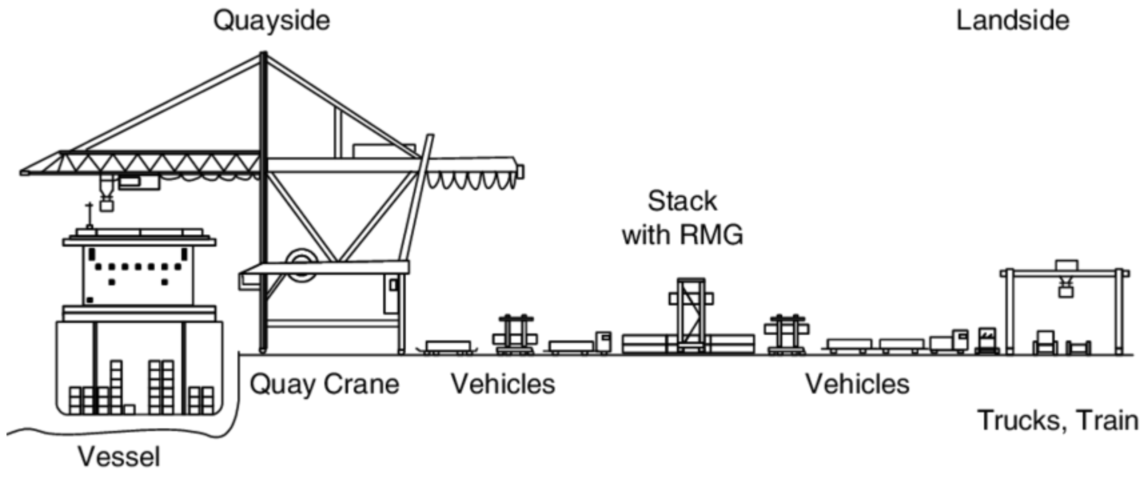


Figure 2.6. *A schematic side-view of a perpendicular layout. (Steenken, Voss, and Stahlbock 2004)*

3. MULTI-AGENT PATHFINDING (MAPF)

In the domain of multi-agent pathfinding (MAPF), the primary objective is to identify collision-free routes for numerous agents traversing a grid or graph. This area of research has garnered attention due to its real-world applications, such as guiding autonomous fleets, enhancing video games, and solving puzzles. Early investigations into this topic can be traced back to pebble motion problems, with the 15 puzzle being a prominent example, accessible online at (*15-Puzzle* 2022). In the 15 puzzle, the goal is to arrange 15 numbered tiles within a 4x4 frame by sliding them, ensuring that no two tiles occupy the same space. The MAPF problem poses a similar challenge, but typically with more available space, allowing multiple agents to move simultaneously (Sharon, Roni Stern, Felner, et al. 2015).

The 15 puzzle provides valuable insight into MAPF as it requires predicting future configurations resulting from specific tile movements. Both the 15 puzzle and MAPF problems can be classified as combinatorial optimization problems (H. Ma 2022). Identifying the shortest solution (measured by the fewest moves) for the 15 puzzle is an NP-hard problem (Goldreich 2011). Similarly, path planning in MAPF is also NP-hard on graphs (J. Yu and LaValle 2013), including grid-like graphs (Banfi, Basilico, and Amigoni 2017).

3.1 Graphs

To effectively formulate MAPF problems, it is crucial to first define graphs. Graphs provide an intuitive means of representing networks consisting of nodes and connections. A simple graph captures a network as a pair $G = (V, E)$, where:

- V represents a finite set, termed *vertices*, and
- $E \subseteq \{\{v_1, v_2\} | v_1, v_2 \in V \text{and } v_1 \neq v_2\}$, denoted as *edges*.

Graphs are typically depicted as 2D objects on a plane, with vertices portrayed by dots or circles. Two vertices are considered *neighbors* if they are connected by an edge. Edges are deemed adjacent if they share a common vertex. Vertices are *connected* when a *path* exists between them, where a path is a finite set of distinct adjacent edges. Simple graphs only permit a single multi-directional edge between vertices (Jungnickel 2008). A directed graph, or *digraph*, assigns direction to edges, with edges defined by ordered

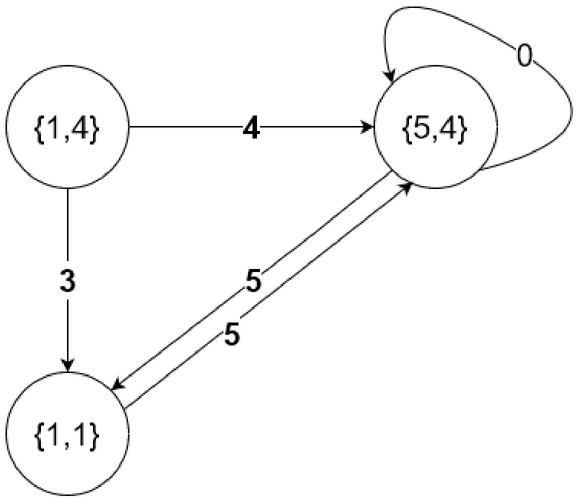


Figure 3.1. *A directed pseudograph with weighted edges.*

vertex pairs. The first vertex designates the start, while the second marks the goal. The possible combinations of unique vertex pairs (directed edges) can be determined using the *cartesian product* $V \times V$ of a vertex set.

By assigning weights $w_{i,j}$ to each edge $e_{i,j}$ in the simple digraph, the graph transforms into a weighted digraph. A weighted digraph is defined as $G = (V, E)$, where:

- V constitutes a finite set, and
- $E \subseteq \{\{v_1, v_2, w_{1,2}\} | (v_1, v_2) \in V \times V,\ w_{1,2} \in \mathbb{R}$ and $v_1 \neq v_2\}$.

In this thesis, a *pseudograph* is employed for modeling purposes. A pseudograph is a graph containing edges with the same starting and ending vertex. These are referred to as *loops* and can be utilized, for example, to model *waiting*. Figure 3.1 illustrates an example of a weighted directed pseudograph. Note that the weights in the figure are distance-based, demonstrating this modeling possibility. However, for a graph abstraction, weights can be chosen arbitrarily.

3.2 MAPF on graphs

A MAPF *instance* can be defined as an ordered pair (G, A), where G represents the underlying graph structure of the environment, and A is a set of agents. Each agent $a_i \in A$ has an assigned starting vertex s_i and a goal vertex g_i. Figure 3.2 illustrates an example of a classical MAPF problem instance on a graph. Utilizing grid maps (with or without holes) or graphs with constant node distances is advantageous, as it enables the following assumptions:

1. Time is discretized into time steps
2. Agents perform one action per time step

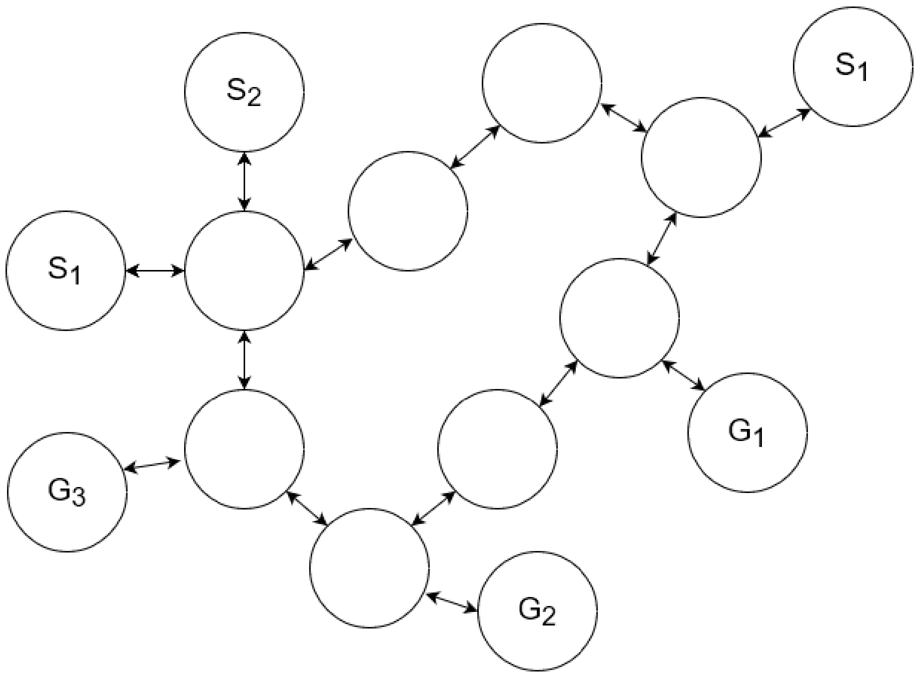


Figure 3.2. *An example of a MAPF instance on a graph.*

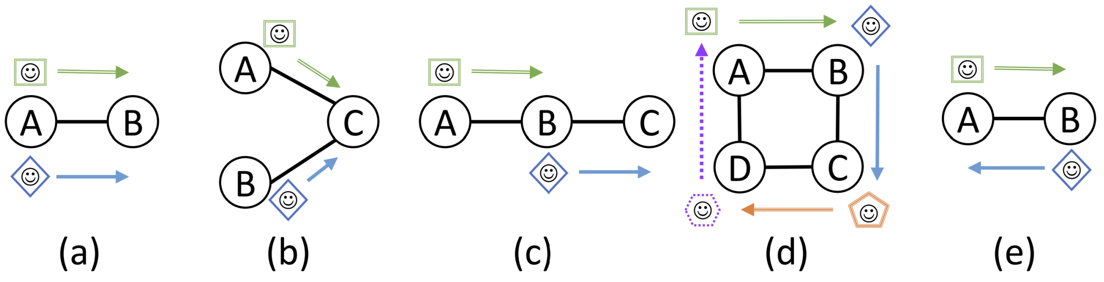


Figure 3.3. *Types of conflict considered in MAPF: an edge conflict (a), a vertex conflict (b), a following conflict (c), a cycle conflict (d) and a swapping conflict (e).(R. Stern et al. 2019)*

In this context, an agent's action refers to either remaining at a vertex or moving to a neighboring vertex. These assumptions necessitate that agents traverse between vertices in a fixed amount of time, potentially stopping at a vertex for the same duration. However, when the distances between vertices vary, the assumptions become more challenging to fulfill, as diverse accelerations are required to meet the simplification demands. In this thesis, the simplifying assumptions are employed in conjunction with a graph where node distances are constant.

Before tackling a Multi-Agent Pathfinding (MAPF) problem, it is crucial to take into account potential conflict situations. Common types of *conflicts* in MAPF problems are depicted in Figure 3.3. As MAPF problems typically involve two-dimensional scenarios, it is undesirable for agents to share the same edge during transitions, as this would place them on top of one another.

Conflicts at vertices represent the most intuitive type of conflict, akin to vehicle collisions at road intersections. Similarly, swapping conflicts can be thought of as head-on collisions. The following conflict, though not always problematic in real systems, can be compared to *tailgating*. Certain simulators disallow following, as their software prevents a vertex from simultaneously "letting out" and "taking in" an agent. Cycle conflicts resemble following conflicts, where actions must be executed synchronously, or no action can be taken at all.

Optimal solutions can be found for most MAPF instances, based on a specified metric. For instance, the solution to an MAPF problem involving n agents might consist of a set of action sequences $\pi = \pi_1, ..., \pi_n$, where each $\pi_i \in \pi$ represents a finite sequence of actions (in this thesis, actions are edges on a directed pseudograph, with waiting represented as a loop). The length $|\pi_i|$ of these sequences indicates the number of time steps needed for an agent to reach its destination. This information is utilized by the two most prevalent cost functions:

- **Makespan**

  Q: How many time steps are required *before* all agents reach their destinations?

  A: The *longest* sequence of actions, i.e., $\max_{1 \leq i \leq n} |\pi_i|$

- **Sum of Costs**

  Q: What is the *total* number of actions performed by all agents?

  A: The *sum* of action sequence lengths, i.e., $\sum_{1 \leq i \leq n} |\pi_i|$

This thesis focuses on the *sum of costs* metric, as the goal in horizontal transportation is to minimize the total distance and waiting time for the entire machine fleet.

3.3 Common strategies and methods

There are two overarching strategies to address a Multi-Agent Pathfinding (MAPF) problem: centralized and distributed approaches. In a centralized approach, a single CPU or multiple CPUs with full knowledge sharing work together to solve the MAPF problem. Conversely, the distributed approach allocates computing power to each agent, employing various communication methods between agents. Regardless of the chosen strategy, solution attempts can be broadly categorized as follows:

- **Reduction-based** - The MAPF problem is transformed into a well-known problem, such as integer linear programming (ILP), boolean satisfiability (SAT), or answer set programming (ASP). This approach is typically used to find optimal solutions but does not scale well for larger instances.
- **Search-based** - The MAPF problem is treated as a series of conflict resolution

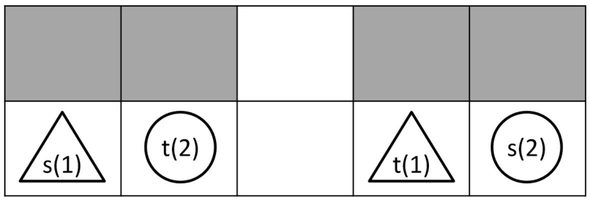


Figure 3.4. *A conflict that cannot be solved using prioritized planning. (Roni Stern 2019)*

instances, with each agent's initial individual solution designed to avoid collisions optimally or suboptimally. This can be achieved through prioritization or intelligent conflict analysis. While generally suboptimal, search-based algorithms can be optimal and tend to scale better for larger problems compared to reduction-based approaches. However, priority-based methods are inherently incomplete (see Figure 3.4).

- **Rule-based** - Like search-based methods, the MAPF problem is viewed as a series of conflict resolution instances. Instead of *searching* for alternative paths, individual conflicts are resolved using a predefined set of actions for the agents. This approach sacrifices optimality for faster computation times. Interestingly, rule-based methods often ensure completeness, meaning that a solution will always be found if it exists, unlike priority-based solutions. (Sharon, Roni Stern, Felner, et al. 2015)

In addressing the MAPF problem, there is an inherent trade-off between optimality, scalability, and completeness. Recognizing this, several powerful ideas have been proposed for optimal MAPF solving over the past decade. These ideas include:

- **Independence detection (ID):** (Standley 2010) presents a generalizable framework for detecting path independence. The approach begins with separate agent paths and groups colliding paths together for solving. In the worst-case scenario, all agents are merged into the same group, effectively solving the original problem. However, in most cases, independent groups can be detected and solved separately, saving time.
- **Extensions of A***: By considering an agent state space, A* can be run to find the *sum of costs* optimal solution to the MAPF instance. However, the branching factor of a graph representing such a search-space is exponential (b^n), where b is the number of actions for one agent (wait/left/right/up/down) and n is the number of agents. To address this issue, a few ideas have been proposed:
  - **Operator decomposition (OD):** (Standley 2010) introduces a novel method for decomposing the problem to reduce branching by creating a deeper search tree. The vertices are divided into *intermediate* and *full vertices*. At each

layer of the search tree, only one agent's possible movements are considered. If agents are at different time-steps, the vertex is considered intermediate; otherwise, it is considered full. The search is completed when a full vertex, representing the goal, is found.

  - **Enhanced partial expansion A* (EPEA):** (M. Goldenberg et al. 2014) present a novel method for reducing *surplus* nodes, which are nodes further away from the start than the goal node. Most A* implementations visit but never expand surplus nodes. EPEA uses a heuristic that prevents surplus nodes from being visited, reducing the branching factor. EPEA is considered a state-of-the-art A*-based MAPF solver (Kaduri, Boyarski, and Roni Stern 2020).

- **Two-level solvers:** Sharon et al. (Sharon, Roni Stern, Meir Goldenberg, et al. 2013) introduce the idea of using two separate solvers, involving a *high-level* and a *low-level* solver. There are two ways to implement this approach. The first option is for the high-level solver to find the lengths for individual solution paths, and the low-level solver to determine if such lengths can generate a feasible solution. The second option is for the high-level planner to use the *conflict* space and indicate to the low-level planner which conflicts to consider. In both cases, the process is repeated until a feasible solution is found, which, depending on the solver logic, will also be the optimal solution to the MAPF instance. Two-level solvers include:

  - **The Increasing Cost Tree Search (ICTS):** (Sharon, Roni Stern, Meir Goldenberg, et al. 2013) introduce a two-level method called the increasing cost tree (ICT) search. The high-level solver uses a vector tree, searched in a breadth-first manner, to provide solution lengths to the low-level solver. The root of the tree is a vector $(c_1, c_2, ..., c_k)$, consisting of the lowest individual path costs to the destination. Each child node is a vector with a cost increase of one to any of the path lengths. Creating and searching the ICT is an exponential process, but ultimately an optimal solution will be found. Effective pruning of the ICT can accelerate the process.

  - **Conflict-based search (CBS):** (Sharon, Roni Stern, Felner, et al. 2015) introduce a conflict space utilizing a two-level solver. Instead of an ICT, a constraint tree (CT) is used by the high-level solver. The nodes in the CT are a pair $\langle n.conflict, n.plan \rangle$, where the root starts with no conflicts. The low-level solver is tasked with generating a plan, given conflicts of agents at given vertices and times $\langle a, v, t \rangle$. The generated plan is used to create more nodes on the CT tree. The next node (conflict) to explore is chosen based on the plan cost until a feasible plan is found. Due to the way the CT is constructed, the first feasible plan will be the optimal solution to the MAPF instance.

- **Constraint programming (CP):** CP is a common methodology for combinatorial problems, such as the MAPF problem. The theoretical basis for CP lies in *logical inference* rather than *algebra*, as in general mathematical optimization. CP is restricted to discrete problems and can be further divided into two distinct categories:

- **Constraint satisfaction problem (CSP):** The CSP consists of a set of constraints and variables. To find an optimal solution, the cost function value is predetermined. In the MAPF case, for example, an initial lower bound L for the makespan cost can be found by examining the individual shortest paths for agents. If no solution is found, L is incremented by one and the process is repeated. CSP can be solved using various approaches such as Boolean satisfiability (SAT) and answer set programming (ASP).
    - **Boolean satisfiability (SAT)** - a particular case of CSP, commonly applied to MAPF problems. For instance, in (Surynek 2017), five distinct SAT formulations of MAPF are examined using the makespan objective. The sum of cost objective, which appears to be more challenging to model in SAT, is addressed, for example, by (Barták and Švancara 2021). The general concept in SAT involves formulating a *boolean formula*, where each variable is binary (true/false), and the relationships between variables are composed of only three basic logical operators: *not, or*, and *and*. By providing such a formula for the MAPF problem, highly efficient and continually improving solvers can be employed. Annual SAT algorithm competition results and source codes are available at (SAT Competition 2022).
    - **Answer set programming (ASP)** - can be employed to address CSP problems. ASP problems are frequently converted into SAT problems because they are simpler to solve, and the solution can be transformed back. The key distinction between ASP and SAT lies in their *logic*. SAT utilizes propositional logic, which means something is explicitly *proposed*, and all involved variables are either true or false (the boolean formula). In contrast, ASP employs predicate logic, featuring sets of *predicates* with non-binary variables that yield a truth value based on a specified set of rules. In (Erdem et al. 2013), an ASP approach is applied to address the MAPF problem.
- **Constraint optimization problem (COP):** COP is a generalization of CSP that includes an objective function. An optimal solution to a COP problem minimizes or maximizes the objective function $f(d)$. Several algorithms can be used to achieve this result, such as primal simplex, dual simplex, interior-point methods, and the branch-and-bound method.
    - **Primal Simplex** - A simplex is a particular type of convex polyhedron in

geometry that is a generalized triangle-like structure. In linear programming, the feasible solutions to a problem can be represented as points inside a polyhedron, where the vertices of the polyhedron correspond to the basic feasible solutions. The optimal solution to the problem is also located at one of these vertices. For convex LP problems, the Primal Simplex method starts at a vertex and moves along the edges of the polyhedron to neighboring vertices with lower (or higher) costs until the global minimum (or maximum) is found. This algorithm is used to efficiently solve linear programming problems with a convex feasible region. (Kalai 1997).

- **Dual Simplex** - The Dual Simplex method is another algorithm used to solve linear programming problems, and it takes a different approach than the Primal Simplex method. Rather than moving through feasible solutions to find the optimal feasible solution, the Dual Simplex method moves through non-feasible optimal solutions until it finds a feasible optimal solution. Despite the differences in approach, both the Dual and Primal Simplex methods will converge to the same solution if an optimal feasible solution exists. This is in accordance with the strong duality theorem for linear programs. (PAN 2013).
- **Interior-point methods** - The Interior-point methods are another set of algorithms used to solve linear programming problems. Unlike the Simplex methods which move from one vertex to another to reach the optimal solution, interior-point methods aim to reach the optimal solution vertex from any direction. There are two types of interior-point methods: the barrier method and the primal-dual method. The barrier method incorporates inequalities into the objective function as a logarithmic barrier function, and it follows a central path within the polyhedron towards the optimal feasible solution using Newton’s method. The primal-dual method uses Lagrangian multipliers to add dual problem constraints to the objective function, and it also follows the central path towards the optimal solution using Newton’s method. However, the primal-dual method requires fewer iterations and has improved accuracy compared to the barrier method. Both methods are suitable for solving convex linear programming problems. (VANDERBEI 2021).
- **Branch-and-bound method** - The branch-and-bound method is an algorithm that can be used solve combinatorial non-convex integer linear programming (ILP) problems, such as the ILP-reduced MAPF problem. It works by generating a solution tree where only the solution nodes leading to the optimal solution are branched. The branching is guided by a

bounding function that calculates the cost of each generated child node and compares it to the current solution bounds. To find the initial upper and lower bounds, the problem is first relaxed from a non-convex ILP problem into a set of convex LP problems, which can be solved using previously discussed methods such as the Simplex method. Once the bounds are found, the algorithm branches and searches for the optimal solution until it is no longer possible to branch. This method has been applied to various optimization problems such as automated container terminal routing and dispatching problems. (Wang and Zeng 2022).

The following three subsections will establish the foundation for creating two MAPF optimization models in chapter 4.

3.3.1 Binary Linear Programming (BLP)

A fundamental problem in multi-agent pathfinding (MAPF) can be represented as a binary linear programming (BLP) problem, which seeks to find a feasible solution that minimizes or maximizes a given objective function, subject to constraints. Specifically, a BLP problem can be formulated as a constrained optimization problem (COP), where the decision variable x is a binary vector with n dimensions, and each element is either 0 or 1. The general form of a BLP problem can be written as:

$$\begin{aligned} \min_{x\in\{0,1\}^n} \; & f_0(x) \\ \text{s.t.} \; & f_i(x) \leq 0, \;\; i = 1, ..., m \\ & h_j(x) = 0, \;\; j = 1, ..., p \end{aligned} \tag{3.1}$$

Here, f_0, f_i, and h_j are affine functions, and the constraints are defined by $f_i(x) \leq 0$ and $h_j(x) = 0$. Note that the solution space of a BLP problem is not convex, which means that standard convex optimization techniques cannot be used to solve it. Instead, combinatorial algorithms such as branch-and-bound are needed to find the optimal solution. In the upcoming sections, we will discuss how BLP problems can be used to model and solve MAPF problems, and explore various optimization techniques that can be employed to improve their efficiency and efficacy.

3.3.2 The Lagrange Dual Problem

The Lagrange dual problem is a powerful tool that can be used to solve non-convex combinatorial problems, including the binary linear programming (BLP) problem discussed earlier. The Lagrange dual problem involves transforming the constraints of an optimization problem into the objective function, resulting in an augmented objective function called the *Lagrangian*. For the BLP problem 3.1 , the Lagrangian can be written as:

$$\mathcal{L}(x, \lambda, \upsilon) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} \upsilon_j h_j(x), \tag{3.2}$$

where λ_i and υ_j are Lagrangian multipliers. Generally, λ and υ are real numbers, but for the inequality constraints f_i, the condition $\lambda \geq 0$ is required, as only values on the wrong side of the inequality should be penalized. In order to solve the original BLP problem, the Lagrangian is minimized with respect to the decision variable x, while maximizing with respect to the Lagrangian multipliers λ and υ.

The Lagrangian dual *function* is defined as the minimum of the Lagrangian over all possible values of x. The Lagrangian dual function can be written as:

$$g(\lambda, \upsilon) = \min_{x} \mathcal{L}(x, \lambda, \upsilon), \tag{3.3}$$

The Lagrange dual *problem* seeks to find the tightest possible lower bound for the Lagrangian dual function. In other words, the Lagrange dual problem seeks to maximize the Lagrangian dual function subject to the constraint $\lambda \geq 0$, which can be written as:

$$\begin{aligned} &\max_{\lambda, \upsilon} g(\lambda, \upsilon) \\ &\text{s.t. } \lambda \geq 0. \end{aligned} \tag{3.4}$$

Denoting the optimal solution to the Lagrange dual problem by $d^* = \max\limits_{\lambda,\upsilon} \min\limits_{x} \mathcal{L}(x, \lambda, \upsilon)$, and the optimal solution to the BLP problem 3.1 by p^*, a property called weak duality or max-min inequality is defined by:

$$d^* \leq p^*. \tag{3.5}$$

The possibility of a duality gap between d^* and p^* indicates that the Lagrangian dual problem may not always be able to perfectly capture the optimal objective function value of the original BLP problem. In other words, the optimal solution to the Lagrange dual problem d^* may be strictly less than the optimal solution to the original BLP problem p^*.

The Lagrange dual problem is a valuable tool for solving complex combinatorial problems (such as MAPF) and can be used to provide lower bounds on the optimal objective function value of the original problem. In particular, the Lagrange dual problem can be used to obtain tight lower bounds for the BLP problem, which can then be used in a branch-and-bound algorithm to search for the optimal solution.

Furthermore, the Lagrange dual problem can be used to derive the Karush-Kuhn-Tucker (KKT) conditions, which are necessary conditions for optimality for constrained optimization problems. The KKT conditions are a set of equations and inequalities that must be satisfied by any solution to the optimization problem, including the optimal solution. Therefore, verifying the KKT conditions for a given solution can help confirm its optimality.

In summary, the Lagrange dual problem is a useful tool for solving non-convex combinatorial problems, including the BLP problem 3.1. It provides lower bounds on the optimal objective function value of the original problem and can be used to derive necessary conditions for optimality. (Boyd and Vandenberghe 2021)

3.3.3 The quadratic penalty function

Penalty function methods, such as the quadratic penalty function method, can also be useful for solving combinatorial non-convex optimization problems, including the binary linear programming (BLP) problem. The quadratic penalty function involves penalizing deviations from the constraints quadratically in the objective function. For the BLP problem 3.1, the quadratic penalty function can be written as:

$$Q(x,u) = f_0(x) + \frac{\alpha}{2}\sum_{i=1}^{m} h_i(x)^2 + \frac{\alpha}{2}\sum_{j=1}^{p}[f_j(x)^+]^2, \tag{3.6}$$

where $f_j^+ = \max\{0, f_j\}$ and α is the penalty parameter. The quadratic penalty function method involves solving the original problem by fixing increasing values for α and computing $\min_x Q(x,u)$ iteratively until optimality or sufficient optimality is reached.

One advantage of the quadratic penalty function method is that it can be combined with the Lagrange dual method to obtain even tighter lower bounds on the optimal objective function value. This is achieved by using the augmented Lagrangian, which involves adding an additional penalty term to the quadratic penalty function. (Nocedal and Wright 2006)

3.3.4 The augmented Lagrangian method

The augmented Lagrangian method, also known as the method of multipliers, is a popular technique used to solve constrained optimization problems by transforming difficult constraints into the objective function. This method is described in detail in (Nocedal and Wright 2006) and (Birgin and M. 2014), and it has been used in various applications such as multi-agent path finding, as shown in (Nishi, Ando, and Konishi 2005).

One way to construct the augmented Lagrangian is by combining the Lagrangian and the quadratic penalty function into the objective function. This approach is useful because it allows for faster convergence for the Lagrangian and it's dual problem and mitigates some numerical issues associated with the quadratic penalty function method. For the

BLP problem in 3.1, the augmented Lagrangian is defined as follows:

$$\mathcal{L}(x,\lambda,\upsilon)_{aug} = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} \upsilon_j h_j(x) + \frac{\alpha}{2}\sum_{i=1}^{m} \lambda_i h_i(x)^2 + \frac{\alpha}{2}\sum_{j=1}^{p} \upsilon_j [f_j^+]^2 \tag{3.7}$$

where λ and υ are the Lagrange multipliers, and α is the penalty parameter. Now the augmented Lagrangian *problem* can be formulated as follows:

$$\begin{aligned} &\min_x \max_{\lambda,\upsilon} \mathcal{L}_{aug}(x,\lambda,\upsilon) \\ &\text{s.t. } \ \lambda \geq 0, \end{aligned} \tag{3.8}$$

The solution to this problem will be the solution to the original problem, however, it is typically difficult to solve. Therefore, similar to the Lagrangian problem, the augmented Lagrangian problem is also approached via the it's dual:

$$\begin{aligned} &\max_{\lambda,\upsilon} \min_x \mathcal{L}_{aug}(x,\lambda,\upsilon) \\ &\text{s.t. } \ \lambda \geq 0. \end{aligned} \tag{3.9}$$

The basic augmented Lagrangian method (method of multipliers) is described in (Bertsekas 2015). This method involves iteratively solving the dual problem to converge to the optimal dual solution. The algorithm works as follows:

- Find $x_{k+1} \in \arg\min_x \mathcal{L}_{aug}(x,\lambda_k,\upsilon_k)$
- Update the Lagrange multipliers:
  - Set $\lambda_{k+1} = \lambda_k + c_k h(x)$, where c_k is a step size and $h(x)$ is the vector of constraint violations.
  - Set $\upsilon_{k+1} = \upsilon_k + c_k f(x)^+$, where $f(x)^+$ is the vector of positive parts of the constraint functions.
- Repeat until convergence.

It is worth noting that the choice of the step size c_k can greatly affect the convergence rate of the algorithm. Choosing a step size that is too small can lead to slow convergence, while choosing a step size that is too large can lead to instability and divergence. There are several methods for choosing the step size, including heuristics, line search methods, and trust-region methods.

4. SOLVING THE MAPF PROBLEM

In this chapter, the Multi-Agent Path Finding (MAPF) problem is formulated as an optimization problem. First, a framework is introduced for modeling the environment and actions on a pseudograph. This framework is then used to formulate a constrained binary optimization problem. While this standard form optimization problem can be solved optimally, the combinatorial nature of the problem results in an exponential increase in computational time as the number of agents or map size increases. To address these issues, coupled constraints are relaxed using Lagrangian methods, which enable the decoupling of machines. One approach to solving this decoupled problem is presented, wherein a graph pathfinding algorithm is executed separately for each agent, and a solution to the MAPF instance is obtained iteratively through communication between agents.

4.1 Formulating a BLP model

In a basic Multi-Agent Path Finding (MAPF) problem, each agent is capable of taking an action at every time step. These actions consist of either moving to a neighboring node or staying at the current node. To model these actions, a pseudograph is used where each edge is assigned an integer identifier (not weight), as illustrated in Figure 4.1. The list of actions an agent must take to reach its goal is expressed as a sequence of edge identifiers. For instance, in Figure 4.1, the optimal sequence of actions for the agent starting at edge "1" and ending at edge "4" is $\pi = \{"2", "3"\}$, as no waiting is required. It is important to note that the sequence of edge identifiers must remain unshuffled, and repeating elements may occur, as waiting may add the same edge to the sequence multiple times.

The decision of whether an agent uses an edge at a given time is a binary (1 or 0) one. When combining the decisions of all agents, a binary vector is formed that represents the paths chosen by the entire fleet. To be more specific, for a given agent a,

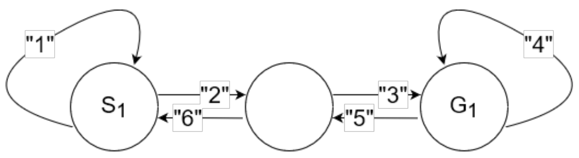


Figure 4.1. *The connection between edge choice and action.*

time k, and edge e, the edge is either used by the agent at that time ($x_{a,e,k} = 1$) or not used ($x_{a,e,k} = 0$). Hence, the decision variable $x_{a,e,k}$ is binary, taking values in $0, 1$. The solution to the problem (MAPF instance) is represented by the binary vector $x_{sol} = x_{a,e,k} \in \{0, 1\}, a \in A, k \in K, e \in E$.

A BLP model (introduced in section 3.3.1) can now be formulated using the general constraints for the MAPF problem (figure 3.3). The BLP problem can be written as

$$\begin{array}{llcll}
\min\limits_{x_{a,e,k}} & \sum_{a\in A}\sum_{e\in E}\sum_{k\in K} c_{a,e,k}x_{a,e,k} & & & \\
\text{s.t.} & x_{a,e_s(a),0} - 1 & = & 0 & \forall a \in A \quad (1)\\
& \sum\limits_{e\in E_{in}(n)} x_{a,e,k} - \sum\limits_{e\in E_{out}(n)} x_{a,e,k+1} & = & 0 & \forall a \in A, \forall n \in N, \forall k \in K \quad (2)\\
& \sum\limits_{e\in E} x_{a,e,k} - 1 & = & 0 & \forall a \in A, \forall k \in K \quad (3)\\
& \sum\limits_{a\in A}\sum\limits_{e\in E_{in}(n)} x_{a,e,k} - 1 & \leq & 0 & \forall n \in N, \forall k \in K \quad (4)\\
& \sum\limits_{a\in A}\sum\limits_{e_2\in E_{swap}(e)} (x_{a,e,k} + x_{a,e_2,k}) - 1 & \leq & 0 & \forall e \in E, \forall k \in K \quad (5)\\
& x_{a,e,k} & \in & \{0,1\}. & \forall a \in A, \forall e \in E, \forall k \in K \quad (6).
\end{array} \tag{4.1}$$

Table 4.1 summarizes the interpretations of the symbols used in the BLP model. In order to minimize the total sum of actions taken by the fleet of agents, a cost function of the sum of costs is considered. To achieve this, $c_{a,e,k}$ is assigned to be zero at the goal of the agent and any positive constant elsewhere. This choice of cost function motivates agents to reach their goals as quickly as possible, as lingering along the way incurs additional cost. It is worth noting that in the MAPF setting, minimizing actions is equivalent to minimizing space-time distance, as waiting also incurs a cost.

Let's now examine the constraints of the BLP model 4.1. Assuming the decision variable to be a binary vector, there are five additional constraints:

1. (1) At the first time step $(k = 0)$, each agent must be at its starting position on the graph. This constraint ensures that the initial state of the MAPF problem is satisfied.
2. (2) If an agent travels to a node, it must either remain there or move to a neighboring node. This constraint ensures that agents cannot jump to their destination and must traverse the edges in a continuous path.
3. (3) An agent must always be on exactly one edge. This constraint ensures that agents do not disappear or duplicate during the course of the plan.
4. (4) Only one agent can occupy a node or an edge at a time. This constraint prevents collisions between agents at nodes.

Table 4.1. *Interpretation of symbols in the BLP model.*

Symbol	Interpretation
x	Decision variable, indicates the use of an edge by a given agent at a given time
c	Cost of using an edge
a	Agent ID (natural number)
a_2	Another agent's ID, i.e., $a \neq a_2$
k	Time step from start time (positive integer)
e	Edge ID (positive integer)
n	Node ID (positive integer)
A	Set of agent IDs (positive integers)
K	Set of times within the horizon, i.e., $0, 1, 2, \ldots, end\_time$
E	Set of edge IDs (positive integers)
N	Set of node IDs (positive integers)
$e_s(a)$	The starting edge of an agent (different for each agent)
$E_{in}(n)$	Set of edges pointing to node n
$E_{out}(n)$	Set of edges starting from node n
$N_{goal}(e)$	The node to which edge e is pointing to
$N_{start}(e)$	The node from which edge e starts
$E_{swap}(e)$	Returns the parallel edge that connects the same pair of nodes in the opposite direction

5. (5) A node cannot be left and entered at the same time. This constraint prevents agents from colliding on edges or following each other too closely.

These constraints ensure that the BLP model produces feasible solutions that satisfy the initial state and the constraints of the MAPF problem.

Constraints 1-3 are linearly independent for each machine, whereas constraints 4-5 are not. Constraints 4 and 5 prevent conflicts of different types as shown in Figure 3.3: constraint 4 prevents conflicts of types a) and b), while constraint 5 prevents conflicts of types c), d), and e). These constraints can also be written as Equations $(4b)$ and $(5b)$ respectively:

$$\begin{aligned} x_{a,e,k} + \sum_{a_2 \in A \setminus a} \sum_{e_2 \in E_{in}(N_{goal}(e))} x_{a_2,e_2,k} - 1 &\leq 0 \qquad \forall a \in A, \forall e \in E, \forall k \in K \quad (4b) \\ x_{a,e,k} + \sum_{a_2 \in A \setminus a} \sum_{e_2 \in E_{in}(N_{start}(e))} x_{a_2,e_2,k} - 1 &\leq 0 \qquad \forall a \in A, \forall e \in E, \forall k \in K \quad (5b) \end{aligned} \tag{4.2}$$

However, this formulation results in more equations and is not used for the optimal solver in the next section. This formulation is primarily for later decoupling studies.

4.2 Solving the MAPF problem optimally

In this section, a method for achieving an optimal solution to the BLP problem 4.1 and the associated MAPF instance is presented. The approach involves two main components. Firstly, a modeling tool is required to interpret and store the cost function and constraints. In this study, the open source Python optimization modeling objects (Pyomo) software is utilized for this purpose. Secondly, a solver is required to find a solution to the problem given the constraints and cost function. Both commercial and open source solvers are considered.

4.2.1 Pyomo

To algorithmically solve the compactly written BLP equations 4.1, modeling software is needed. Algebraic modeling languages (AMLs) are often used for this purpose as they have a syntax similar to mathematical notation, allowing for symbolic notations to be used. These notations are then automatically translated into the low-level algebraic form required by algorithms. Open-source AMLs such as GNU MathProg and Zimpl, as well as commercial AMLs like GAMS, MPL, AIMMS, and AMPL are available, each with their own language syntax, solver compatibility, and data management.

In this thesis, the open-source Python optimization modeling objects (Pyomo) library is used as an AML. Unlike most AMLs, Pyomo does not use a custom modeling language. Instead, modeling objects are embedded within Python, enabling seamless incorporation with other Python libraries, such as NumPy, SciPy, and Matplotlib, and virtually endless possibilities for result analysis. The model created using Pyomo is a Python object with attributes equating to the properties of the original problem, including information about the cost function and constraints. Pyomo can be used to model various optimization problems, such as linear programming problems (LP), binary linear programming (BLP), integer linear programming (ILP), mixed-integer linear programming (MILP), nonlinear programming problems (NLP), binary quadratic programming (BQP), integer quadratic programming (IQP), and mixed-integer quadratic programming (MIQP).

The Pyomo model can be stored as either an abstract model or a concrete model, which differ only in the time of component initialization. In the abstract model, the model components (equations) are not initialized on creation, but instead, a high-level formulation is stored, which can later be used to initialize the low-level algebraic equations on command. The concrete model initializes the lowest-level algebraic equations on creation and can be sent to the solver as is. (*Pyomo - optimization modeling in Python* 2013)

In this thesis, simple text files containing information about the graph the agents are operating on and their starting and destination edges are used to import data into the model. The information is then used to create a concrete model of the BLP problem. Pyomo considers each constraint in 4.1 and the objective (cost function) in modeling. Pyomo creates a model object that can be examined to check the equivalence of the actual constraints and the interpreted ones. In the final step by Pyomo, as seen in appendix A, a solver is chosen, and the model is solved. Different kinds of solvers are introduced in the next section.

4.2.2 LP solvers

The BLP problem 4.1 and the subsequent MAPF instance can be solved with well-established linear programming (LP) solvers. LP solvers are used for at least four different kinds of linear problems: BLP, ILP, MBLP, and MILP. In BLP, all variables are binary, in ILP, all variables are integer, in MBLP, all variables are binary or continuous, and in MILP, all variables are integer or continuous.

Although there is not much research on which solvers are best at solving large-scale BLP problems, there seems to be a clear divide in performance by commercial and open-source solvers. For example, in (Meindl and Templ 2013), a wide range of LP problems were tested on open-source and commercial solvers, and the results indicated that the best commercial solver was around five times faster than the best open-source solver. There also seem to be problems for which even the commercial solvers struggle to find solutions in adequate time, or perhaps even at all. The solvers that will be used to solve the BLP problem 4.1 are listed in Table 4.2.

Ultimately, the choice of LP solver will depend on the specific needs and constraints of the problem at hand. In the context of container terminals, where large-scale optimization problems are common, it may be beneficial to consider using a powerful commercial solver like CPLEX or Gurobi. However, open-source solvers like GLPK and `lp_solve` may also be suitable, particularly if cost is a concern or if the problem is not particularly complex.

4.3 Solving the MAPF problem suboptimally

In Section 4.2, a way to optimally solve the MAPF problem was introduced. However, the combinatorial nature of the problem causes computation time to increase exponentially as the graph size and agent count grow. To address this issue, a suboptimal solution that distributes computational load is proposed. The idea is to decouple the common constraints 4.2 and solve the optimization problem separately for each agent iteratively. To achieve this result, two steps are taken. First, the BLP problem 4.1 is reformulated

Table 4.2. *Common LP solvers.*

Solver	Devoloped by	Algorithms used	Open source
glpk	GNU project	Primal and dual Simplex, Interior-point, Branch-and-bound	Yes
lp_solve	Michel Berkelaar	Primal and dual Simplex, Branch-and-bound	Yes
Cplex	IBM	Unknown	No
Gurobi	Gurobi Optimization	Unknown	No

to be an augmented Lagrangian relaxed problem. Next, the dual of this reformulated problem is solved suboptimally using graph algorithms. This approach has been shown to be effective in many MAPF problems, including applications in semiconductor fabrication bays (Nishi, Ando, and Konishi 2005).

4.3.1 Formulating the augmented Lagrangian dual

The penalty functions (see 3.3.3) of inequalities 4.2, can be written as

$$\begin{aligned} g(a,e,k)^+ &= \max\{0, x_{a,e,k} + \sum_{a_2 \in A \setminus a} \sum_{e_2 \in E_{in}(N_{goal}(e))} x_{a_2,e_2,k} - 1\} \\ h(a,e,k)^+ &= \max\{0, x_{a,e,k} + \sum_{a_2 \in A \setminus a} \sum_{e_2 \in E_{in}(N_{start}(e))} x_{a_2,e_2,k} - 1\}. \end{aligned} \tag{4.3}$$

The penalty functions in equations 4.3 are designed to evaluate only constraint violations. When the left side of the inequality is negative (-1), indicating that the constraint is not violated, no penalty or reward is added to the cost function. This is ensured by using the *max* function to take the maximum between 0 and amount violated. The augmented Lagrangian can be written as

$$\mathcal{L}_{aug}(x, \lambda, \phi) = J_1 + J_2 + J_3 + J_4 + J_5, \tag{4.4}$$

where

$$\begin{aligned}
J_1 &= \sum_{a\in A}\sum_{e\in E}\sum_{k\in K} c_{a,e,k}x_{a,e,k} \\
J_2 &= \sum_{a\in A}\sum_{e\in E}\sum_{k\in K} \lambda(a,e,k)g(a,e,k)^+ \\
J_3 &= \sum_{a\in A}\sum_{e\in E}\sum_{k\in K} \phi(a,e,k)h(a,e,k)^+ \\
J_4 &= \frac{\alpha}{2}\sum_{a\in A}\sum_{e\in E}\sum_{k\in K}[g(a,e,k)^+]^2 \\
J_5 &= \frac{\alpha}{2}\sum_{a\in A}\sum_{e\in E}\sum_{k\in K}[h(a,e,k)^+]^2.
\end{aligned}$$

The augmented Lagrangian $\mathcal{L}_{aug}(x,\lambda,\phi)$, defined in equation 4.4, consists of the original cost function J_1 and Lagrangian costs J_2 and J_3, where $\lambda(a,e,k)$ and $\phi(a,k,e)$ are Lagrangian multipliers. The costs J_4 and J_5 are quadratic penalty terms, scaled by $\alpha > 0$. This new cost function can be used to solve the original minimization problem by finding the optimal solution of the augmented Lagrangian.

The intuition behind the augmented Lagrangian method is that the Lagrangian multipliers are chosen to maximize J_2 and J_3, causing the costs to be positive if there are violations in the constraints, and zero otherwise. The quadratic penalty terms J_4 and J_5 always pull the solution towards feasibility, resulting in positive costs when there are violations in the constraints, and zero otherwise. As a result, in the feasible solution space, $J_2 = J_3 = J_4 = J_5 = 0$, and only the original cost function remains. The augmented Lagrangian dual problem (see section 3.3.4) can be written as

$$\begin{array}{llrcll}
\max\limits_{\lambda_{a,e,k},\phi_{a,e,k}} & \min\limits_{x_{a,e,k}} \mathcal{L}_{aug}(x,\lambda,\phi) & & & & \\
\text{s.t.} & & x_{a,e_s(a),k} - 1 & = & 0 & \forall a\in A \quad (1)\\
& \sum\limits_{e\in E_{in}(n)} x_{a,e,k} - \sum\limits_{e\in E_{out}(n)} x_{a,e,k+1} & & = & 0 & \forall a\in A, \forall n\in N, \forall k\in K \quad (2)\\
& & \sum\limits_{e\in E} x_{a,e,k} - 1 & = & 0 & \forall a\in A, \forall k\in K \quad (3)\\
& & x_{a,e,k} & \in & \{0,1\} & \forall a\in A, \forall e\in E, \forall k\in K. \quad (6)
\end{array} \tag{4.5}$$

The augmented Lagrangian method provides an alternate approach for solving the problem, where the dual problem is solved instead. Rather than determining the least upper bound of the augmented Lagrangian function with respect to x, the greatest lower bound with respect to λ is determined instead. However, for non-convex problems like the BLP problem, there might be a *duality gap* as discussed in 3.3.2 If there is no gap, the solution to the dual is the solution to the original problem. This possibility of strong duality offers some justification for the following simplified optimization strategy:

- Attempt to solve the dual problem $\max_{\lambda_{a,e,k},\phi_{a,e,k}} \min_{x_{a,e,k}} \mathcal{L}_{aug}(x, \lambda, \phi)$ (with constraints 1-3).
- Use a found *feasible* solution as the solution to the original problem.

This strategy may result in a suboptimal solution, as even if the dual is optimally solved, the possibility of a duality gap prevents the certainty that the found solution is optimal.

4.3.2 Decoupling

In this section a divide and conquer algorithm design paradigm is applied, and the optimization problem is divided to individual agent pathfinding subproblems, where each agent's singular pathfinding problem is solved iteratively in parallel fashion to obtain a solution to the original MAPF problem. As established in section 3, the MAPF problem consist of agents finding paths without colliding with each other. In the corresponding BLP problem 4.1 there are agent decoupled (1-3) and coupled (4-5) constraints. The coupled constraints essentially bind the sub-problems of individual pathfinding into a one large optimization problem. This works well for small instances, but adding more agents becomes problematic very quickly, as even if it only increases the size of the optimization problem linearly, together with increasing the map size, the problem starts to grow exponentially.

As mentioned earlier, the concept of using the augmented Lagrangian and its dual in the MAPF setting was originated from (Nishi, Ando, and Konishi 2005). The directed pseudograph-based MAPF formulation used in this thesis allows for a more straightforward formulation of the decoupled problem since each cost function in 4.4 can be written directly as a function of the corresponding agent:

$$\mathcal{L}_{aug}(x, \lambda, \phi, a) = J_1(a) + J_2(a) + J_3(a) + J_4(a) + J_5(a), \tag{4.6}$$

where

$$
\begin{aligned}
J_1(a) &= \sum_{e\in E}\sum_{k\in K} c_{a,e,k}x_{a,e,k} \\
J_2(a) &= \sum_{e\in E}\sum_{k\in K} \lambda(a,e,k)g(a,e,k)^+ \\
J_3(a) &= \sum_{e\in E}\sum_{k\in K} \phi(a,e,k)h(a,e,k)^+ \\
J_4(a) &= \frac{\alpha}{2}\sum_{e\in E}\sum_{k\in K}[g(a,e,k)^+]^2 \\
J_5(a) &= \frac{\alpha}{2}\sum_{e\in E}\sum_{k\in K}[h(a,e,k)^+]^2.
\end{aligned}
$$

The decoupled, distributed problem can now be written as

$$
\begin{array}{lrcll}
\max\limits_{\lambda_{a,e,k},\phi_{a,e,k}} \; \min\limits_{x_{a,e,k}} \sum\limits_{a\in A} \mathcal{L}_{aug}(x,\lambda,\phi,a) & & & & \\
\text{s.t.} & x_{a,e_s(a),k} - 1 & = & 0 & (1) \\
& \sum\limits_{e\in E_{in}(n)} x_{a,e,k} - \sum\limits_{e\in E_{out}(n)} x_{a,e,k+1} & = & 0 \qquad \forall n\in N, \forall k\in K & (2) \\
& \sum\limits_{e\in E} x_{a,e,k} - 1 & = & 0 \qquad \forall k\in K & (3) \\
& x_{a,e,k} & \in & \{0,1\} \qquad \forall e\in E, \forall k\in K & (6)
\end{array}
\tag{4.7}
$$

The intuition underlying this process is that each agent's pathfinding problem becomes an individual BLP problem, linked solely through shared Lagrangian multipliers. One crucial question concerns the selection of these multipliers. Essentially, Lagrangian multipliers determine which agent will yield in a conflict situation. This represents a fundamental challenge in MAPF that must be addressed without resorting to priority-giving methods, as they are inherently incomplete. For instance, Sharon and Guni (2015) present an innovative yet straightforward method for optimally managing conflicts. In this thesis, however, suboptimality is accepted in exchange for increased speed and scalability. The following section introduces an iterative algorithm designed to solve problem (4.7) in a distributed manner.

4.3.3 A Distributed algorithm

This section presents a distributed algorithm derived from the decoupled augmented Lagrangian dual problem outlined in equation 4.7. The primary concept involves employing pathfinding algorithms on graphs, as the dual problem can be interpreted as a least weighted pathfinding problem on space-time graphs. A space-time graph refers to a di-

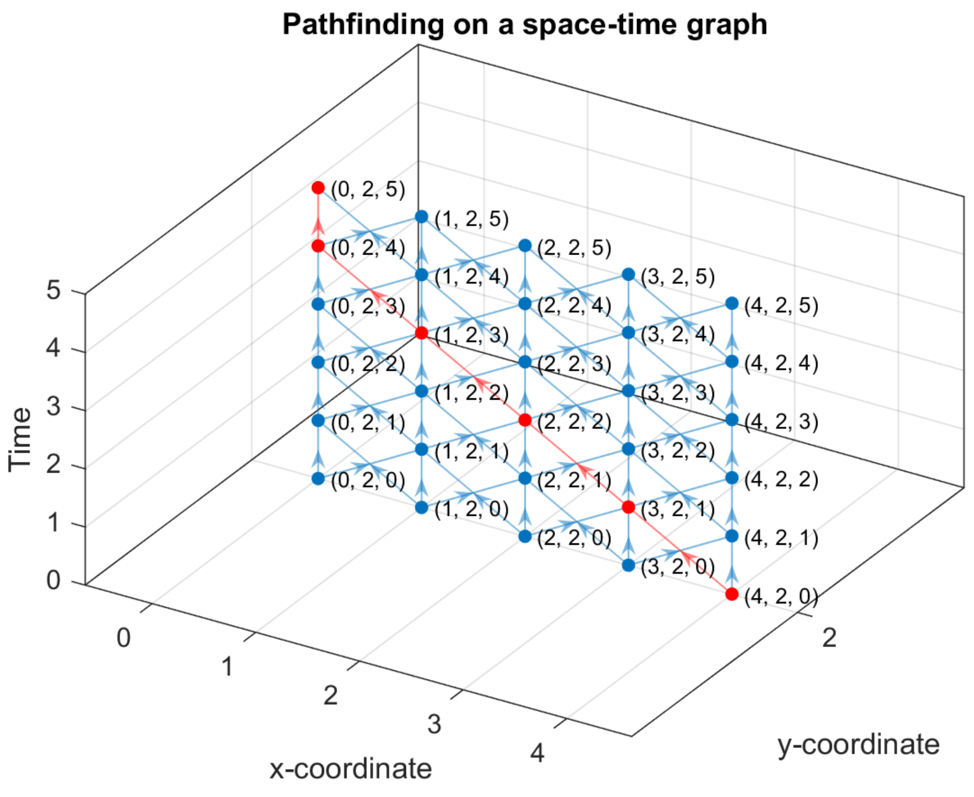


Figure 4.2. *An example of pathfinding on a space-time graph, where each node represents a location in space and time. The red path represents an agents path between space (x,y) locations (4,2) and (0,2).*

rected graph with nodes tied to specific locations in space-time. For instance, in a 2D horizontal transportation scenario, the space-time graph nodes possess three parameters: the x-coordinate, the y-coordinate, and the time coordinate. This concept is demonstrated in Figure 4.2. The space-time representation effectively allows for modeling waiting and transition time. Unlike on the pseudograph, it is now feasible to utilize well-established graph pathfinding algorithms, eliminating the need for LP solvers.

The minimization problem in the dual 4.7 and finding a least-weighted path on a space-time graph is equivalent, if the space-time graph's edge weights are determined by the following equation:

$$W_a(e,k) = w + \lambda_a(e,k) + \phi_a(e,k) + \frac{\alpha}{2} \cdot \sum_{b \in A \setminus a} (edges(x_b)^2 + swap\_edges(x_b)^2). \quad (4.8)$$

This equation takes into account Lagrangian multipliers $\lambda_a(e,k), \phi_a(e,k)$ which are used to penalize node and edge collisions , and a quadratic penalty function which penalizes the use of the same edges ($edges(x_b)$) or swapping edges (edges going between same nodes ($swap\_edges(x_b)$)) by a scaling factor of $\alpha/2$. In other words the violation of penalty functions $g(a,e,k)^+$ and $h(a,e,k)^+$ in 4.3 are penalized by edge weights on a space-time graph. Each agent has their own individual space-time graph with its own set of weights. The initial weight w in this thesis is always set to 1.

A simple and potentially effective approach to iteratively solve the $max\ min$ problem 4.7 is to utilize a variant of the *augmented Lagrangian method*, 3.3.4. The Lagrangian multipliers are updated according to algorithm 1. With a constant step size assumed, the multipliers increase with each iteration if constraints are violated.

Algorithm 1 Augmented Lagrangian algorithm (fixed step size)

```
1: $x_{next} \in \arg\min_{x \in \{0,1\}^n} \mathcal{L}_{aug}(x, \lambda, \phi)$
2: $\lambda_{next} = \lambda + \Delta\lambda * g(a,e,k)^+$
3: $\phi_{next} = \phi + \Delta\phi * h(a,e,k)^+$
```

When dealing with a non-convex problem and a fixed step size, convergence and optimality in the dual cannot be guaranteed. Moreover, it can be challenging to prevent oscillations, which occur when agents attempt to correct conflicts in a symmetric manner. This results in the same Lagrangian multipliers being continuously updated, which, in turn, leads to another conflict and perpetuates the cycle. To address this issue, (Nishi, Ando, and Konishi 2005) proposed a solution: allowing an agent to randomly maintain its path without correction in the next iteration.

Based on these ideas, distributed stochastic heuristic algorithm 2 is designed to find the shortest paths for multiple agents from their respective starting nodes to their goal nodes without colliding. The idea of the algorithm 2 is to establish the location and goal for each agent and then start to iteratively calculate the least weighted paths for each agent in parallel. The agents continue to iterate until no collisions is confirmed via communication.

On each iteration, the algorithm adds penalty weight to the weights of edges that were used by other agents, in order to discourage agents from using those edges. The size of the penalty depends on the number of agents using the edge, and is determined by the α parameter. This penalty encourages agents to explore different paths in the graph, rather than following the same paths that other agents have previously taken. By exploring different paths, agents are more likely to find collision-free and swap-free paths that are better suited to their individual needs, and are less likely to get stuck in a cycle of repeated conflicts.

The Lagrangian multipliers in the MAPF algorithm are used to discourage collisions between agents' paths. They are updated in each iteration based on the current node and edge collisions in the system. Specifically, when a node collision or edge collision (swap) is detected, the Lagrangian multipliers associated with the affected edges are increased by a fixed amount ($\Delta\lambda$ for collisions, $\Delta\phi$ for swaps). By continuously updating the Lagrangian multipliers based on the current state of the system, the algorithm can slowly increase the penalty for node and edge collisions, and encourage agents to avoid edges that have led to conflicts in the past.

The edge weights are reset on each iteration because they are being updated in response to the current set of agent paths. The updated edge weights are meant to penalize agents for using edges that have been used by other agents or that lead to collisions. By resetting the edge weights the algorithm ensures that the updated weights are based on the current set of agent paths, without any carry-over from previous iterations expect for the Lagrangian multipliers which slowly accumulate weight on frequent collision points. The approach guarantees that the edge weights are consistent with the current set of paths, which in turn ensures that the algorithm is correctly penalizing agents for using each other's paths.

Skipping is added to the algorithm 2, which refers to the process of not updating the paths of all agents in each iteration. Instead, a random subset of agents are selected to update their paths. This is controlled by the ξ parameter, which determines the probability of an agent being updated in a given iteration. For example, if $\xi = 0.6$, then there is an 60% chance that an agent's path will not be updated in a given iteration. Skipping is important because it allows the algorithm to converge more quickly to a feasible solution. By only updating a random subset of agents' paths in each iteration, the algorithm can avoid problematic cyclic patterns. This reduces the likelihood of collisions and increases the chances of finding a feasible solution to the MAPF problem.

The MAPF algorithm 2 is distributable because it can be decomposed into independent sub-problems that can be solved simultaneously by different processors or machines. In other words, the algorithm can be parallelized, where different agents can be assigned to different processors, and each processor can solve the sub-problem for its assigned agent independently. Specifically, each agent's path calculation can be done independently, once each agent has figured out what the others are planning via communication. Communication plays a vital role in coordinating the paths of multiple agents. Specifically, the agents need to communicate their paths to each other in order to detect conflicts and find a set of collision-free paths. There are two general types of communication that can be used in the algorithm: all-to-all communication and local communication.

All-to-all communication refers to the process of each agent sending its current path to all other agents in the system. This approach ensures that each agent has complete information about the paths of all other agents. However, all-to-all communication can be slow and can lead to a large amount of communication overhead, especially in large systems with many agents. This can make it difficult to scale the algorithm to handle large problems.

Local communication refers to the process of each agent sending its current path only to its immediate neighbors in the graph. This approach reduces the amount of communication overhead and can be more scalable than all-to-all communication. However, local communication can result in agents having incomplete information about the paths of other agents that are not their immediate neighbors. This can make it more difficult to detect conflicts and find collision-free paths.

Algorithm 2 A distributed pathfinding procedure for a single agent via communication to handle conflicts with other agents.

```
    $\xi, \Delta\lambda, \Delta\phi, \alpha, G \leftarrow Initialize()$
 1: $location, goal \leftarrow RecieveMessage(''plan'')$
 2: $solution \leftarrow ShortestSpaceTimePath(G, location, goal)$
 3: $BroadcastMessage(solution, ''paths'')$
 4: $WaitForOthers()$
 5: $solutions \leftarrow RecieveMessage(''paths'')$
 6: while $Conflicts(solutions)$ do
 7:     if $RandomRealBetween(0, 1) \leq \xi$ then
 8:         $e_n, k_n \leftarrow FindNodeCollisions(solution, solutions)$
 9:         $e_e, k_n \leftarrow FindEdgeCollisions(solution, solutions)$
10:         $\lambda(e_n, k_n) \leftarrow \lambda(e_n, k_n) + \Delta\lambda$
11:         $\phi(e_e, k_e) \leftarrow \phi(e_e, k_e) + \Delta\phi$
12:         $edges \leftarrow EdgesUsed(solutions)$
13:         $swap\_edges \leftarrow SwapEdgesUsed(solutions)$
14:         $G.Weight \leftarrow w + \lambda + \phi + \frac{\alpha}{2} \cdot \sum_{solutions}(edges^2 + swap\_edges^2)$
15:         $solution \leftarrow ShortestSpaceTimePath(G, location, goal)$
16:     $BroadcastMessage(solution, ''paths'')$
17:     $WaitForOthers()$
18:     $solutions \leftarrow RecieveMessage(''paths'')$
19:     $G.Weight \leftarrow w$
20: return $solution$
```

In summary, this section presented a approach for solving the multi-agent pathfinding problem using a distributable stochastic algorithm 2, based on the decoupled augmented Lagrangian dual formulation of the MAPF problem 4.7. The proposed algorithm enables using pathfinding algorithms on space-time graphs, where each node represents a location in space and time, and each agent has its own individual space-time graph with its own set of weights. The algorithm is designed to find the shortest paths for multiple agents from their respective starting nodes to their goal nodes without colliding. The approach guarantees that the edge weights are consistent with the current set of paths, which in turn ensures that the algorithm is correctly penalizing agents for using each other's paths. The algorithm is distributable and can be parallelized, allowing for scalable solutions to the multi-agent pathfinding problem. Lastly, it should be noted that practical implementation details, such as the specific communication protocols or hardware requirements needed to implement the algorithm in a real-world setting, are outside the scope of this thesis.

5. SIMULATIONS

This chapter focuses on the applicability of the distributed MAPF algorithm 2 to the horizontal transportation case in automated container terminals. The first step involves establishing a connection between MAPF and horizontal transportation through a graph representation of the terminal. Next, a basic MAPF simulator is implemented in Matlab to evaluate the effectiveness of the algorithm. The simulation is conducted on two maps: a bottleneck-free map 5.2 and a map with a bottleneck 5.7. The impact of parameter selection and the number of agents is studied through repeated simulations of randomized missions on both maps.

5.1 Terminal simulator

To apply MAPF to horizontal transportation, the container terminal must be modeled as a set of non-overlapping planning areas, with only one agent allowed to exist in each area at any given time. This can be accomplished by representing the terminal layout as a *grid* or a more general *graph*, as seen in related research such as (Hu et al. 2021) and (Wang and Zeng 2022). In this chapter, the perpendicular layout of the terminal is used as an example, as depicted in figure 2.5. The horizontal transportation area is further divided into three functional areas: the quay crane operational area, the buffer area, and the yard side area.

In the domain of horizontal transportation, the MAPF task involves moving containers between the quayside and yardside. However, the problem is not limited to routing, as each agent in the terminal is responsible for carrying out multiple container transporting missions. The problem can be divided into three interconnected components: scheduling, dispatching, and routing. Scheduling entails the ordering and timing of container handling, while dispatching assigns equipment to handle specific containers based on the schedule. Routing determines the paths for agents in line with the dispatcher's instructions. While this thesis focuses on the routing aspect, it is worth noting that any simulation of terminal operations requires some level of dispatcher implementation.

To construct a simulator for horizontal transportation, a graph similar to figure 5.1 is implemented in MATLAB. The goal is to simulate a vessel unloading mission, where containers are unloaded by quay cranes and transported by horizontal equipment from quay cranes

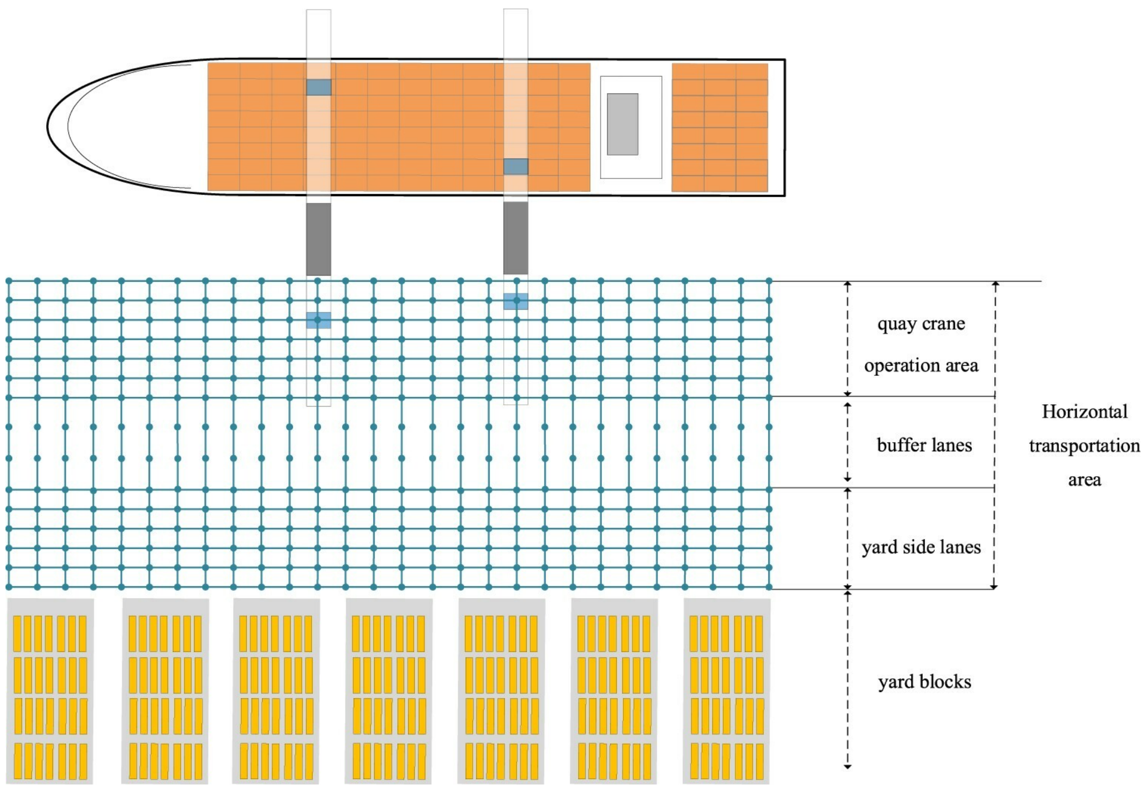


Figure 5.1. *Graph representation of horizontal transportation (Hu et al. 2021).*

to yard cranes. This situation requires a high-level coordinator and a continuously running MAPF algorithm. The simulator follows the following basic procedure:

- Agents are initialized at queue nodes
- Quay and yard crane productivity are assumed to be infinite
- The MAPF algorithm is run at every time step, with a long enough time horizon to plan a path all the way to the agent's destination
- If an agent reaches its destination, a new destination is assigned by the high-level coordinator

The high-level coordinator operates according to the following logic:

- Agents at queue nodes are assigned available quay crane nodes
- Agents at quay cranes are assigned random yard crane nodes
- Agents at yard cranes are assigned queue nodes with the shortest queue

When an agent visits a yard crane node, a container is delivered, and the mission is considered completed when a fixed number of containers have been delivered.

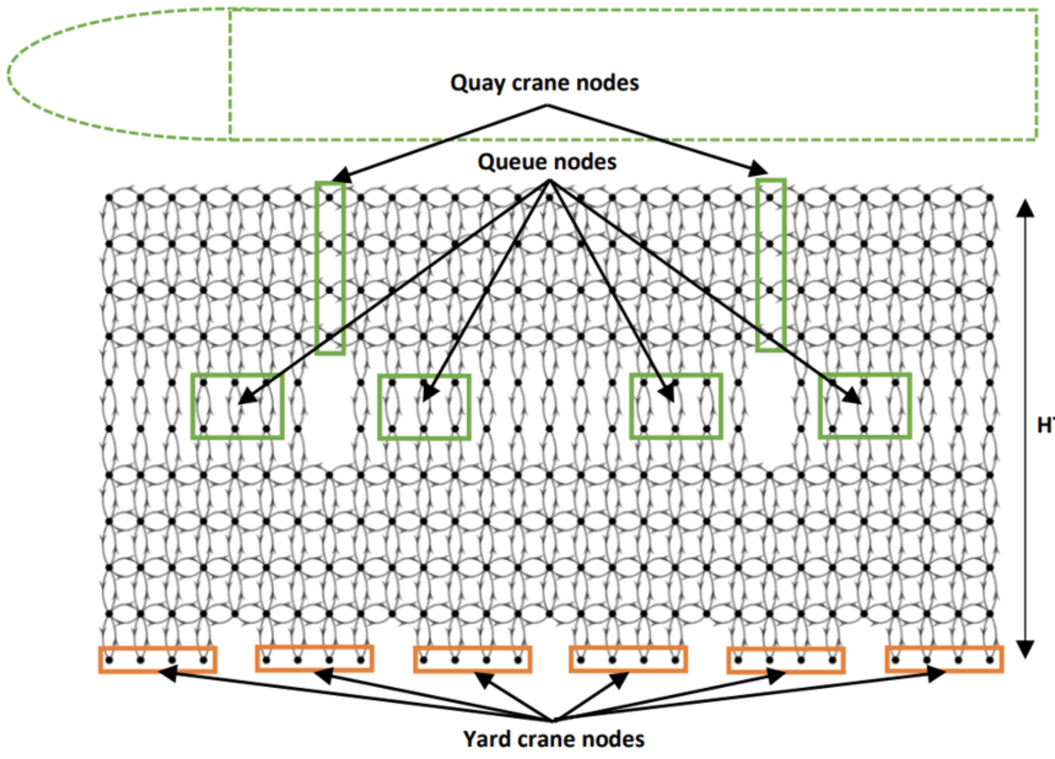


Figure 5.2. *A graph model of a bottleneck-free container terminal layout.*

5.2 Bottleneck-free case

The first case to be considered is a container terminal with a layout that allows for smooth and efficient agent movement without any bottlenecks. This layout is illustrated in figure 5.2. The layout features multiple adjacent lanes that allow agents to avoid collisions easily, without having to significantly increase the distance traveled. The number of agents considered on the map ranges from 1 to 24, resulting in a relatively low agent density. It is worth noting that some optimal solution methods, such as solving the BLP optimization problem 4.1, may not work well for these kinds of medium/large sparsely populated graphs, as noted in (Švancara and Barták 2019). A search-based method could be considered, but the focus of this analysis is on evaluating the performance of the developed distributable algorithm 2.

Two tests are conducted to analyze the impact of parameter selection and the number of agents on the overall performance of the terminal. The first test examines the effect of parameter choice (see algorithm 2) on the overall time steps and the number of iterations required to transfer 500 containers with a fixed number of agents (15). The results of the first test reveal a good set of parameters, which are used in the second test. The second test examines the impact of varying agent amounts on the performance with fixed parameter choices. Figures 5.3 and 5.4 depict the impact of different parameters on optimality and number of iterations in the first test, while figures 5.5 and 5.6 illustrate the effect of varying agent numbers in the second test. Further details about the tests are discussed in the following paragraphs.

Figure 5.3 shows the average completion time of ten 500-container missions in a simulated container terminal. The x-axis represents different parameter combinations (which can be found in table 5.2, while the y-axis shows the average mission completion time. The figure contains three lines, with the proposed method's completion time varying depending on the parameter combination. The optimal completion time line is a lower bound for the solution quality, determined using the BLP model, while the priority-based line is an upper bound for the solution quality. This serves as an important reference value, as any method performing over this threshold of convenience and reliability should not be considered. In Figure 5.3, only four parameter combinations achieve a completion time below the priority method, at the cost of higher iterations. The idea of priority planning is explained in section 3.3.

Figure 5.4 illustrates the simulated time complexity of the algorithm 2 when used to solve ten 500 container missions. The figure presents the average number of iterations and maximum iterations required to during the completion of the missions using different parameter combinations. The x-axis represents the parameter combination used (which can be found in table 5.2), while the y-axis shows the number of iterations required. The figure reveals that achieving a good mission completion time may result in a high number of iterations. For example, choosing combination 2 requires on average 14 iterations to solve each MAPF instance but may require up 80+ iterations in some cases. In the bottleneck-free case, the choice of parameters is a very clear trade-off between mission completion time and the number of iterations required.

Table 5.1 presents the correlation matrix between various indicators and parameters related to the algorithm's performance. The table includes the average transfer time, average number of iterations, maximum number of iterations, collision weighting parameters $\Delta\lambda, \Delta\phi$, other machines path weight parameter α, and skipping ratio parameter ξ. The matrix indicates that there is a strong connection between transfer time and collision weighting parameter, meaning that increasing the collision weighting parameter can significantly increase the time required to complete the mission. Additionally, there is a negative connection (-0.4) between skipping ratio and average iterations, indicating that increasing the skipping ratio parameter may result in a reduction of the average number of iterations required to solve the problem. These main findings can also been seen from figures 5.3 and 5.4. It is important to note that the parameters used are picked within a certain range. Therefore, the results obtained from the analysis may not be generalizable outside of this specific range. For example, from the figures, it is evident that choosing a very small α increases iterations significantly, but this trend does not continue for larger α. In essence, different parameter values may yield different results, and the optimal parameter combination for one problem instance may not be the same for another.

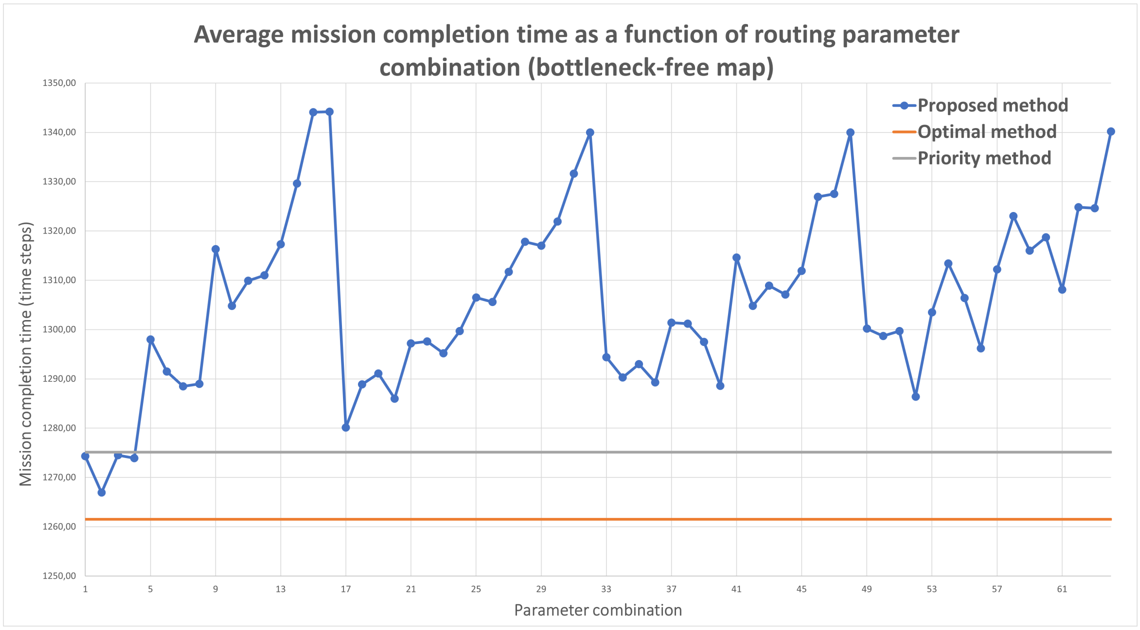


Figure 5.3. *Impact of different parameter combinations on the average completion time of ten 500-container missions conducted by 15 agents on a bottleneck-free map.*

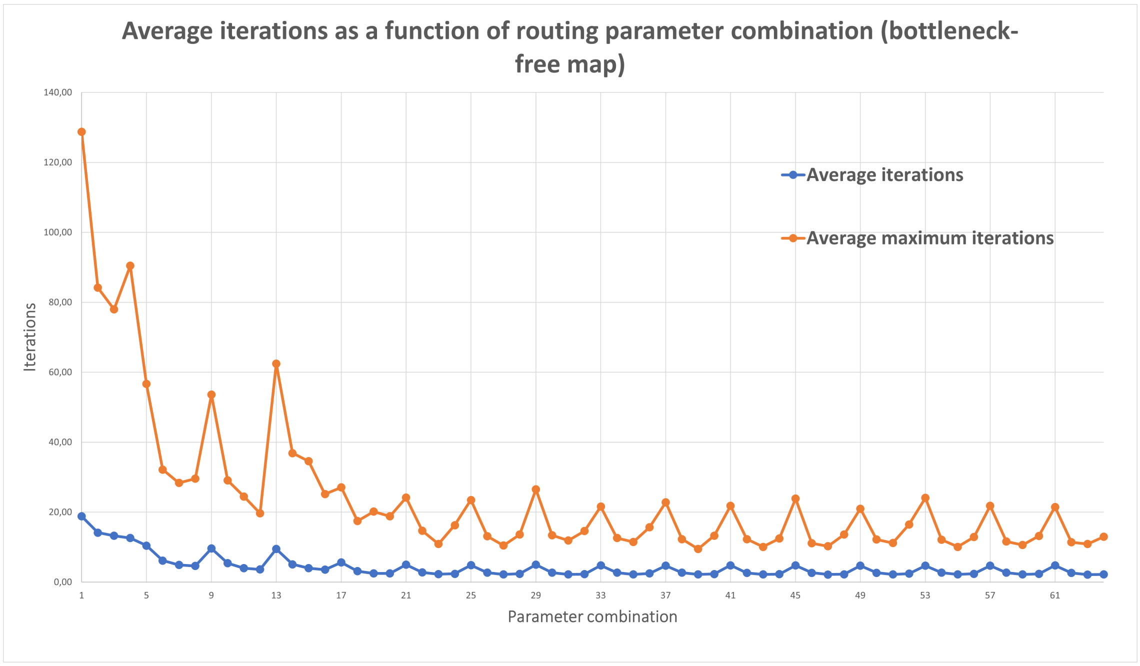


Figure 5.4. *The effect of parameter selection on the average number of algorithm iterations, measured as both the mean of the averages and the mean of the maximums, for ten missions with 500 containers each on a bottleneck map, with 15 agents.*

Table 5.1. *Correlation matrix between various indicators and parameters related to the algorithm's performance based on 10 simulated 500 container missions with 15 agents on a bottleneck-free map.*

	Avg. Time	Avg. Ite.	Max. Ite.	$\Delta\lambda, \Delta\phi$	α	ξ
Avg. Time	1,00	0,12	0,11	0,76	0,17	0,10
Avg. Ite.	0,12	1,00	0,17	0,00	0,05	-0,41
Max. Ite.	0,11	0,17	1,00	-0,07	0,24	0,18
$\Delta\lambda, \Delta\phi$	0,76	0,00	-0,07	1,00	0,00	0,00
α	0,17	0,05	0,24	0,00	1,00	0,00
ξ	0,10	-0,41	0,18	0,00	0,00	1,00

Figure 5.5 provides an analysis of the average time required to transfer a single container as a function of agent amount. The parameter combination 2 is chosen, as it provides the best mission completion time for 15 agents, as seen in figure 5.3. The x-axis represents the number of agents involved in the mission, while the y-axis shows the average time required to transfer a single container. The simulation involves ten times more containers than agents, and the time required for a single container to move from quay cranes to yard cranes is calculated by dividing the total simulation time by 10, which is the number of containers per agent. The process is repeated 100 times for a good average value. The priority method is again used as a reference to evaluate the algorithm's performance. The results indicate that more than 15 agents are required before the algorithm outperforms the priority method. On the other hand, for a low number of agents, the algorithm's performance may be worse than the priority method.

Figure 5.6 represents the relationship between the number of agents and the iterations required by the MAPF algorithm 2 during the mission. The x-axis corresponds to the agent amount, while the y-axis represents the number of iterations. This visualization serves as a form of simulated time complexity analysis, as it captures the average iterations and average maximum iterations needed during the mission. The results indicate a almost linear relationship between the number of agents and the iterations required. The figure suggests that, in the given scenario, the developed algorithm demonstrates a relatively consistent performance as the agent count increases, with only moderate growth in the number of iterations required. It is important to note that this observed relationship between the number of agents and the iterations required is primarily due to the absence of bottlenecks in the map, which allows for smooth and unconstrained navigation, thus minimizing the impact of increasing agent density on the algorithm's performance.

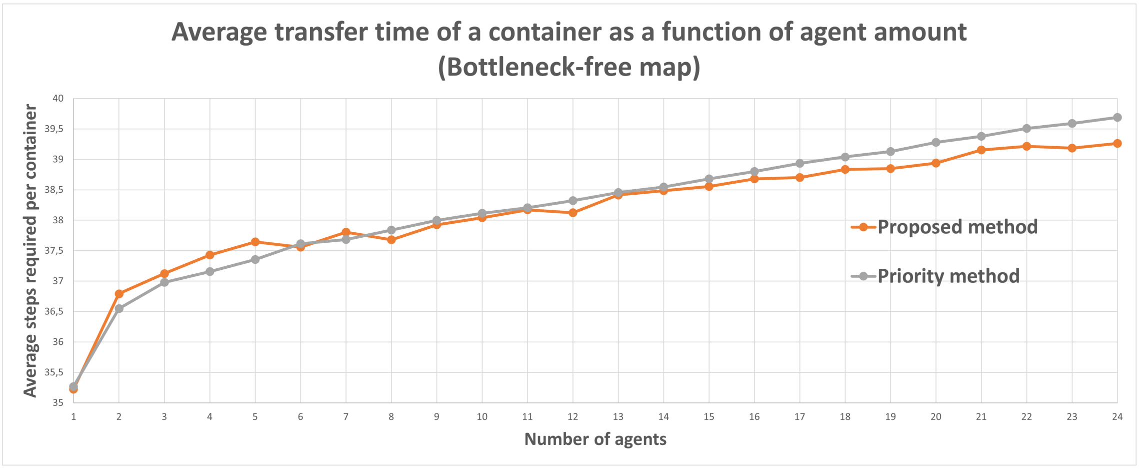


Figure 5.5. *Average time required to transfer a single container as a function of agent amount by choosing parameter combination 2 on a bottleneck-free map.*

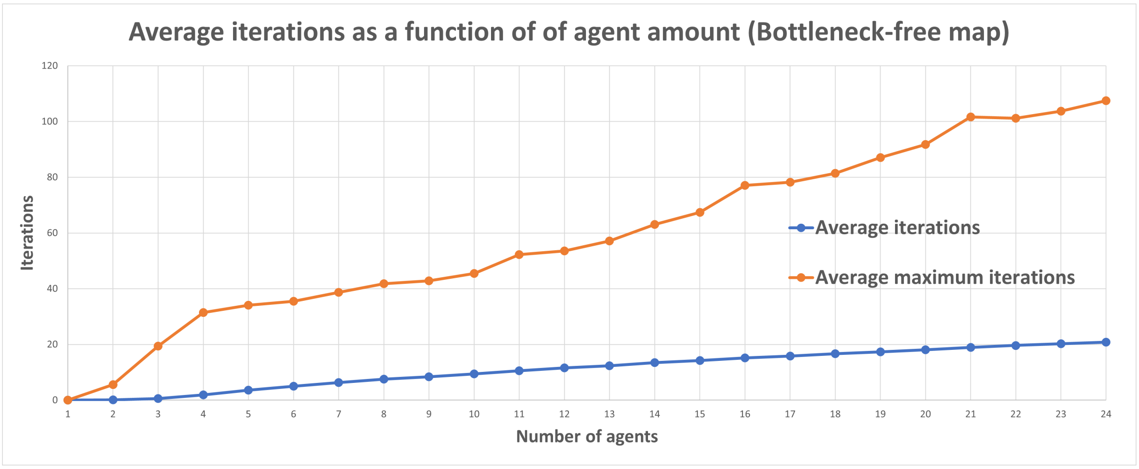


Figure 5.6. *Average iterations required for a single container transfer with parameter combination 2 on a bottleneck-free map as a function of agent amount.*

Table 5.2. *Parameter combinations used in figures 5.3 and 5.4*

Combination	$\Delta\lambda, \Delta\phi$	α	ξ
1	0.1	0.5	0.2
2	0.1	0.5	0.4
3	0.1	0.5	0.6
4	0.1	0.5	0.8
5	0.5	0.5	0.2
6	0.5	0.5	0.4
7	0.5	0.5	0.6
8	0.5	0.5	0.8
9	1	0.5	0.2
10	1	0.5	0.4
11	1	0.5	0.6
12	1	0.5	0.8
13	10	0.5	0.2
14	10	0.5	0.4
15	10	0.5	0.6
16	10	0.5	0.8
17	0.1	2	0.2
18	0.1	2	0.4
19	0.1	2	0.6
20	0.1	2	0.8
21	0.5	2	0.2
22	0.5	2	0.4
23	0.5	2	0.6
24	0.5	2	0.8
25	1	2	0.2
26	1	2	0.4
27	1	2	0.6
28	1	2	0.8
29	10	2	0.2
30	10	2	0.4
31	10	2	0.6
32	10	2	0.8
33	0.1	3	0.2
34	0.1	3	0.4
35	0.1	3	0.6
36	0.1	3	0.8
37	0.5	3	0.2
38	0.5	3	0.4
39	0.5	3	0.6
40	0.5	3	0.8
41	1	3	0.2
42	1	3	0.4
43	1	3	0.6
44	1	3	0.8
45	10	3	0.2
46	10	3	0.4
47	10	3	0.6
48	10	3	0.8
49	0.1	10	0.2
50	0.1	10	0.4
51	0.1	10	0.6
52	0.1	10	0.8
53	0.5	10	0.2
54	0.5	10	0.4
55	0.5	10	0.6
56	0.5	10	0.8
57	1	10	0.2
58	1	10	0.4
59	1	10	0.6
60	1	10	0.8
61	10	10	0.2
62	10	10	0.4
63	10	10	0.6
64	10	10	0.8

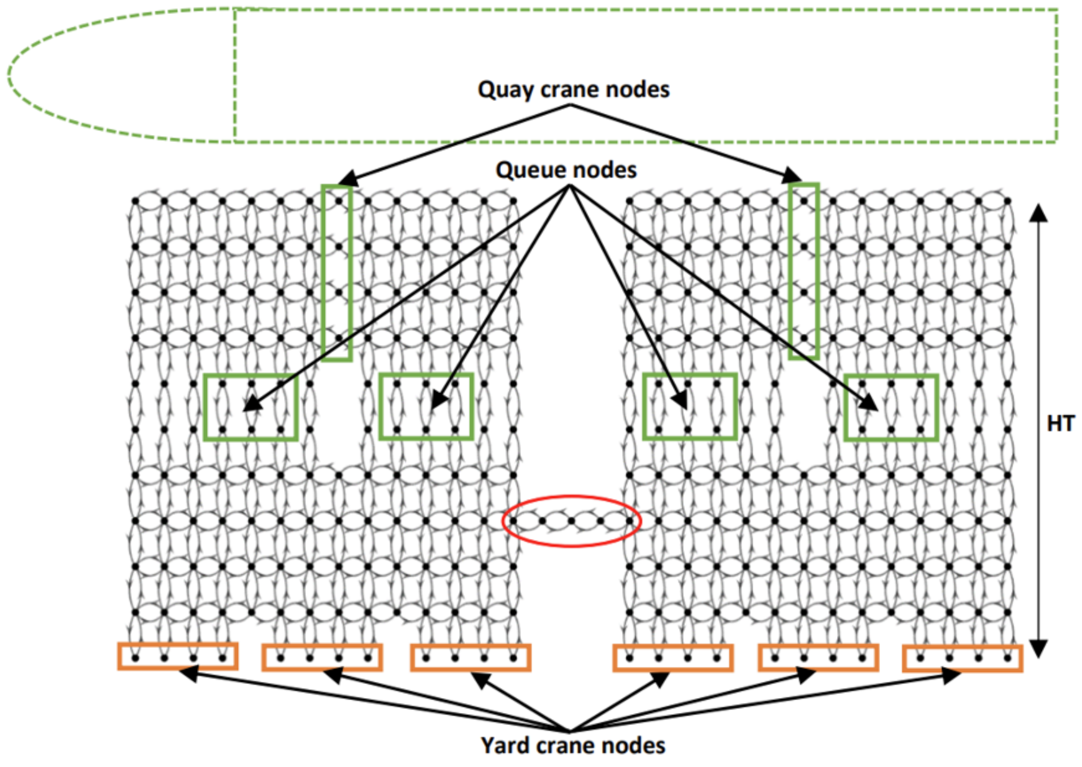


Figure 5.7. *A graph model of a container terminal layout with a conspicuous bottleneck.*

5.3 Bottleneck case

The second case presents a more challenging environment for agent navigation by introducing a bottleneck. The bottleneck divides the terminal layout into two clusters as seen in figure 5.7. The narrow passage between the clusters constraints the available movement options for agents and causes congestion and increased travel distances, despite the relatively low agent density. Simulating this kind of scenario will help to better understand the limitations and potential improvements when dealing with bottlenecks or other challenging layout features. However, the main idea here is to further examine the performance of the developed distributed algorithm 2.

As in the previous case, two tests are conducted. One test to see how the parameter choices affect performance in the presence of a bottleneck, a second test to see how the performance changes as a function of agent amount. The scenario remains the same, only the map layout is changed. In the first test 500 containers are transferred from the quay cranes to yard cranes by 15 machines, and results can be seen in figures 5.8 and 5.9. In the second test, the parameter combination resulting in the lowest mission completion time is chosen and agent amount is varied. The results of the second test are illustrated by figures 5.10 and 5.11.

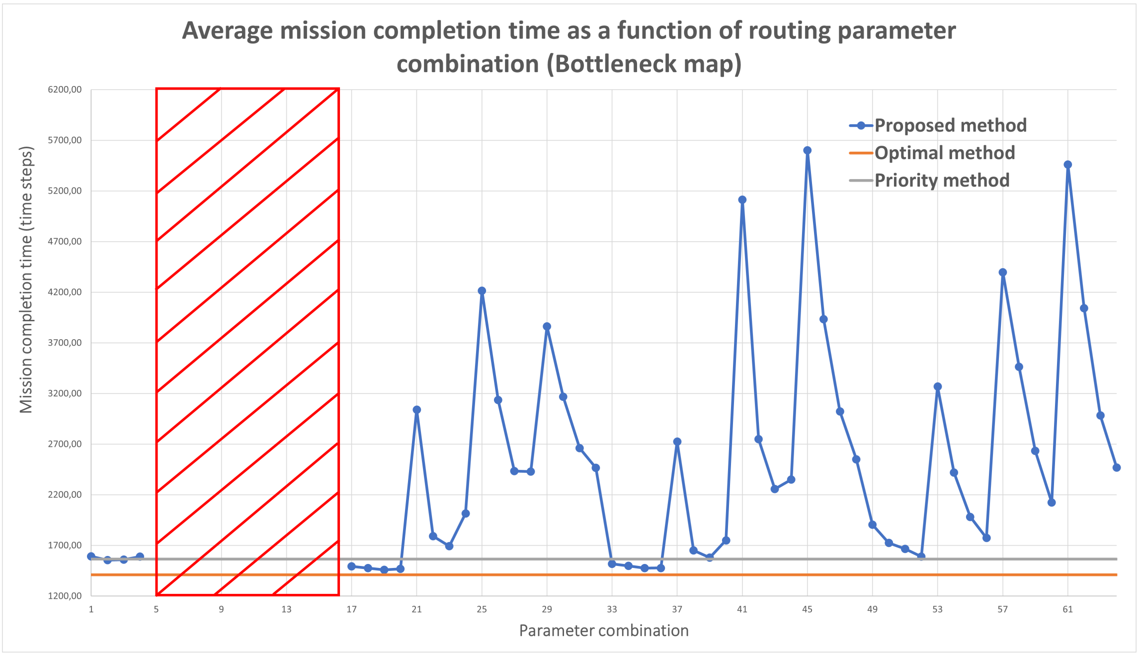


Figure 5.8. *The effect of parameter selection on the average completion time for ten missions, each with 500 containers, executed by 15 agents on a bottleneck map.*

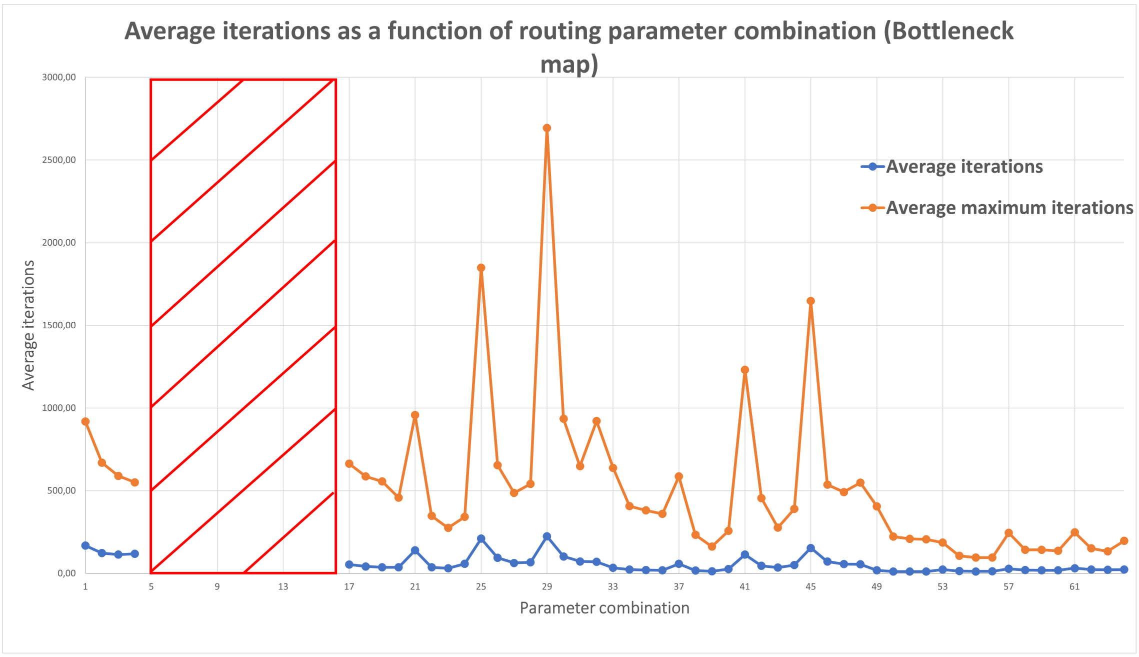


Figure 5.9. *The impact of parameter selection on the average number of algorithm iterations, considering both the mean of the averages and the mean of the maximums, across ten missions with 500 containers each on a bottleneck map, conducted by 15 agents.*

Table 5.3. *Correlation matrix between various indicators and parameters related to the algorithm's performance based on 10 simulated 500 container missions with 15 agents on a bottleneck map*

	Avg. Time	Avg. Ite.	Max. Ite.	$\Delta\lambda, \Delta\phi$	α	ξ
Avg. Time	1,00	-0,34	-0,40	0,55	-0,09	-0,39
Avg. Ite.	-0,34	1,00	0,70	-0,19	0,44	-0,15
Max. Ite.	-0,40	0,70	1,00	-0,21	0,50	-0,07
$\Delta\lambda, \Delta\phi$	0,55	-0,19	-0,21	1,00	-0,08	0,00
α	-0,09	0,44	0,50	-0,08	1,00	0,00
ξ	-0,39	-0,15	-0,07	0,00	0,00	1,00

Figure 5.8 presents the simulated average completion time of ten 500 container transferring missions on a bottleneck map. Similarly to figure 5.3, the average mission completion time on the y-axis is evaluated with different parameter combinations on the x-axis (found in table 5.4). Notably, if the ratio between the priority parameter α and the collision parameters $\Delta\lambda$ and $\Delta\phi$ is too small, the transfer time increases significantly. Therefore for the bottleneck case, combinations 5-16 are completely excluded and the highest value for $\Delta\lambda$ and $\Delta\phi$ is reduced to 1.5. The figure highlights that similar parameter combinations can yield vastly different results due to the bottleneck, a good combination can now be found between 17-20 and 33-36, unlike in figure 5.3. In addition, the gap between the priority method and the optimal method is now ten times wider than without a bottleneck, indicating that the priority method's performance suffers from bottlenecks.

Figure 5.9 presents the relationship between parameter selection and the average number of algorithm iterations, similarly to figure 5.4. The x-axis displays the parameter combinations (5.4) , while the y-axis represents the average iterations during the ten 500-container missions. The graph indicates peaks at low skipping ratios (0.2), while maintaining relative consistency in other instances. Notably, the average number of iterations is significantly higher than in figure 5.4, and the predictability is not as apparent. Nevertheless, the downward trend observed in 5.4, caused by the priority parameter α, is also present in figure 5.9,. This information highlights the even greater importance of carefully selecting parameter combinations on a bottleneck map.

Table 5.3 provides a correlation matrix between various indicators and parameters associated with the algorithm's performance in the bottleneck case, comparable to the findings in table 5.1. The table encompasses the average transfer time, average iterations, maximum iteration collision weighting parameters $\Delta\lambda, \Delta\phi$, other machines path weight parameter α, and skipping ratio parameter ξ. It demonstrates a diminished influence of the collision weighting parameter on transfer time, implying that other factors may be more crucial in determining the algorithm's efficiency in bottleneck environments. Furthermore, a noticeable shift occurs in the relationship between the skipping ratio parameter and av-

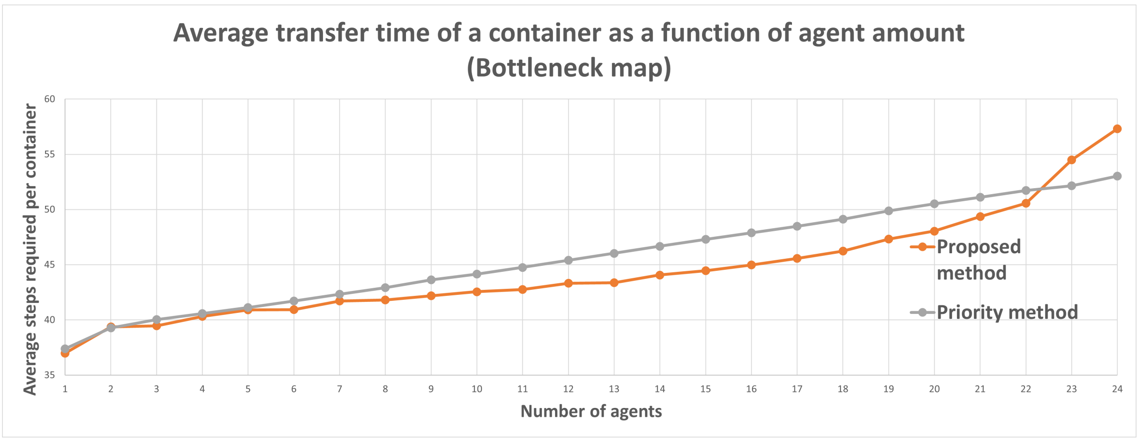


Figure 5.10. *Average time required to transfer a single container as a function of agent amount by choosing parameter combination 19 on a bottlenecked map.*

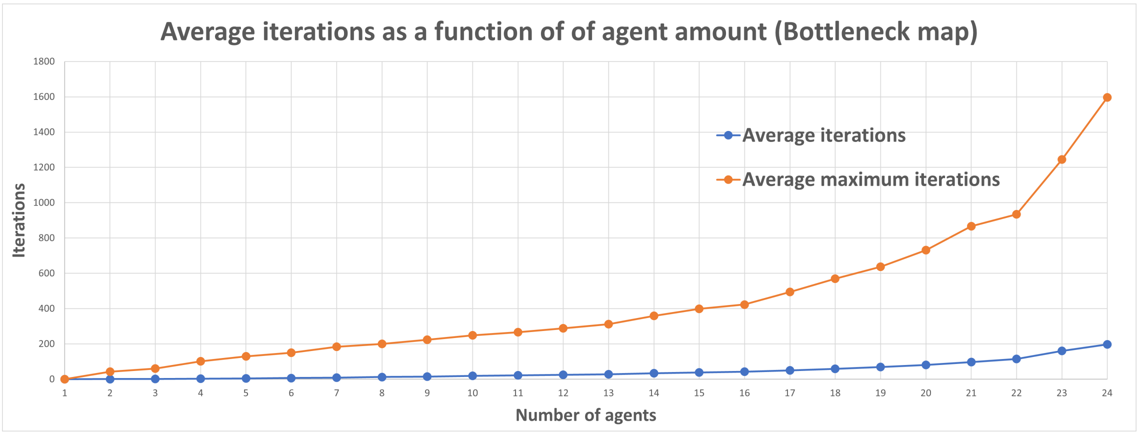


Figure 5.11. *Average iterations required for a single container transfer with parameter combination 19 on a bottleneck map as a function of agent amount.*

erage iterations, now displaying a more pronounced impact on the number of iterations instead of the mission completion time. It is essential to again reiterate that the parameters employed fall within a specific range (see table 5.4), and the analysis results may not apply outside this range. Consequently, caution should be exercised when generalizing these findings.

Figure 5.10 provides an analysis of the average time required to transfer a single container as a function of agent amount in the bottleneck case, using the parameter combination 19. The x-axis represents the number of agents involved in the mission, while the y-axis shows the average time required to transfer a single container. The simulation involves ten times more containers than agents, and the time required for a single container to move from quay cranes to yard cranes is calculated by dividing the total simulation time by 10, which is the number of containers per machine. The process is repeated 100 times for a reliable average value. The priority method is again used as a reference to evaluate the algorithm's performance as in figure 5.5. The results indicate that the proposed algorithm 2 performs clearly below the priority method for 5-22 agents but rapidly surpasses it as the number of agents increases beyond this range. This contrast with the non-bottleneck case 5.5 suggests that the presence of a bottleneck impacts the algorithm's performance in a different way compared to the priority method. One possible explanation for the algorithm's worse performance with a large number of agents could be the increased congestion in the bottleneck areas of the map. As the agent density increases, the algorithm may struggle to find efficient routes for all agents to avoid collisions, leading to a slower overall mission completion time. This highlights the importance of adapting the algorithm's parameters, such as the chosen combination 19, to the specific environment and map layout to ensure optimal performance, especially in scenarios with bottlenecks and varying agent densities.

Figure 5.11 represents the relationship between the number of agents and the iterations required by the MAPF algorithm 2 during multiple missions in the bottleneck case. The x-axis corresponds to the agent amount, while the y-axis represents the number of iterations. Similarly to figure 5.6 his visualization again serves as a form of simulated time complexity analysis, capturing the average iterations and average maximum iterations needed during the mission. In contrast to figure 5.6, the results for the bottleneck case indicate a clear exponential growth in the number of iterations required as more agents are added. At 24 agents, there is, on average, 200 iterations and up to 1600 iterations in some cases, which is significantly higher than the iterations observed in figure 5.6.

This stark difference between figure 5.6 and 5.11 demonstrates the substantial impact that bottlenecks can have on the algorithm's performance. The presence of bottlenecks in the map creates additional navigation constraints and increases the complexity of avoiding collisions for the agents, especially as the agent density rises. As a result, the algorithm requires a considerably higher number of iterations to find feasible solutions in bottleneck scenarios compared to non-bottleneck cases.

Table 5.4. *Parameter combinations used in figures 5.8 and 5.9*

Combination	$\Delta\lambda, \Delta\phi$	α	ξ
1	0.1	0.5	0.2
2	0.1	0.5	0.4
3	0.1	0.5	0.6
4	0.1	0.5	0.8
5	0.5	0.5	0.2
6	0.5	0.5	0.4
7	0.5	0.5	0.6
8	0.5	0.5	0.8
9	1	0.5	0.2
10	1	0.5	0.4
11	1	0.5	0.6
12	1	0.5	0.8
13	1.5	0.5	0.2
14	1.5	0.5	0.4
15	1.5	0.5	0.6
16	1.5	0.5	0.8
17	0.1	2	0.2
18	0.1	2	0.4
19	0.1	2	0.6
20	0.1	2	0.8
21	0.5	2	0.2
22	0.5	2	0.4
23	0.5	2	0.6
24	0.5	2	0.8
25	1	2	0.2
26	1	2	0.4
27	1	2	0.6
28	1	2	0.8
29	1.5	2	0.2
30	1.5	2	0.4
31	1.5	2	0.6
32	1.5	2	0.8
33	0.1	3	0.2
34	0.1	3	0.4
35	0.1	3	0.6
36	0.1	3	0.8
37	0.5	3	0.2
38	0.5	3	0.4
39	0.5	3	0.6
40	0.5	3	0.8
41	1	3	0.2
42	1	3	0.4
43	1	3	0.6
44	1	3	0.8
45	1.5	3	0.2
46	1.5	3	0.4
47	1.5	3	0.6
48	1.5	3	0.8
49	0.1	10	0.2
50	0.1	10	0.4
51	0.1	10	0.6
52	0.1	10	0.8
53	0.5	10	0.2
54	0.5	10	0.4
55	0.5	10	0.6
56	0.5	10	0.8
57	1	10	0.2
58	1	10	0.4
59	1	10	0.6
60	1	10	0.8
61	1.5	10	0.2
62	1.5	10	0.4
63	1.5	10	0.6
64	1.5	10	0.8

6. CONCLUSION

In conclusion, this thesis has presented a distributed algorithm for multi-agent path finding (MAPF) in the context of horizontal transportation in automated container terminals. To evaluate the performance of the algorithm, a simulator was developed in MATLAB and the algorithm was tested on both bottleneck-free and bottlenecked container terminal maps. The tests conducted show the impact of parameter choice and the number of agents on the performance of the algorithm. These results provide valuable insights into how the algorithm can be optimized for different container terminal scenarios.

The results demonstrated that the proposed algorithm can outperform a straightforward priority method, and can produce results close to optimal in some cases. The algorithm's performance is significantly impacted by the presence of bottlenecks, resulting in a higher number of iterations required to find feasible solutions. These findings highlight the importance of adapting the algorithm's parameters to the specific environment and map layout to ensure good performance, especially in scenarios with bottlenecks and varying agent densities.

Any practical implementation of the algorithm is not within the scope of this thesis. However, the presented algorithm and simulation framework provide a solid foundation for further research and development in the field of MAPF in automated container terminals. Overall, this thesis has contributed to advancing the understanding of the challenges and potential solutions for the MAPF problem in the context of horizontal transportation in automated container terminals. The underlying distributed MAPF work is also applicable in other contexts.

In future work, the proposed distributed algorithm could be further improved and extended to account for other factors that impact container transfer tasks in maritime ports. For example, the algorithm could be modified to consider crane scheduling and vehicle charging. Furthermore, the proposed algorithm could be tested and validated in a more realistic container terminal environment to assess its applicability and effectiveness in practice.

REFERENCES

15-Puzzle (2022). URL: https://15puzzle.netlify.app/ (visited on 10/30/2022).

Anvari, Bani et al. (2020). “Comparison of Fleet Size Determination Models for Horizontal Transportation of Shipping Containers Using Automated Straddle Carriers”. In: *Handbook of Terminal Planning*. Springer International Publishing, pp. 73–100.

Bae, Hyo Young et al. (2011). “Comparison of operations of agvs and alvs in an automated container terminal”. In: *Journal of Intelligent Manufacturing* 22.3, pp. 413–426.

Banfi, Jacopo, Nicola Basilico, and Francesco Amigoni (2017). “Intractability of Time-Optimal Multirobot Path Planning on 2D Grid Graphs with Holes”. eng. In: *IEEE robotics and automation letters* 2.4, pp. 1941–1947.

Barták, Roman and Jiří Švancara (2021). “On sat-based approaches for multi-agent path finding with the sum-of-costs objective”. In: *Proceedings of the International Symposium on Combinatorial Search* 10.1, pp. 10–17.

Bartošek, A. and O. Marek (2013). “Quay cranes in container terminals”. In: *Transactions on Transport Sciences* 6.1, pp. 9–18.

Bertsekas, Dimitri P. (2015). *Convex Optimization Algorithms*. Athena Scientific.

Birgin, E. G. and Martínez J. M. (2014). *Practical augmented lagrangian methods for constrained optimization*. SIAM, Society for Industrial and Applied Mathematics.

Boyd, Stephen P. and Lieven Vandenberghe (2021). *Convex optimization*. Cambridge University Press.

Carlo, Héctor J., Iris F.A. Vis, and Kees Jan Roodbergen (2014). “Storage yard operations in Container Terminals: Literature overview, trends, and Research Directions”. In: *European Journal of Operational Research* 235.2, pp. 412–430.

CHU, CHIN-YUAN and WEN-CHIH HUANG (Mar. 2005). “Determining container terminal capacity on the basis of an adopted yard handling system”. In: *Transport Reviews* 25, pp. 181–199.

Digiesi, Salvatore, Francesco Facchini, and Giovanni Mummolo (2019). “Dry port as a lean and green strategy in a container terminal hub: A mathematical programming model”. In: *Management and Production Engineering Review* 10.1, pp. 14–28.

Erdem, Esra et al. (2013). “A general formal framework for pathfinding problems with multiple agents”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 27.1, pp. 290–296.

Goldenberg, M. et al. (2014). “Enhanced partial expansion a*”. In: *Journal of Artificial Intelligence Research* 50, pp. 141–187.

Goldreich, Oded (2011). “Finding the shortest move-sequence in the graph-generalized 15-puzzle is NP-hard”. In: *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pp. 1–5.

Haralambides, Hercules E. (2019). “Gigantism in container shipping, ports and global logistics: a time-lapse into the future”. eng. In: *Maritime economics logistics* 21.1, pp. 1–60.

Hu, Hongtao et al. (2021). “Anti-conflict AGV path planning in automated container terminals based on multi-agent Reinforcement Learning”. In: *International Journal of Production Research*, pp. 1–16.

Jungnickel, Dieter (2008). *Graphs, Networks and Algorithms*. eng. Vol. 5. Algorithms and Computation in Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg.

Kaduri, Omri, Eli Boyarski, and Roni Stern (2020). “Algorithm selection for optimal multi-agent pathfinding”. In: *Proceedings of the International Conference on Automated Planning and Scheduling* 30, pp. 161–165.

Kalai, Gil (1997). “Linear programming, the simplex algorithm and simple polytopes”. In: *Mathematical Programming* 79.1-3, pp. 217–233.

Kemme, Nils (2020). “State-of-the-Art Yard Crane Scheduling and Stacking”. In: *Handbook of Terminal Planning*. Springer International Publishing, pp. 383–413.

Kim, Bokyung, Geunsub Kim, and Moohong Kang (2022). “Study on comparing the performance of fully automated container terminals during the COVID-19 pandemic”. In: *Sustainability* 14.15, p. 9415.

Knatz, Geraldine, Theo Notteboom, and Athanasios A. Pallis (2022). “Container Terminal Automation: Revealing distinctive terminal characteristics and operating parameters”. In: *Maritime Economics amp; Logistics* 24.3, pp. 537–565.

Kumawat, Govind Lal and Debjit Roy (2020). “AGV or lift-AGV? performance trade-offs and design insights for container terminals with robotized transport vehicle technology”. In: *IISE Transactions* 53.7, pp. 751–769.

Ma, Hang (2022). “Graph-Based Multi-Robot Path Finding and Planning”. In: *Current Robotics Reports* 3.3, pp. 77–84.

Ma, Ziyuan, Yudong Luo, and Hang Ma (2021). “Distributed Heuristic Multi-Agent Path Finding with Communication”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8699–8705.

Meindl, Bernhard and Matthias Templ (2013). “Analysis of Commercial and Free and Open Source Solvers for the Cell Suppression Problem.” In: *Trans. Data Priv.* 6.2, pp. 147–159.

Meisel, Frank. (2009). *Seaside Operations Planning in Container Terminals*. 1st ed. 2009. Contributions to Management Science. Heidelberg: Physica-Verlag HD.

Michail, Nektarios, Konstantinos Melas, and Dimitris Batzilis (Jan. 2021). “Container shipping trade and real GDP growth: A panel vector autoregressive approach”. In: *Economics Bulletin* 41, pp. 304–315.

Nishi, T., M. Ando, and M. Konishi (2005). “Distributed route planning for multiple mobile robots using an augmented lagrangian decomposition and coordination technique”. In: *IEEE Transactions on Robotics* 21.6, pp. 1191–1200.

Nocedal, Jorge and Stephen J. Wright (2006). *Numerical optimization*. Springer.

PAN, Ping-Qi (2013). “Duality Principle and dual simplex method”. In: *Linear Programming Computation*, pp. 101–121.

Pyomo - optimization modeling in Python (2013). eng. Springer optimization and its applications, vol. 67. New York: Springer.

Review of Maritime Transport (Nov. 2021). URL: https://unctad.org/webflyer/review-maritime-transport-2021 (visited on 10/30/2022).

Rodrigue, Jean-Paul (2020). *The Geography of Transport Systems*. Routledge.

SAT Competition, SATComp Organizing (2022). *The International Sat Competition Web Page*. URL: http://www.satcompetition.org/ (visited on 10/30/2022).

Sharon, Guni, Roni Stern, Ariel Felner, et al. (2015). “Conflict-based search for optimal multi-agent pathfinding”. In: *Artificial intelligence* 219, pp. 40–66.

Sharon, Guni, Roni Stern, Meir Goldenberg, et al. (2013). “The increasing cost tree search for optimal multi-agent pathfinding”. In: *Artificial Intelligence* 195, pp. 470–495.

Standley, Trevor (2010). “Finding optimal solutions to cooperative pathfinding problems”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 24.1, pp. 173–178.

Steenken, Dirk, Stefan Voss, and Robert Stahlbock (Jan. 2004). “Container terminal operation and operations research - A classification and literature review”. In: *OR Spectrum* 26, pp. 3–49.

Stern, R. et al. (2019). “Multi-agent pathfinding: Definitions, variants, and benchmarks”. In: *Proceedings of the 12th International Symposium on Combinatorial Search, SoCS 2019*, pp. 151–158.

Stern, Roni (Oct. 2019). “Multi-Agent Path Finding – An Overview”. In: pp. 96–115.

Surynek, Pavel (2017). “Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems”. In: *Annals of Mathematics and Artificial Intelligence* 81.3-4, pp. 329–375.

Švancara, Jiří and Roman Barták (2019). “Combining strengths of optimal multi-agent path finding algorithms”. In: *Proceedings of the 11th International Conference on Agents and Artificial Intelligence*.

Tang, Gang et al. (2021). “Geometric A-star algorithm: An improved A-star algorithm for AGV Path Planning in a port environment”. In: *IEEE Access* 9, pp. 59196–59210.

VANDERBEI, ROBERT J. (2021). *Linear Programming: Foundations and extensions*. SPRINGER.

Vis, Iris F.A. (2006). “A comparative analysis of storage and retrieval equipment at a container terminal”. In: *International Journal of Production Economics* 103.2, pp. 680–693.

Wang, Zehao and Qingcheng Zeng (2022). “A branch-and-bound approach for AGV dispatching and routing problems in Automated Container Terminals”. In: *Computers amp; Industrial Engineering* 166, p. 107968.

Yu, Hang et al. (2022). “Yard Operations and Management in Automated Container Terminals: A Review”. In: *Sustainability (Basel, Switzerland)* 14.6, pp. 3419–.

Yu, Jingjin and Steven M. LaValle (2013). “Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs”. In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. AAAI’13. AAAI Press, pp. 1443–1449.

Zhou, Chenhao, Byung Lee, and Haobin Li (June 2020). “Integrated optimization on yard crane scheduling and vehicle positioning at container yards”. In: *Transportation Research Part E: Logistics and Transportation Review* 138, p. 101966.

APPENDIX A: MAPF CODE

A.1 Optimal MAPF with Pyomo

```
# MAPF with Pyomo

import os
import pyomo.environ as pyo
import importlib

# Data importation not included for clarity

# List of imported sets:

# Es := starting edges
# E := set of edges
# N := set of nodes
# A := set of agents
# K := set of times instances: (0,1,2,... horizon)
# E_in := set of pairs: {node, edges going to node}
# E_out := set of pairs: {node, edges going out of node}
# E_swaps := set of pairs: {edge, "swap" edge}
# c := matrix: {edges, agents, value: 0 at goal, 1 elsewhere}


############################################################

# 1. Model creation

# Concrete model
model = pyo.ConcreteModel()

# Binary decision variables
model.x = pyo.Var(A,E,K, within=pyo.Binary)
```

```
# Constraint creation

# (1)
# Starting nodes are fixed
for a in range(0,len(Es)):
    model.x[a,Es[a],K[0]].fix(1.0)

# (2)
# Adjacency constraints
def adjacency_rule(mdl,a,n,k):
    return   sum(mdl.x[a,e,k] for e in E_in[n-1]) \
        == sum(mdl.x[a,e,k+1] for e in E_out[n-1])
model.rule2 = pyo.Constraint(A,N,K[:-1],rule=adjacency_rule)

# (3)
# Agent number constraints
def number_rule(mdl, a,k):
     return sum(mdl.x[a,e,k] for e in E) == 1
model.rule3 = pyo.Constraint(A,K, rule=number_rule)

# (4)
# Collision constraints
def approach_rule(mdl,n,k):
  return   sum(sum(mdl.x[a,e,k] for e in E_in[n-1]) \
for a in A) <= 1
model.rule4 = pyo.Constraint(N,K,rule=approach_rule)

# (5)
# Swapping constraints
def swapping_rule(mdl,i,k):
   return   sum(mdl.x[a,E_swaps[i][0],k] + \
mdl.x[a,E_swaps[i][1],k] for a in A)<= 1
I = [i for i in range(len(E_swaps))]
model.rule5 = pyo.Constraint(I,K[:-1],rule=swapping_rule)

# Objective
def obj_rule(mdl):
    return  sum(mdl.x[a,e,k]*c[e-1][a-1] \
for a in A for e in E for k in K)
```

```
model.obj = pyo.Objective(rule=obj_rule,sense=pyo.minimize)

############################################################

# 2. Solving the MAPF problem

# Choosing solver: cplex, gurobi, glpk, cbc, clp
# Note: Only glpk is included in the Pyomo package,
# other solvers will need to be acquired elsewhere
solver = pyo.SolverFactory("cplex")

# Solve the problem
res = solver.solve(model)

# Report success/failure
pyo.assert_optimal_termination(res)

# Final solution
solution = model.x

############################################################
```