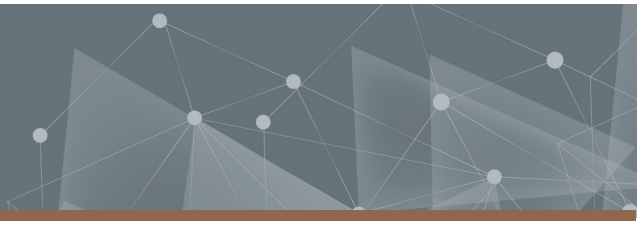
CHALMERS
UNIVERSITY OF TECHNOLOGY



Evaluating and Optimizing Fleet Management and Task Allocation Techniques for Multiagent Systems

Master's thesis in System, Control and Mechatronics

ALBIN BENGTSSON
DAVID WANNHEDEN

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Evaluating and Optimizing Fleet Management and Task Allocation Techniques for Multiagent Systems

ALBIN BENGTSSON
DAVID WANNHEDEN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2022

Evaluating and Optimizing Fleet Management and Task Allocation Techniques for Multiagent Systems
ALBIN BENGTSSON
DAVID WANNHEDEN

Advisor: Fredrik Hagebring, Kollmorgen Automation AB
Supervisor: Sabino Roselli, Department of Electrical Engineering
Examiner: Knut Åkesson, Department of Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Evaluating and Optimizing Fleet Management and Task Allocation Techniques for Multiagent Systems
ALBIN BENGTSSON
DAVID WANNHEDEN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

As more factories and warehouses around the world implement autonomous multi-agent systems to execute tasks, the demand for optimization of order allocation and path planning grows. In this thesis a multi-step optimization algorithm is presented, with the aim of avoiding congestion in large scale environments. The solution employs both Mixed Integer Linear Programming and Satisfiability Modulo Theories to compute solutions for an online setting where orders are generated continuously and vehicles are routed in an undirected graph that represents the layout.

It is shown that the solution proposed is able to avoid congestion based on a high level analysis. However, since the solution has only been tested with a time-step based simulation further evaluation would be necessary to assess the quality of the solution.

Keywords: AMR, Automated Mobile Robot, VRP, Vehicle Routing Problem, Optimization, MILP, Mixed Integer Linear Programming, SMT, Satisfiability Modulo Theories, CAVRP, Congestion Avoidance Vehicle Routing Problem.

Acknowledgements

We would like to take this opportunity to thank Kollmorgen AB and especially Fredrik Hagebring who have guided us with passion throughout this project.

We would also like to extend our gratitude towards our supervisor at Chalmers, Sabino Roselli, who have helped a lot with the optimization tools used as well as great discussions regarding optimization of vehicle routing problems. Also, we would like to thank our examiner Knut Åkesson for the guidance in writing a good report.

Albin Bengtsson and David Wannheden, Gothenburg, June 2022

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AGV	Automated Guided Vehicle
AMR	Automated Mobile Robot
CAVRP	Congestion Avoiding Vehicle Routing Problem
CFEVRP	Conflict Free Electric Vehicle Routing Problem
CFVRP	Conflict Free Vehicle Routing Problem
ETA	Estimated Time of Arrival
FCFS	First Come First Serve
FMS	Fleet Manager System
IA	Instantaneous Assignment
MILP	Mixed-Integer Linear Problem
MR	Multi-Robot Tasks
MRTA	Multi-Robot Task Allocation
MT	Multi-Task Robots
SLAM	Simultaneous Localization And Mapping
SMT	Satisfiability Modulo Theories
SR	Single-Robot Tasks
ST	Single-Task Robots
TA	Time-Extended Assignment
TOC	Temporal and Ordering Constraints
TSP	Travelling Salesperson Problem
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

c	Index for cluster
o	Index for order
v	Index for vehicle
e	Index for edge
n	Index for narrow edge
p	Index for path alternative

Sets

$\mathcal{A}_⌋$	Set of order-vehicle assignments in cluster c
$\mathcal{C}$	Set of order clusters
$\mathcal{O}$	Set of orders
$\mathcal{S}$	Set of vehicles
$\mathcal{V}$	Set of vehicles
$\mathcal{V}_c$	Set of vehicles in cluster c
$\mathcal{O}_c$	Set of orders in cluster c
$\mathcal{V}_p$	The set of vehicles that require path allocation optimization
$\mathcal{P}_v$	The set of paths generated for each vehicle
$\mathcal{E}_v$	The set of edges used by a vehicle
$\mathcal{N}_v$	The set of narrow edges used by a vehicle

Parameters

t_o	Determines how often an optimization cycle is computed
α	Scaling parameter for cluster size
γ	Maximum number of orders per cluster
σ	Buffer time for detecting regular encounters
τ	Critical Buffer time for detecting critical encounters
κ	Cost of regular encounter
λ	Cost of critical encounter

Variables

t	Current time-step in the system
δ_v	Current ETA of each vehicle
ϕ	Time buffer used for determining regular encounters
Φ	Time buffer used for determining critical encounters
d_{cv}	Euclidean distance between cluster c and vehicle v
y_{cv}	Binary decision variable that describes whether cluster c is assigned to vehicle v
y_{ov}	Binary decision variable that describes whether order o is assigned to vehicle v
z_{vp}	Boolean decision variable that describes whether vehicle v should take path p
β	Booking encounter
ω	Optim encounter
B	Critical booking encounter
Ω	Critical optim encounter
p_{vp}	Path encounters
π_{vp}	Critical path encounters
w_{vp}	Weight of path

Contents

List of Figures

List of Tables

1

Introduction

There is a growing demand in multi-agent system implementations in factories, warehouses and other similar settings where items need to be moved from one station to another [1].

In a warehouse setting there may typically be a large volume of orders that need to be carried out that are more basic in nature (such as collecting items for e-commerce orders) [19]. Although, in a factory setting there may be more time-sensitive tasks, such as materials being brought to a machine for an operation which may take a variable amount of time. At the end of the operation, one would need to collect the finished product and load up the machine with new raw material, and doing so in an efficient manner may be extremely important in order to maximize throughput of an expensive piece of machinery.

However, factories and warehouses are not the only settings in which multi-agent systems have been implemented for carrying out transportation tasks. Hospitals are an example of a public environment where multi-agent systems have been implemented in order to help with transportation of medical supplies [13], and some restaurants use robot servers for transporting food from the kitchen to the clients seated at tables. In settings like these the challenges involved concern making autonomous systems carry out tasks where people may not be aware of how the systems work and may act unpredictably. Also, restaurants and hospitals may have layouts that are more difficult to navigate than factories and warehouses, which generally feature large open spaces with a single ground floor.

Autonomous Mobile Robots (AMR) are state of the art technologies for such applications [17]. However, in order to have several AMRs sharing a working space a fleet manager system (FMS) is required.

An FMS can be designed in many different ways, according to the type of AMRs it manages and the environment in which they operate. However, generally it could be said that an FMS takes in orders from a front-end system, for example a warehouse managing tool. From thereon, the FMS can retrieve parameters from the AMRs it is currently managing, such as battery level, position, etc. The FMS allocates orders to vehicles and computes a path for the vehicles to reach their goals. The order allocation could be made on a first come first serve basis, where the closest available vehicle is assigned to the latest order and a pathing algorithm computes

the shortest path to its destination. However, it is possible to achieve better results by optimizing this process [18].

There exist general purpose solvers that can be used to solve optimization problems such as these. Satisfiability Modulo Theories (SMT) solvers [2] as well as Mixed Integer Linear Programming (MILP) solvers [5] can be used to find solutions, or in this case plans for which order each vehicle should serve and which paths they should take. However, depending on how many vehicles there are and which parameters the solver is supposed to take into consideration when computing the solution, this could become computationally intensive. The solution is calculated with respect to a chosen cost function, such as time spent finishing a task or the total travel distance of all vehicles. It is generally the case that the solver can quickly find a decent solution [15] depending on how the problem is formulated, while finding the best solution (and thus knowing it is the best solution) takes much more time.

There are several different challenges regarding implementations of this type mainly related to the fact that multiple vehicles utilizing the same space can sometimes lead to undesirable situations, where vehicles find themselves in a *deadlock situation* and cannot reach their goal. This could be caused by situations such as two vehicles accessing a narrow corridor from opposite sides simultaneously, or two vehicles trying to access the same feature of the environment such as a delivery station at the same time. All of these challenges require different solutions based on how the problem is approached as a whole. Therefore many companies are developing their own fleet managing systems with their own optimization algorithms according to their needs and what they believe will give the best results.

With this in mind an optimization algorithm for an FMS will be developed and implemented with the help of MILP solvers as well as SMT solvers in order to optimize task allocation of a multi-agent AMR system.

1.1 Vehicle Routing Problem

The problem at the base of this task is commonly referred to as the Vehicle Routing Problem (VRP) and has been studied for a long time. An example of a VRP is the well-known Travelling Salesperson Problem (TSP), where given a list of location and paths with the calculated distance of each path the objective is to compute the shortest possible that visits each location exactly one and returns to the origin location. In 1959 the so called “Truck Dispatching Problem” was published by Dantzig and Ramser [4] which detailed how to model a problem where a fleet of trucks was supposed to deliver oil to gas stations from a central hub while minimizing travel distance. This problem was five years later generalized into a linear optimization problem by Clarke and Wright [3], and has since then evolved significantly over time.

To summarize the generalized VRP there are a certain amount of nodes or “customers” which are connected by edges or “roads” and can be visited in different orders. Based on this information the task is to solve how to choose a path to visit all customers while minimizing the distance traveled.

Today VRPs can be modeled in many different ways in order to take into consideration characteristics and complexities of real life, such as time windows for pickup and delivery and order of execution of tasks.

1.2 AGVs and AMRs

The VRP of interest today sometimes involves autonomous systems, rather than vehicles driven by humans. Two types of such systems are described here. Both an Automated Guided Vehicle and Autonomous Mobile Robot, or AGV/AMR, respectively, are vehicles, typically forklifts or other vehicles used for transportation within a warehouse or similar industrial environments [19].

1.2.1 Automated Guided Vehicles

An automated guided vehicle (AGV) is typically a vehicle used for transportation. It could be a forklift, a robot which moves cars in an automated garage etc. AGVs require a pre-defined map and some kind of markers within the environment that the map represents. It uses these markers to establish where in the map it is located. Markers that are typically used are lines, ArUco markers or QR-codes as well as reflectors.

The AGV follows trajectories strictly and these trajectories need to be included in the pre-defined map. This makes AGVs rather reliable and predictable which could be considered a pro regarding safety. However, if there is an obstacle placed on the planned trajectory most conventional AGVs will simply wait until the obstacle has is no longer in the way, which means that if the obstacle is an inanimate object such as a pallet the AGV could be delayed significantly in reaching its goal.

1.2.2 Automated Mobile Robots

Automated mobile robots (AMRs) are similar to AGVs and are used for similar purposes, however the most crucial aspect in which they differ is that AMRs do not have any stored trajectories for how to navigate between stations [17]. Instead the vehicles continuously scan their environments generating a map which an algorithm such as SLAM (Simultaneous Localization And Mapping) [20] can use to generate trajectories for the vehicles. This causes the AMR to be able to handle temporary obstacles better than an AGV since it can update its trajectory in real time as it is moving and detecting the obstacles. While obstacle avoidance is useful to make the system more flexible it also means an increase in difficulty of understanding whether or not a situation could cause a deadlock, since planning paths for AGVs centrally means having a better idea of what the actual path taken by the vehicles will be.

1.3 General Purpose Solvers

The optimization problems presented in this thesis have been modeled as MILP and SMT problems. By using MILP one can solve any kind of real world problem that

can be formulated as a linear problem, however SMTs allow for logical conditions over literals, which makes certain problems easier to formulate. In this thesis the optimization problems that were simpler to formulate were solved using a MILP solver, while an SMT solver was used for the rest. Furthermore, the solvers were mainly treated as a black box tool and focus was put more into formulating the VRP correctly rather than understanding how to use the solvers in the best way. For better understanding of MILP and SMT, see [5] and [2] respectively.

1.4 Research Questions

The purpose of this thesis is to develop and evaluate an optimization algorithm to be used by an FMS for AMRs. The optimization to be made is to allocate available vehicles to execute orders from an order list and decide which path each vehicles should take. General purpose solvers will be used for this endeavor. The solution should take into consideration the most common challenges that are being discussed in the industry today, which means that the aim is to develop a multi-faceted optimizer that could be used in a real product, rather than focusing entirely on one aspect of the problem. The quality of the solution will then be evaluated to a benchmark that can be understood so that the solution can be compared to other solution presented in similar published research.

The project task was assigned by Kollmorgen as part of their research on fleet optimization of AMR systems. The authors were given a great deal of freedom in choosing which issues to focus on, however they expressed an interest in knowing more about how to compute solutions in real time for large scale systems, and how to route vehicles so that the overall system can operate smoothly and independent of human supervision.

The research questions can thus be narrowed down to:

- How does one compute an efficient solution that maximizes order throughput within a time frame that allows the system to run smoothly while also scaling well with larger environments, fleets and order lists?
- How do different specifications in form of number of vehicles, orders and stations affect the performance of the system? Which of these variables will be the biggest issue in regards to the complexity of the problem?

1.5 Related Work

In this section work related to what will be presented in this report is presented.

1.5.1 Conflict-Free Electric Vehicle Routing Problem

The conflict-free electric vehicle routing problem or CFEVRP [16] differs mainly from the ordinary VRP in the fact that it is conflict free. This means that vehicles should not block each other's way by traveling simultaneously through road segments whose geometry impedes them. Note here that road segments are considered to have capacity constraints, meaning that only a certain number of vehicles may travel across each segment at a time.

1.5.2 Vehicle Routing Problem with Time Windows

The vehicle routing problem with time windows (VRPTW) which is similar to the original VRP but with the additions of time windows. This means that the solution accounts for time windows for when a vehicle can access an edge or a node [12]. A time window features an upper and a lower bound, which indicate the earliest possible time and the latest possible time that an order can be served. This could be used to model for example constraints for heavy machinery that can only serve one product at a time, where there is an interest in delivering new raw material and collecting the manufactured product as soon as a production cycle has finished.

2

Problem Definition

In this chapter the problem is defined on a high level and some background is given to the industrial point of view for the problem.

2.1 Industrial Challenges

One of the main issues is to understand what constitutes a good result and how to design a desirable cost function that the optimizer can minimize. Intuitively one could imagine that aiming to minimize the distance traveled by the vehicles or maximizing the throughput of orders could be the best solution. This is what is often seen in academic publications since it offers a simple and reliable benchmark to compare different types of optimization algorithms, however, in a practical implementation there are many other factors to take into consideration, and longer travel distances may be acceptable if the solution fulfills other desirable criteria.

2.1.1 Vehicle Routing

An important goal of the research is to be able to understand how to develop a system of this kind that runs smoothly without need of supervision. There may be instances where one would rather send a vehicle on a long path to their destination in order to avoid encountering another vehicle causing a scenario that can only be resolved with human intervention. There may also be instances where an order could be carried out faster if a vehicle simply waited for a certain space to become clear before proceeding. However, if it took a longer route around the obstacle it would be traveling longer at full speed giving a better impression of an efficient system in the eyes of a potential customer interested in purchasing a multi-agent system. Since these issues do not affect the throughput of orders they can seem uninteresting to solve from an academic viewpoint, however they can be extremely important in order to sell a product. A company may be able to promise great results in terms of throughput, but if the client sees machines idling or struggling in crowded situations they may lose interest in the product.

2.1.2 Scalability

Another aspect of the sought implementation is that it should be possible to use it on large scale systems. Depending on the approach taken, finding a solution could mean a huge difference in required computation as the system grows. With this in mind the task is to develop and evaluate an optimization algorithm that scales well for large implementations while also reliably delivering solutions for carrying out orders efficiently without causing vehicles to stay idle for too long or end up in difficult situations that require manual reset from human operators.

2.1.3 Types of Tasks

Depending on the environment in which the AMRs operate there could be different types of tasks that need to be carried out. The most basic type of tasks (in the context of path optimization) is for the vehicle to simply move to a certain location. This could work for a vehicle that performs cleaning operations, but is otherwise quite limited in terms of functionality. In the context of warehouses the most common type of operation is of the type *pick up* and *drop off*. This is a common scenario to work with, since modeling orders of higher complexity generally means branching out into more specific functionality.

2.1.4 Time Windows

Orders can generally always be considered to be time sensitive, in the sense that they should preferably be carried out as soon as possible. However, in certain situations it may be necessary that a vehicle is located at the order target destination in a specific time window, such as when a machine requires a vehicle to deliver a product which then after the operation is completed needs to be brought to the next station. In cases like this, it is important that the vehicle is punctual in terms of arriving within said time window. This is done in order to not let expensive machinery remain idle.

2.1.5 Order Throughput/Starvation

Since optimized solutions do not serve orders on a first come first serve basis, it is important to make sure that certain orders are not left unattended for too long. This can be addressed by using a priority system, so that the optimizer will sometimes favor older standing orders instead of new ones, even if there is not a vehicle is nearby to serve it.

2.2 Online System Representation

Testing solutions on a real environment is generally not an option when it comes to big scale optimization, since it requires a vast amount of hardware and space, and simulations need to be executed for a long time since the benefits of optimized solution are not expected to be apparent in the short term. This is an issue for AMRs

since their low-level path planning is less predictable than it is for AGVs, whose behavior is much simpler to simulate. Therefore, in order to be able to evaluate the quality of the solution there needs to be some sort of online system representation. A more advanced solution could involve physics engines and simulation of how AMRs localize themselves in an environment in order to avoid obstacles, while a simpler solution could use time-step updates of an environment described on a higher level.

2.3 Taxonomy

Since VRPs may differ significantly in nature it is useful to classify them according to their features and limitations, and to what their core functionality of their solution is. Furthermore, since this is still an emerging field some conventions are not yet properly defined.

A paper published by Gerkey and Mataric [8] proposes a taxonomy for multi-robot systems that provides a starting point for classifying multi-robot task allocation (MRTA) problems. They use the following three axes in order to describe MRTA problems:

- **Single-task robots (ST)** vs. **Multi-task robots (MT)**: ST means that each robot is capable of executing as most one task at a time, while MT means that some robots can execute multiple tasks simultaneously.
- **Single-robot tasks (SR)** vs. **Multi-robot tasks (MR)**: SR means that each task requires exactly one robot to achieve it, while MR means that some tasks can require multiple robots.
- **Instantaneous assignment (IA)** vs. **Time-extended assignment (TA)**: IA means that the available information concerning the robots, the tasks, and the environment permits only an instantaneous allocation of tasks to robots, with no planning for future allocations. TA means that more information is available, such as the set of all tasks that will be assigned, or a model of how tasks are expected to arrive over time.

Combining these axes generate eight different types of MRTA which can be used to classify a VRP with more than one agent. The types of tasks to be carried out in an MRTA have also been classified in literature. One such type of taxonomy is the one proposed by Zlot [21]:

- **Elemental Task**: A task which cannot be decomposed into sub-tasks and which is allocated to a single robot.
- **Decomposable Simple Task**: A task which can be decomposed into elemental tasks in exactly one way, however all those elemental tasks must be carried out by the same robot.
- **Simple Task**: Either an elemental task or a decomposable simple task.

- **Compound Task**: A task which can be decomposed into elemental tasks in exactly one way, which can be carried out by different robots.
- **Complex Task**: A task which can be decomposed into sub-tasks in more than one way, and where at least one of the possible decompositions can be allocated to different robots.

2.4 Assumptions and Delimitations

An important aspect of the project is understanding how to properly assess the quality of the solutions returned by the optimization algorithm, in order to be able to analyze the strengths and weaknesses of the developed product. The best way to do so would be through a mix of testing in real environments and simulations. Testing with a simulation is useful in order to be able to collect data over large amounts of iterations in many different type of layouts with different amounts of vehicles and orders. Furthermore, testing in a physical environment could be useful to confirm that the algorithm can handle unforeseen events that may not be modeled properly in the simulator, for example if a vehicle breaks down or if an there is an unforeseen obstacle such as a human in the pathway. However, this is generally not a possibility since having an environment for testing an optimization algorithm on hundreds of vehicles would be extremely uncommon. This means that the evaluation process of the solution may appear somewhat limited.

2.5 Focus

The reason why Kollmorgen offered this project was to gain a better understanding of how they wanted their FMS to work, and therefore, with the given purpose, the project focuses more on developing a versatile and scalable architecture as a proof of concept rather than delivering a complete product that is ready for implementation.

Out of the various possible desirable objectives that were presented in this chapter the authors chose a combination that they believed would yield the most valuable insight. These were chosen with respect to the research questions presented in section 1.4.

3

Problem Formulation

In this chapter the various challenges presented in the problem definition will be addressed alongside the reasoning behind the design choices.

3.1 The Congestion Avoiding VRP

The Congestion Avoiding VRP (CAVRP) can be described by using the MRTA taxonomy as defined in section 2.3 as an **ST-SR-TA** problem, which means that the vehicles in the system are able to execute one task at a time, and each task requires only one vehicle, and that there is possibility of allocating tasks with regard for future planning.

The reasoning behind developing a system that avoids congestion stems from the fact that AMRs are able to avoid only some obstacles, and knowing what defines an unavoidable obstacle is very difficult. For this reason, it was thought that developing a system where vehicles do not have to resolve situations that are difficult for the vehicles to navigate in the first place could be useful. In this context congestion reduction is referred to as the sum of all encounters between vehicles. Encounters are expanded upon in the subsection 3.5.3.

Since typical warehouse tasks consists of moving around products the tasks solved in this thesis can be classified according to the MRTA taxonomy from section 2.3 as **Decomposable Simple Tasks**, since the task of moving something can be decomposed into a *pick up* operation and a *drop off* operation which must be carried out by the same vehicle.

In regard to future planning, the system will be utilizing a backlog with a number of orders that most of the time will be larger than the number of vehicles. However, there is functionality to handle when there are more vehicles than orders as well. Since this is an online solution, this means that orders will be generated continuously as the vehicles carry out orders, in a way such that there is always a significant backlog for the AMRs to choose from. While this will not always hold true in a real setting, it is the scenario which we are interested in optimizing, since it offers most room for improvement.

In the previous chapter it was mentioned that the proposed solution should take into

consideration how to solve the problem in large scale environments, which means that it is necessary to understand how to define the movement of the vehicles in the layout in such a way that congestion avoidance can be achieved in solutions that are calculated in a reasonable time frame.

3.2 Environment

Deciding how the environment or the layout should be represented is fundamental when solving a path planning problem such as the one involved in this project. The AMRs build up a detailed reconstruction of the environment in which they operate in order to correctly compute their low-level trajectory planning and obstacle avoidance.

However, these detailed features are not necessary to take into consideration when optimizing path planning on a high level. This is similar to how one would plan the path of driving a car, where one only considers the major intersections and routes to take but not which lane they would stay in or how to avoid obstacles such as parked cars and pedestrians. Furthermore, one would generally not plan which parking spot exactly to park in at the beginning of the journey, but only plan the route to an area where parking spaces are available close to the destination. For the AMRs, which in a big environment might have several tens of thousands of stations in form of shelves where they could pick up and drop off payloads. Therefore it can be useful to optimize their route to the general area where they are supposed to carry out an order and from there let them decide on their own how to finalize the pathing with their own trajectory planning.

With these assumptions in mind, it is therefore possible to use an undirected graph to describe the environment, where the nodes represent important features such as intersections, choke points, and clusters of stations and the edges represents the general path between the nodes. Abstracting the environment as a graph structure is one of the simplest possible representations of a two-dimensional layout, which is important in order to ensure scalability. This derives from the fact that for a huge number of vehicles an even bigger number of paths have to be computed in each optimization cycle. If the path can be represented with a list of nodes it is possible to make huge savings in terms of computation power and memory.

The graph is designed to be undirected which means that vehicles are able to travel in both directions along the edges. This means that the graph does not need to be cyclic since vehicles will not get stuck in a sink state. However, there could be corridors in a layout where if two vehicles would enter simultaneously from opposite directions a deadlock situation could arise. These edges are defined as *narrow edges*, and are handled differently in the path planning. This is expanded upon later on.

An example of a map like this can be seen in figure 3.1, where a small office layout is abstracted into a graph with 12 nodes, while the layout in reality contains 55 stations. Note here that the edges in the graph are not corresponding to the trajectories that a vehicle will take, AMRs will compute trajectories independently from

one node to the other using SLAM. This means that the sparse graph is enough for indicating to the vehicle which corridors it should travel across to reach its destination, which represents the abstraction level of interest in terms of path planning in the CAVRP.

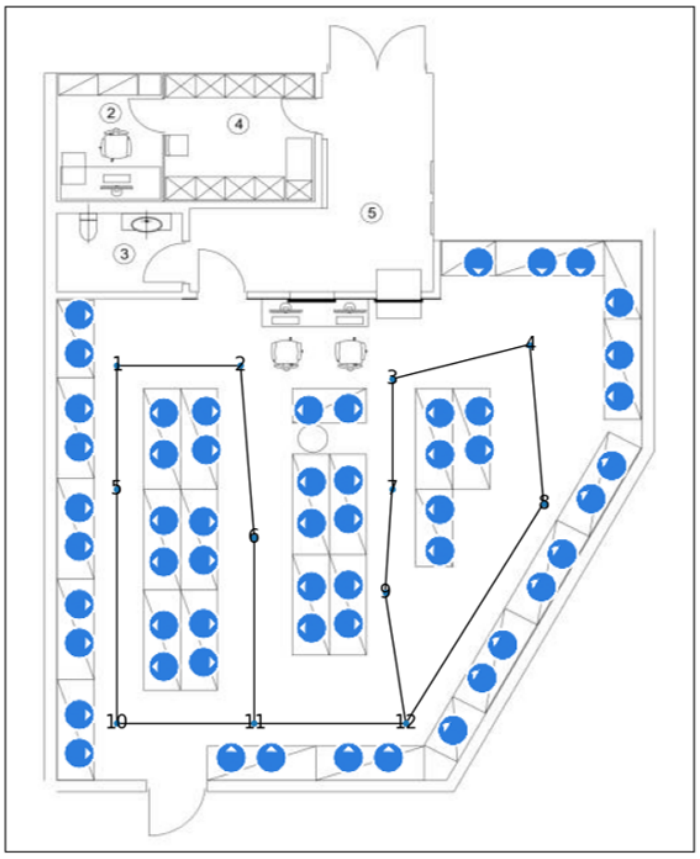


Figure 3.1: Example of a map where the larger blue circles represent stations, the smaller blue dots with numbers are nodes and the lines in between the nodes are the edges.

3.3 Online System Representation / Simulation

To be able to test and verify the developed system it is needed to perform some kind of simulation. Since the implementation of the online functionality is closely related to how the system is simulated the two subjects are discussed here.

Ideally, some kind of continuous time simulation would be optimal to get a good idea of how well the system works, where the entirety of the environment is represented in 3D with the vehicles employing the same trajectory planning and collision avoidance as they would in real life. However, setting up such a simulator would have proved too big of a task and outside the scope for this project, especially when taking into account the scalability aspect of the solution.

Instead, a discrete time-step based type of simulation was chosen to make it possible to develop a system that can handle incoming orders in an online setting. The vehicles move between nodes on the layout with discrete steps, never spending any time in between two nodes. This means that the focus is on inspecting the movement of vehicles that is strictly related to the congestion avoidance aspect of the solution, since only the time of access of an edge is taken into consideration when defining what constitutes an encounter between two vehicles.

3.4 Orders

In this section the proposed solution for modelling of orders will be discussed.

3.4.1 Task Modeling

The tasks are modeled as *pick up* and *drop off* operations, that have to be performed in sequence by the same agent since they carry a specific payload. In the system this is represented with an order that represents the *pick up* operation, which can be assigned to a vehicle. To represent the *drop off* operation a dummy order is generated which is assigned to the same vehicle that was assigned the previous order. The combination of these two orders is referred to as a complex order. In order to simulate the time it takes to execute the operation at the station, the vehicle remains at the goal node for a short period of time before carrying on with the next order.

3.4.2 Order Priority

When there are a lot of orders in the system and more orders are added there is a risk that some orders will never be handled. This issue can be addressed by introducing order priority. If each order has a priority which is increased each iteration that the order is not assigned the order will eventually have the highest priority and therefore be assigned to a vehicle.

By the use of priority the FMS can also make sure that an order that is critical can be pushed to the top of the order list when it is entered into the system.

3.4.3 Order Clustering

Scalability can be increased using clustering. This would divide the large problem into several smaller problems which could be easier and more time efficient to solve. However, this may also cause loss of information when computing a solution. More clusters mean that there will be less vehicles in the same optimization iteration at once. Vehicles that are part of a cluster that will be optimized first will not take the vehicles assigned to clusters that are optimized later into consideration.

By using the K-means algorithm [11] one can cluster orders based of their goal nodes' positions and then run an optimization program in order to assign vehicles to these order clusters optimally.

There will be several cases depending on the the current state of the online system. The optimization problem must be handled differently depending on whether there are more orders present than available vehicles, and vice versa.

3.5 Vehicle Routing

For smaller scale problems it is possible to optimize order allocation at the same time as path allocation. However, since this problem grows in complexity extremely fast the two parts are handled sequentially. Figure 3.2 shows an example of a desired result that avoids congestion in an example scenario.

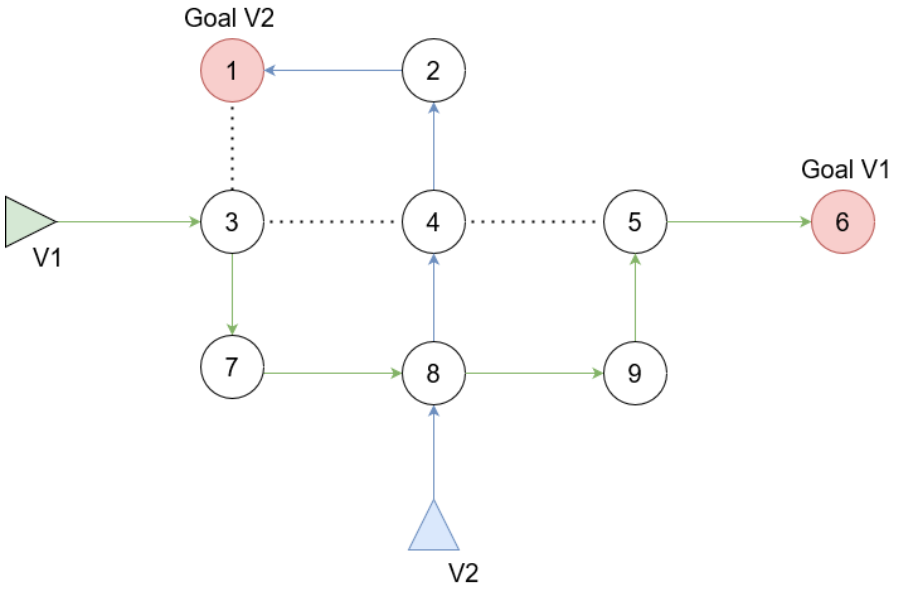


Figure 3.2: Figure showing the the desired pathing result for a specific scenario where two vehicles would be accessing the same location at the same time if they both were assigned the shortest path to their respective destinations. In this case vehicle 1 has been assigned a longer path to avoid the encounter at node 4.

3.5.1 Assigning Vehicles to Orders

Once orders have been clustered there are two optimization steps taken in order to assign orders to vehicles. The first optimization step assigns vehicles to the order clusters minimizing the total euclidean distance of the vehicles to the centroids of the clusters, then for each cluster another optimizer assigns orders to vehicles, minimizing the euclidean distance between the orders and the vehicles inside each cluster.

This means that for each loop iteration one new order is assigned to each available vehicle.

3.5.2 Pathing

Pathing is computed using Dijktra's algorithm [7]. More than one path alternative needs to be computed for each vehicle and order pair, since the pathing optimization step that follows requires alternatives to choose from in order to minimize its objective function. Once the first path has been computed, all weights of the edges in that path are increased, so that when Dijkstra's algorithm is applied again it will not return the same path, unless there are no other alternatives. Once the desired amount of path alternatives have been computed, the weights are reset.

In the context of vehicles solving complex orders it is necessary to compute the path from the vehicle to the pickup location and the path from the pickup location to the dropoff location. Since the system operates in an online context, the starting point is not the vehicles current position, but rather the final location of its current planned path. The paths are then appended to each other, meaning that the path optimizer will take into consideration the path of the vehicle from its planned destination, through the pickup point to its dropoff destination.

Since the path optimizer requires as previously mentioned alternatives, n alternatives are computed for both sub-paths. The paths are appended so that n^2 total alternative paths can be passed to the path optimizer.

3.5.3 Encounters

Encounters between AMRs can be of many types in a real situation. The vehicles have sensors and low-level trajectory planning capability in order to avoid collisions that can cause damage to the vehicle, and some degree of obstacles avoidance capability. However, they may have difficulty resolving more complex situations such as two vehicles trying to access the same corridor at the same time and vehicles trying to access a narrow corridor from two opposite sides, causing a deadlock. Ideally, any scenario in which an AMR has to employ low-level trajectory planning due to another AMR present nearby should be treated as an encounter.

For this reason, two types of encounters have been defined which are taken into account in the path optimizer.

- Regular Encounters: when vehicles are attempting to access an edge from the same direction. The optimizer checks if there are any accesses within a certain buffer time σ, and if the accesses are from the same direction it is considered a regular encounter.
- Critical Encounters: for vehicles trying to access a narrow edge from opposite sides. The optimizer checks if there are any accesses within a certain extended buffer time τ, and if the accesses are from the different direction it is considered a critical encounter. These are only accounted for on narrow edges, since on a normal edge it is to be expected that vehicles are able to meet oncoming traffic and utilize the space available to not cause a traffic jam.

3.5.4 Booking System

In order for the system to be able to know where vehicles are expected to be located in the layout, so that it can route new vehicles in such a way that they do not encounter vehicles traveling on paths previously assigned, a booking system will be used. The booking system will contain for each edge in the graph updated information about which vehicles will access it, when they will access it and in what direction they will access it.

4

System Architecture

In this chapter the architecture of the system developed is presented in detail, with explanation of how the various components work and communicate with each other.

4.1 Architecture Overview

The developed optimization loop is divided into three different sub-optimization modules. Using a structure featuring steps of sub-optimization means that loss of information will occur during the computation of the solution. Ideally, all information relevant to the problem would be taken into consideration in one single optimizer that outputs a complete solution for all vehicles. However, using a sequence of optimizers designed to each solve specific parts of the problem is extremely helpful for the purpose of maintaining scalability. In this context sub-optimization is referred to the optimization computation that is part of the optimizer as a whole, and not that it necessarily produces a sub-optimal solution.

The structure of the optimizer can be seen in fig 4.1. It features three different optimization steps, referred to in order as Vehicle to Cluster optimization, Vehicle to Order optimization, and Path Allocation optimization.

After clustering orders based on the location of the orders' goal nodes, the optimizer computes the Vehicle to Cluster optimization. From here, the software branches out and computes solutions for each cluster in sequence. After a cluster has been assigned a set of vehicles that are candidates for carrying out the orders in the cluster, the Vehicle to Order optimization is computed. At this point a set of path alternatives are computed between the vehicles and their assigned orders, which are then chosen in the Path Allocation optimization.

The main purpose of each sub-optimization would therefore be maintaining scalability for the Vehicle to Cluster optimizer, minimizing travel distance for the Vehicle to Order optimizer and minimizing the number of encounters for the Path Allocation optimizer.

An algorithm that demonstrates the structure of the code can be seen in Appendix A.

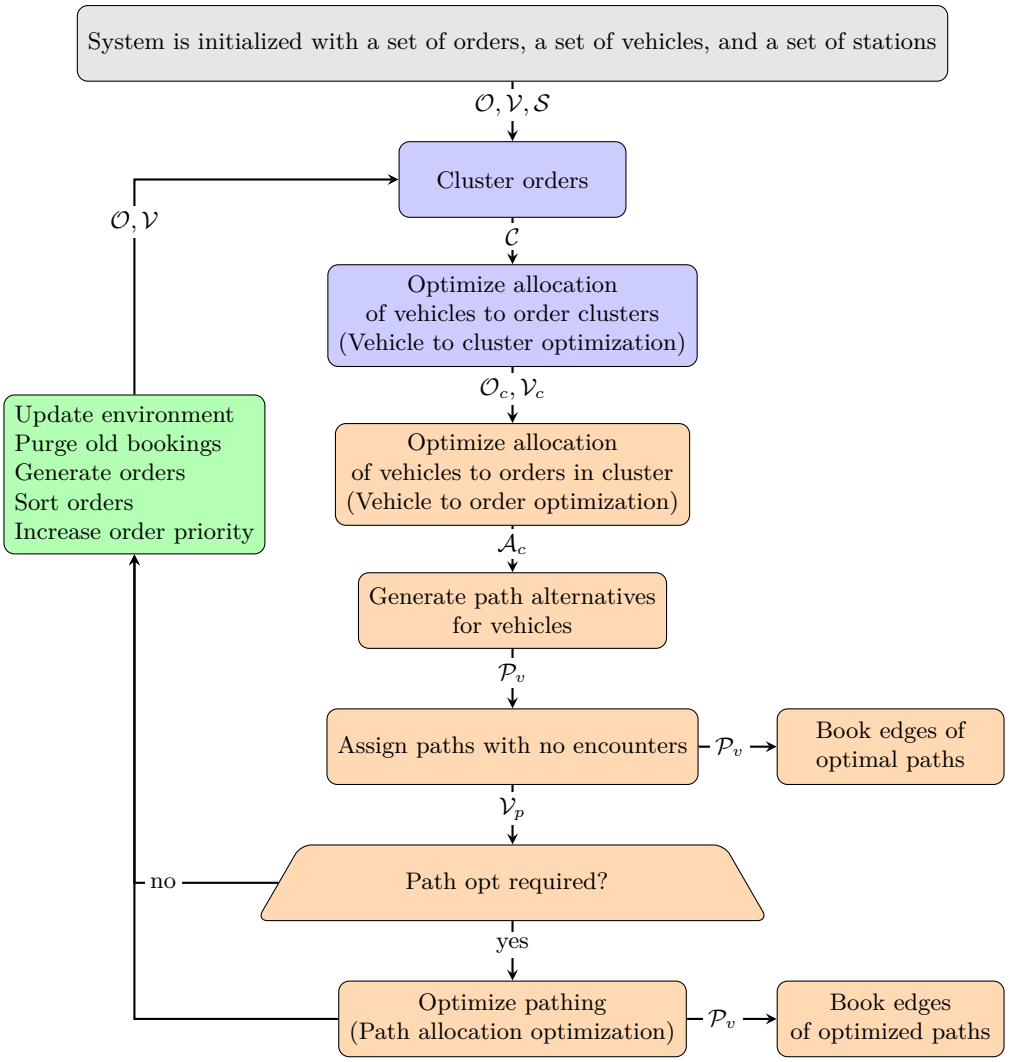


Figure 4.1: Flowchart describing the system architecture. Blocks in orange are operations performed in sequence for every cluster.

4.2 System Initialization

The system is initialized with a graph, a set of stations, a set of vehicles, and a set of orders. All vehicles are placed in some random node in the map at initialization. NetworkX [10] is used to initialize an undirected graph. NetworkX offers functionality for pathing, adding and removing nodes etc.

The system can be set to either start with a backlog of orders, or to generate a certain amount of orders every optimization iteration, or both. An order is a Python object assigned to a station that a vehicle needs to reach in order to complete the order, a time of initialization and a time of completion.

Stations are Python objects initialized at the beginning of the execution of the system and are assigned to a certain node. This is to simulate that the graph is an abstraction of a real layout, meaning that there could be several stations present at the same node.

Vehicles are Python objects that are initialized at the beginning of the execution of the system and can be removed or added during the online simulation. Each vehicle stores their planned path and the complete path that they have traveled during the simulation. They also keep an updated expected time of arrival (ETA).

4.3 Order Priority Handling

Each order has a priority assigned to them, the list of orders is sorted depending on this priority before the clustering is executed. The orders with the highest priority is then sent to the clustering step.

After each optimization cycle where orders are being assigned to vehicles each order's priority is incremented in order to ensure that all orders are eventually going to be fulfilled.

Orders that are added to the system during the simulation can have different priorities, if an order is considered to be critical the priority can be set to a high value while if an order is not very critical it can be set to a low value.

4.4 Online Simulation

When the optimization has been performed the environment is updated with new orders, vehicles can be removed or added in order to simulate that vehicles are taken out of rotation which could be due to low battery, repairs or similar. How the vehicle is routed to a depot or battery station is not handled by the current implementation.

If a vehicle is taken out of rotation its orders are returned to the order list with an increased priority to make sure that they will not be left unfulfilled for too long. Order priority is also increased for all orders every time they are not assigned

during optimization, otherwise an order with low priority could potentially never be assigned to a vehicle.

At the end of each time-step, the global time variable for the environment is updated as well as all of the edge bookings. The old bookings that are no longer relevant are purged.

The parameter t_o indicates how many time-steps need to pass before the system can run its optimization program, in order to simulate a real system that changes while the new solution is computed. This parameter could therefore be changed every optimization iteration, to simulate how longer and shorter computation times make a difference in how long the vehicles can travel or stay idle between optimization iterations.

4.5 Clustering - K-means

The clustering of orders is done using the Python package *scikit learn* [14]. It takes all of the positions of the orders' pickup location as input and clusters the orders using the K-means algorithm into k clusters according to the positions. The variable k is calculated based on an empirically determined parameter α, which is used for scaling the size of clusters. k is then calculated as:

$$k = ceil\left(\frac{size(\mathcal{V})}{\alpha}\right) \tag{4.1}$$

where the *ceil*-function is rounding the input to its nearest higher integer.

4.6 Vehicle to Cluster Optimization

When clustering is performed using the K-means algorithm vehicles must be assigned to each cluster. This is done by solving a optimization problem using a MILP solver, in this case Gurobi [9]. The objective function of the problem is to minimize the total distance between vehicles and centroids of the cluster, while still satisfying the constraint that each cluster should be assigned a vehicle. Additionally, if there are more orders than vehicles, then all vehicles should each be assigned to a cluster. If there are more vehicles than orders, at least as many vehicles as orders should be assigned. This means that there are two different problem cases.

To solve the problem the following definitions are made:

- $\mathcal{C}$ - Set of clusters.
- $\mathcal{O}$ - Set of orders.
- $\mathcal{V}$ - Set of available vehicles.

- d_{cv} - Euclidean distance between cluster c and vehicle v (this is calculated beforehand).
- y_{cv} - Binary decision variable that describes whether cluster c is assigned to vehicle v.
- δ_v - Estimated time of arrival of vehicle v.
- t - The current time-step of the simulation
- γ - Maximum number of orders per cluster

The following constraints are defined in the optimizer:

$$\text{minimize} \sum_{c \in \mathcal{C}} \sum_{v \in \mathcal{V}} y_{cv}(d_{cv} + \delta_v - t), \quad s.t. \tag{4.2}$$

$$\sum_{v \in \mathcal{V}} y_{cv} \geq 1, \quad \forall\ c \in \mathcal{C} \tag{4.3}$$

$$\sum_{v \in \mathcal{V}} y_{cv} \leq \gamma, \quad \forall\ c \in \mathcal{C} \tag{4.4}$$

$$\sum_{v \in \mathcal{V}} y_{cv} \leq size(\mathcal{C}), \quad \forall\ c \in \mathcal{C} \tag{4.5}$$

$$\sum_{c \in \mathcal{C}} y_{cv} \leq 1, \quad \forall\ v \in \mathcal{V} \tag{4.6}$$

The expression defined in (4.2) describes the objective function. Note that the ETA of the vehicles is added as a cost so that vehicles who are soon to become idle will be prioritized. The purpose is to minimize the total sum of all euclidean distances between assigned vehicles and cluster centroids. Constraint (4.3) makes sure that all clusters have at least one vehicle assigned to them. Constraint (4.4) assures that the maximum number of vehicles per cluster is not exceeded and constraint (4.5) assures that there are enough orders in a cluster to keep the assigned vehicles busy. Constraint (4.6) ensures that a maximum of one order is assigned to each vehicle.

Furthermore, depending on the amount of vehicles $\mathcal{V}$ and orders $\mathcal{O}$ present in the system additional constraints are defined according to the following two cases:

4.6.1 Case 1 - $size(\mathcal{V}) \geq size(\mathcal{O})$

$$\sum_{c \in \mathcal{C}} \sum_{v \in \mathcal{V}} y_{cv} \geq size(\mathcal{O}) \tag{4.7}$$

Constraint (4.7) is added to ensure enough vehicles are assigned to cover all orders.

4.6.2 Case 2 - $size(\mathcal{V}) < size(\mathcal{O})$

$$\sum_{c \in \mathcal{C}} y_{cv} \geq 1, \quad \forall\, v \in \mathcal{V} \tag{4.8}$$

Constraint (4.8) is added to ensure that all vehicles are assigned to clusters.

4.7 Vehicle to Order Optimization

After vehicles have been assigned to clusters the rest of the solution is computed for each cluster. The Vehicle to Order optimizer allocates each vehicle in a cluster to an order. This is done using Gurobi to solve a MILP-problem. In order to solve this problem the following definitions are provided:

- $\mathcal{O}_c$ - Set of orders in cluster c
- $\mathcal{V}_c$ - Set of vehicles in cluster c
- d_{ov} - Euclidean distance between order o and vehicle v (this is calculated beforehand).
- y_{ov} - Binary decision variable that describes whether order o is assigned to vehicle v.

The problem is then solved by solving the optimization problem which can be described by the following equations:

$$\text{minimize} \sum_{o \in \mathcal{O}_c} \sum_{v \in \mathcal{V}_c} y_{ov}(d_{ov} + \delta_v - t), \quad s.t. \tag{4.9}$$

$$\sum_{v \in \mathcal{V}_c} y_{ov} \leq 1, \quad \forall\, o \in \mathcal{O}_c \tag{4.10}$$

$$\sum_{o \in \mathcal{O}_c} y_{ov} \leq 1, \quad \forall\, v \in \mathcal{V}_c \tag{4.11}$$

The objective function (4.9) minimizes the distances between vehicles and orders, while also taking into consideration the ETA of the vehicles. Constraint (4.10) ensures that at most one order is assigned to each vehicle, while (4.11) ensures that at most one vehicle is assigned to each orders.

Depending on the number of vehicles $\mathcal{V}_c$ and orders $\mathcal{O}_c$ additional constraints are defined according to the following three different cases:

4.7.1 Case 1 - $size(\mathcal{O}_c) > size(\mathcal{V}_c)$

$$\sum_{o \in \mathcal{O}_c} y_{ov} \geq 1, \quad \forall\, v \in \mathcal{V}_c \tag{4.12}$$

In the scenario where there are more orders in a cluster than vehicles, the constraint (4.12) ensures that all vehicles are assigned to orders.

4.7.2 Case 2 - $size(\mathcal{O}_c) < size(\mathcal{V}_c)$

$$\sum_{v \in \mathcal{V}_c} y_{ov} \geq 1, \quad \forall\ o \in \mathcal{O}_c \tag{4.13}$$

In the scenario where there are more vehicles in a cluster than orders, the constraint (4.13) ensures that all orders are assigned to vehicles.

4.7.3 Case 3 - $size(\mathcal{O}_c) = size(\mathcal{V}_c)$

In the scenario that a cluster contains the same number of orders and vehicles, both constraints (4.12) and (4.13) are added to the model.

4.8 Congestion Avoidance

Once vehicles have been assigned orders, paths for each vehicle are computed using the Networkx pathing function. Alternative paths are also computed according to how it was described in section 3.5.2. In order to assign to the vehicles the paths that will cause the least overall congestion, it is necessary to detect the possible encounters that will be caused between vehicles if certain paths are assigned. The way in which possible encounters are detected in the system is the most important aspect in ensuring scalability of the solution. If the path allocation optimizer knows beforehand how many encounters will be caused by assigning a certain path it is significantly easier for it to find an optimal solution. The task was split up into detection of potential encounters between vehicles with preexisting bookings, and detection of potential encounters within the same optimization cycle. The following types of encounters are defined:

- β - Encounter on a regular edge caused by preexisting booking.
- B - Encounter on a narrow edge caused by preexisting booking.
- ω - Encounter on a regular edge caused by another vehicle in the same optimization cycle.
- Ω - Encounter on a narrow edge caused by another vehicle in the same optimization cycle.

4.9 Handling Encounter Free Paths

Since most of the vehicles will most times not require any optimization, if it is detected that they can choose their optimal path without causing any encounters they are allowed to do so and are excluded from the path optimization process. This is done by checking for each vehicle if the edge accesses in their shortest path causes an encounter with an already present booking or with the possible edge accesses of other vehicles $\mathcal{V}_c$. If no encounter is detected, the shortest path is assigned to the vehicle in question and the edges of that path are booked according to the time of

access and direction of access for each edge in the path. This reduces the complexity of the optimization that follows.

4.10 Path Allocation Optimization

The goal of the path allocation optimizer is to assign vehicles the shortest paths that will cause the least amount of encounters possible. The problem was modeled with SMT and Z3 [6] was used to build the model and compute the solution. Note that this optimizer does not instantiate variables for each timestep it takes into consideration in order to detect encounters. The objective function in the optimizer weighs the benefit of choosing a short path with more encounters or a longer path with less encounters. This means that it is possible to later on tune the cost function according to the ability of the AMRs to solve encounters, since it is not currently known what exactly constitutes a scenario that will cause a deadlock.

The Path Allocation optimizer is designed so that it will always compute a solution. This is because from the delivered solution it is always possible to know when, where and which vehicles are trying to access the same location simultaneously, so that if the solution is deemed to have too many encounters the vehicles that are considered to have an excessive amount of encounters can be rerouted by a separate function. The reason behind this is that the solution should not be discarded entirely due to the optimizer not being able to assign a route to one of the vehicles.

The following sets and variables are defined to solve the problem:

- e - Edge that can be traversed in both directions without causing a critical encounter.
- n - Narrow edge that cannot be traversed in both directions without causing a critical encounter.
- p - A path, consisting of edges e or narrow edges n or both.
- $\mathcal{V}_p$ - The set of vehicles that require path allocation optimization.
- $\mathcal{P}_v$ - The set of paths generated for each vehicle.
- $\mathcal{E}_v$ - The set of edges used by a vehicle.
- $\mathcal{N}_v$ - The set of narrow edges used by a vehicle.
- w_{vp} - Weight of path p for vehicle v.
- κ - Constant for weighing cost of encounter.
- λ - Constant for weighing cost of critical encounter.

The optimizer instantiates the following decision variables:

- z_{vp}: Boolean for each vehicle and each path it might take. This is the main

decision variable, and for each vehicle a boolean determining the one that it should take will be set to true while the rest are set to false.

- ω_{ve}: An integer for each vehicle and each edge it might traverse, counting the number of ω that the vehicle will encounter while using that edge.
- Ω_{vn}: An integer for each vehicle and each narrow edge it might traverse, counting the number of Ω that the vehicle will encounter while using that edge.
- π_{vp}: An integer for each vehicle and each path it might take. Counts the number of regular encounters (both β and ω) the vehicle will cause when assigned a certain path. This variable is not strictly necessary but is used in order to keep the model easier to follow.
- Π_{vp}: An integer for each vehicle and each path it might take. Counts the number of critical encounters (both B and Ω the vehicle will cause when assigned a certain path. This variable is not strictly necessary but is used in order to keep the model easier to follow.

The following model is generated in the Path Allocation optimizer:

$$\text{minimize} \sum_{v \in \mathcal{V}_c} \text{If } (z_{vp},\ w_{vp} + \kappa\pi_{vp} + \lambda\Pi_{vp},\ 0), \quad \forall\ p \in \mathcal{P}_v \tag{4.14}$$

$$\sum_{p \in P} \text{If}(z_{vp},\ 1,\ 0) = 1, \quad \forall\ v \in \mathcal{V}_p \tag{4.15}$$

$$\omega_{ve} \geq 0, \quad \forall\ e \in \mathcal{E}_v, \quad \forall\ v \in \mathcal{V}_p \tag{4.16}$$

$$\omega_{ve} \leq 1, \quad \forall\ e \in \mathcal{E}_v, \quad \forall\ v \in \mathcal{V}_p \tag{4.17}$$

$$\Omega_{vn} \geq 0, \quad \forall\ n \in \mathcal{N}_v, \quad \forall\ v \in \mathcal{V}_p \tag{4.18}$$

$$\Omega_{vn} \leq 1, \quad \forall\ n \in \mathcal{N}_v, \quad \forall\ v \in \mathcal{V}_p \tag{4.19}$$

$$\pi_{vp} = \beta_{vp} + \sum_{e \in p} \omega_{ve}, \quad \forall\ p \in \mathcal{P}_v, \quad \forall\ v \in \mathcal{V}_p \tag{4.20}$$

$$\Pi_{vp} = \text{B}_{vp} + \sum_{n \in p} \Omega_{vn}, \quad \forall\ p \in \mathcal{P}_v, \quad \forall\ v \in \mathcal{V}_p \tag{4.21}$$

$$\text{If } (z_{v_1p_1} \wedge z_{v_2p_2},\ 1,\ 0) \geq \omega_{v_1e}, \quad \forall\ e \in p_1 \tag{4.22}$$

$$\text{If } (z_{v_1 p_1} \wedge z_{v_2 p_2},\ 1,\ 0) \geq \Omega_{v_1 e}, \quad \forall\ e \in p_1 \tag{4.23}$$

The objective function (4.14) to minimize is modeled as a cost function and uses the parameters κ to weigh the cost of a regular encounter and λ to weigh the cost of a critical encounter, which is always set to a very large value. The reasoning is that while critical encounters should always be avoided, if the path optimizers decides to assign a path causing a critical encounter it can always be rerouted by a separate function later. If one were to set this as a hard constraint then the optimizer would sometimes not be able to deliver a solution, which would mean that none of the vehicles in $\mathcal{V}_c$ would be assigned a solution.

The constraint (4.15) ensures that vehicle is assigned exactly one path. The constraints (4.16), (4.17), (4.18), and (4.19) set the bounds for the ω_{ve} and Ω_{vn} decision variables.

Constraints (4.20) and (4.21) are constraints that are strictly not necessary, but are used to sum up the amount of regular and critical encounters caused if each vehicle v were to be assigned the path p. Without these constraints the objective function would have been more complicated to define.

The purpose of constraints (4.22) and (4.23) is to count the encounters caused by assigning paths to two vehicles that are being processed in the same path optimization iteration. The constraints themselves are fairly simple, however they are only generated when choosing a path p_1 for a certain vehicle v_1 and a path p_2 for a certain vehicle v_2. Therefore these are only generated once it has been asserted that these two assignment will cause a regular encounter or a critical encounter.

5

Evaluation

In this chapter the test case used to test the system is presented, along with the results and comments on the results.

5.1 Test Definition

The layout on which the system was tested represents a warehouse, in which products arrive from a source location and have to be brought to a storage area, and products inside the storage area have to be brought to a delivery area. This layout is generated in order to simulate layouts of different scale. The parameters in the generation that can be chosen are the number of rows, the number of groups of stations and the number of parallel aisles at the source and delivery area. The parallel lines are added so that the pathing algorithm is able to compute path alternatives so that the vehicles can avoid encounters and therefore reduce congestion. Three different layouts were generated in different sizes, as can be seen described in the table in Appendix B.1. This type of layout was chosen because it represents a realistic (although simplified) scenario of a warehouse layout where products need to be moved to different locations, with clear features representing chokepoints (entrance to the rows) in which congestion could cause issues. The orders were generated with random priority and the vehicles are initialized at random nodes in the layout.

The orders are all generated in one batch at the beginning of the simulation, even though the system is able to take in new orders as the simulation progresses. This is in order to ensure that the same set of orders is being used to compare the solution of the developed algorithm and the benchmark, which is defined further down. The simulation ends once all orders have been assigned. A screenshot of the simulation can be seen in the figure in Appendix C.1. Parameters that were set in common for all test scenarios can be seen in the table in Appendix B.2.

The computer on which the system was tested uses an Intel i7-8665U CPU @ 1.90 GHz with 32 GB of RAM.

5.2 Benchmark Definition

In order to understand the quality of the solution given by the optimization algorithm, each simulation was repeated with a basic algorithm that would assign orders to closest vehicles on a first come first serve basis, and from there directly assign the shortest path computed with Dijkstra's algorithm between each vehicle and order. This benchmark is denoted as FCFS (First Come, First Serve).

5.3 Results

The results of the 12 tests can be seen in tables 5.1, 5.2, and 5.3. Six benchmark solutions (FCFS) were computed in order to compare the result. Each test and benchmark was repeated ten times, of which the results were averaged, for a total of 180 simulations. The results computed for each test were:

- Congestion - The relative change in total encounters caused by optimized solution compared to the benchmark. The number of encounters was computed by a verification function executed at the end of the simulation. The term congestion here means that there are two or more vehicles that will access the same edge within a certain time frame. This time frame is what is considered to be the upper limit in time to where an encounter between vehicles would occur.
- Travel Distance - The relative change in total distance traveled by all the vehicles in the system in the optimized solution compared to the benchmark.
- Time of Completion - The relative change in the final time of completion of the optimized solution compared to the benchmark. The final time of completion is defined as the earliest time instance when all orders have been carried out.
- Average Computation Time - The average amount of time required to compute one iteration of the optimization cycle.

Layout Type	Small			
Number of Vehicles	5		10	
Conflict Cost κ	100	10000	100	10000
Congestion	−83.77%	−83.33%	−79.10%	−79.08%
Travel Distance	+2.70%	+2.63%	+6.47%	+6.51%
Time of Completion	+2.70%	+2.64%	+3.91%	+4.34%
Average Computation Time	0.07 s	0.07 s	0.19 s	0.19 s

Table 5.1: Small Layout, featuring 55 nodes.

Layout Type	Medium			
Number of Vehicles	25		50	
Conflict Cost κ	100	10000	100	10000
Congestion	−94.50%	−94.90%	−92.30%	−92.78%
Travel Distance	+0.59%	+0.64%	+2.84%	+2.83%
Time of Completion	+3.45%	+3.57%	−0.38%	−1.05%
Average Computation Time	0.84 s	0.87 s	3.45 s	3.77 s

Table 5.2: Medium Layout, featuring 390 nodes.

Layout Type	Large			
Number of Vehicles	150		300	
Conflict Cost κ	100	10000	100	10000
Congestion	−94.79%	−94.95%	−91.10%	−91.39%
Travel Distance	+2.51%	+2.69%	+5.15%	+5.49%
Time of Completion	+0.22%	+0.27%	+1.12%	+1.69%
Average Computation Time	15.06 s	16.43 s	48.90 s	44.32 s

Table 5.3: Large Layout, featuring 1840 nodes.

5.4 Comments

For almost all test cases a reduction of an order of magnitude in congestion can be observed. While this seems like a successful result, it is not conclusive in determining how well the optimized solution actually avoids congestion without physical testing.

The travel distance is slightly increased for all test cases, which can be expected given the test scenario. Since the orders are composed of a pick up and drop off task, the only distance that the optimizer can reduce is the distance between vehicles and pickup location. This distance is generally shorter than the distance betweeen pickup location and dropoff location, since the payload is to be moved between source and storage stations or storage and delivery stations. This is shorter compared to the distance between a vehicle and the closest pickup point of an order. When it comes to distance between pickup and dropoff location, the benchmark will always choose the shortest path, while the optimizer will sometimes assign vehicles a longer path in order to reduce congestion. Still, the increase in travel distance in the optimized solution is small enough to be deemed acceptable since if the vehicles do not encounter other vehicles they do not have to brake and accelerate as often, which reduces wear and tear of the vehicles, and require less power to reach their destination. This is however speculation and requires further study to confirm.

Each of the test cases was computed with a low and high conflict cost parameter κ. The reason behind the parameter κ was so that the optimizer could be tuned to either generate a solution with either shorter travel distance and more congestion or vice versa, since the actual capability of AMRs to resolve encounters between vehicles is

not well understood. In fact, the results do not seem to differ significantly between the instances in which a low and high κ value was used. The reason behind this might be because the various path alternatives computed for each vehicle do not offer enough malleability in cost to generate different results.

Any congestion left in the solution, that perhaps can not be avoided, is not handled by this system. It is rather considered to be handled outside of the system, in the FMS. It might be the case that it is only a few vehicles in a large system that are causing congestion. In that scenario the FMS could take these vehicles out of the system and return their assigned orders to the system with an increased priority. The vehicles could then possibly be reentered into the system and have new orders assigned.

The computation times show that the optimizer is fast enough to always be able to compute a solution in time, so that in none of the cases vehicles should remain idle due to the optimization not being complete. This is true even for the biggest test case with 300 vehicles, were solutions were computed in on average 48.90 seconds and 44.32 seconds, which in a big warehouse should be far lower than the average time to complete a task.

Overall, the results from the simulations seem to point toward the fact that the optimizer manages to avoid congestion at a limited increase in travel distance, which in realistic scenario is a profitable tradeoff, since resolving issues with vehicles not knowing how to reach their destination can cause huge disruptions in the system as a whole and require human operators.

6

Discussion

In the previous chapter the results indicated that the proposed solution could potentially reduce congestion substantially in a real multi-agent system. In this chapter the quality of the proposed solution will be discussed with respect to the research questions and the problem definition.

6.1 Addressing the Research Questions

How does one compute an efficient solution that maximizes order throughput within a time frame that allows the system to run smoothly while also scaling well with larger environments, fleets and order lists?
This question delves into the issue of consistently generating a good solution even in larger environments. It can be noted that the question focuses on maximizing order throughput, a metric in which the solution proposed in this thesis does not immediately improve upon, compared to the basic solution. However, it can be argued that the reduction in congestion that the solution achieves should improve order throughput if disruption caused by congestion is avoided. The question also emphasizes the need for achieving a smooth execution of the system, which can be directly related to congestion avoidance. A system where vehicles travel often at maximum speed and do not need to break for incoming traffic appears efficient, which is a desirable quality in a product for a company that sells autonomous agents. The empirical results also point to the fact that the proposed solution manages to consistently deliver optimized solutions in a timely manner even for large number of vehicles.

How do different specifications in form of number of vehicles, orders and stations affect the performance of the system? Which of these variables will be the biggest issue in regards to the complexity of the problem?
This research question focuses on identifying the factor that limits the scalability of the solution the most. This is difficult to answer since all factors contribute to increased complexity in the optimization problem. However, it can safely be said that how one decides to model the environment has a massive impact on complexity. The proposed solution simplifies the layout in which the vehicles operate to a possibly excessive degree, which is the main reason the optimizer can deliver solutions to large scale systems in real time. There are related works that use time discretized

decision variables, which generate decision variables for each area each vehicle might be located in at a future timepoint, such as the CF-EVRP presented by Roselli et al [16]. Such systems deliver high quality solutions, however they generate very complex models. Increasingly detailed environments also mean each path computed for each vehicle is going to be more complex and take more processing power to compute and evaluate. Other specifications such as number of vehicles and orders are not really driving factors for increased complexity in the proposed solution, since it uses clustering to divide the problem into smaller optimization problems. The loss of information caused by clustering can affect the quality of the solution if the scale of the problem to solve is too large. However, when the main goal is providing a schedule for soft real-time applications, it is necessary to trade optimality for computation efficiency.

6.2 Reflecting on the Problem Definition

The problem defined in this thesis has a wide scope, and therefore it is to be expected that addressing all issues properly would be difficult. Based on the research that was done in the early stages of the project, it was apparent that researchers would often focus on solving one aspect of the problem in the optimal way, while often disregarding other important aspects. As the task was assigned by a company that develops autonomous systems, they are interested in a solution that can be implemented in a real scenario. Therefore they are willing to compromise on certain aspects in favor of a more versatile implementation.

While the proposed solution is able to take into consideration many of the desired aspects of a final product such as running in an online setting, adding and removing vehicles and prioritizing orders, it is difficult to incorporate these aspects into tests whose results can be presented in an academic report. Creating a convoluted test scenario that utilizes these features would be difficult to present to a reader, and most of all hard to compare to a benchmark, since the scenario becomes niche. For this reason, several features which are presented in Chapter 3 and implemented in the developed software are not properly tested in the evaluation of the system, which makes it difficult to answer whether the problems defined in Chapter 2 are properly addressed.

Nevertheless, the proposed solution does present an answer with reasoning to how the system can handle routing vehicles in large environments for a smoother result in the completion of the orders. The orders are somewhat simple since they only model transportation tasks, however they are not trivial either since it is fair to assume that in a large warehouse the most common task would be transporting products. Through order priority sorting and management, it is reasonable to expect that certain orders will not be left unattended for too long. The online representation enables the system to take in new orders and assign them continuously with the possibility of adding and removing vehicles that need to charge their batteries from the system.

However, there is the risk that, in the pursuit of keeping the system lightweight

so that it can operate on large scale scenarios, the environment may have been excessively simplified, and too much of the trajectory planning of the vehicle may have been left up to the lower level steering module of the AMR to solve. Since the environment is represented as a sparse graph there is little information on where vehicles are located exactly and at what speed they are traveling at at any given point in time, which means that many scenarios of interest in the CAVRP may not have been detected.

6.3 Further Research and Development

There are several ways in which the proposed solution could be modified or expanded upon. The evaluation shows that the system is able to greatly reduce congestion, however it is not able to completely avoid it. This is because vehicles have a limited amount of path alternatives they can be assigned to, and if all of them give rise to encounter with other vehicles, the optimizer has no alternative for avoiding encounters altogether. The Path Allocation optimizer simply assigns paths to vehicles and expects vehicles to travel along those paths without pausing. This could potentially be solved by introducing the possibility of assigning dynamic velocities to vehicles during optimization. This was considered in the design process of the project, but was discarded as it is complicated to design an optimizer which assigns stopping times to vehicles in such a way that it is apparent why the vehicle is stopping. In most cases it does not matter where a vehicle stops to wait for clearance along its path, however for a client interested in purchasing an autonomous system the product might seem flawed if vehicles are idling for seemingly no reason. While solving this issue is important, it was decided that spending too much time solving this problem meant trying to make a product more marketable rather than solving an issue with academic interest. Still, adding this functionality would open the possibility for a system that is able to completely avoid encounters between vehicles, and therefore be much less prone to causing deadlock situations. This would be critical for environments where many vehicles share a small space.

Another aspect that is worth investigating further is the modeling of different areas of the environment. The current solution models areas where crowding is expected to occur such as choke points and intersection with the same amount of nodes as an aisle which vehicles rarely make use of. Modeling areas with different level of detail could mean achieving better solutions while maintaining scalability.

Computing path alternatives that are useful for the Path Allocation optimizer is something that could be vastly improved upon. How to compute the shortest path between two points in a graph has been studied at length, however computing additional paths that are still short but also different in a way that is useful for the optimizer to deliver practical solutions is a more complex issue, and worth researching further.

6.4 Ethics and Sustainability

Optimizing routing of autonomous vehicles means achieving better results with altering the specifications, which is directly linked to the aspect of developing sustainable technologies. More specifically, it can be said that having vehicles traveling along routes uninterrupted, means less need of braking and accelerating, which reduces wear and tear in the vehicles, and leads to vehicles needing to be replaced less often. A system which manages to resolve orders without ending up in congested situation that require manual resets will experience an increased throughput of orders over time. Furthermore, improving the performance of autonomous systems means that more companies will consider employing them in their warehouses, to solve tasks that are more basic and therefore would be considered tedious and repetitive for a human to resolve.

7

Conclusion

The solution of the project presented in this thesis is a multi-step optimizer that can allocate AMR vehicles in a multi-agent setting to fulfill orders in an online environment and optimize their pathing so that congestion is avoided. The solution contains several features aimed to address issues that are commonly discussed in the industry, with focus on creating a final product that could potentially be used in a realistic scenario, rather than studying in depth only one aspect of the problem. The solution was tested through simulations and the quality of the solution analyzed based on both the obtained numerical values and on how the system is believed to behave in a realistic scenario.

Optimizing routing of multi-agent systems is an area of study that is growing as autonomous solutions become more ubiquitous. General purpose solvers are being studied and used for solving this type of optimization problems in academic environments, however published research often features precise scopes and finding optimal solutions to specific problem instances, rather than understanding how to develop a system that is versatile and can handle more than one aspect of the problem. On the other hand, a company that develops autonomous systems is more interested in a general satisfactory performance rather than optimal solutions in specific contexts. However, due to market competition, private companies seldom publish detailed explanations of how their systems work, which is the main reason why the project presented in this thesis is believed to be of academic value.

The authors hope that the proposed solution has been analyzed properly both in terms of how it operates and how its solution compare to a real scenario, so that the reader can form an educated opinion on what aspects are good to keep in mind and which mistakes to avoid, should they consider developing something similar of their own.

Bibliography

[1] Automated guided vehicle market size, share & trends analysis report by vehicle type, by navigation technology, by application, by end use industry, by component, by battery type, and segment forecasts, 2022 - 2030, 2022.

[2] Clark Barrett and Cesare Tinelli. *Satisfiability Modulo Theories*, pages 305–343. Springer International Publishing, Cham, 2018.

[3] G. Clarke and J. W. Wright. *Scheduling of Vehicles from a Central Depot to a Number of Delivery Points*, volume 12(4), pages 568–581. 1964.

[4] G. B. Dantzig and J. H. Ramser. *The Truck Dispatching Problem*, volume 6(1), pages 80–91. 1959.

[5] George B. Dantzig. *Linear programming and extensions.* Rand Corporation Research Study. Princeton Univ. Press, Princeton, NJ, 1963.

[6] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[7] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[8] Brian P. Gerkey and Maja J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[9] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.

[10] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[11] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *JSTOR: Applied Statistics*, 28(1):100–108, 1979.

[12] Gerhard Hiermann, Jakob Puchinger, Stefan Ropke, and Richard F. Hartl. The electric fleet size and mix vehicle routing problem with time windows and

recharging stations. *European Journal of Operational Research*, 252(3):995–1018, 2016.

[13] Marko Pedan, Milan Gregor, and Dariusz Plinta. Implementation of automated guided vehicle system in healthcare facility. *Procedia Engineering*, 192:665–670, 2017. 12th international scientific conference of young scientists on sustainable, modern and safe transport.

[14] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[15] Sabino Francesco Roselli, Kristofer Bengtsson, and Knut Åkesson. Smt solvers for job-shop scheduling problems: Models comparison and performance evaluation. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 547–552, 2018.

[16] Sabino Francesco Roselli, Per-Lage Götvall, Martin Fabian, and Knut Åkesson. A compositional algorithm for the conflict-free electric vehicle routing problem, 2022.

[17] Norberto Sousa, Nuno Oliveira, and Isabel Praça. A multi-agent system for autonomous mobile robot coordination. 2021.

[18] P. Toth and D. Vigo. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. MOS-SIAM Series on Optimization. SIAM, 2014.

[19] Günter Ullrich. *Modern Areas of Application*, pages 15–96. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[20] Z. Wang, S. Huang, and G. Dissanayake. *Simultaneous localization and mapping : Exactly sparse information filters : exactly sparse information filters*. World Scientific Publishing Company, 2011.

[21] Robert Zlot, Anthony Stentz, Chair Dias, Manuela Veloso, and Tucker Balch. *An Auction-Based Approach to Complex Task Allocation for Multirobot Teams Thesis Committee*. PhD thesis, 12 2006.

A

Appendix 1

Algorithm 1 Overview of System

```
Initialize system
for i in iterations do
    Sort orders based of priority
    Extract those orders with highest priority
    Compute clusters of orders
    Allocate vehicles to order clusters
    for c in 𝒞 do
        Euclidean optimization
        Calculate paths between orders and vehicles
        for v in 𝒱_c do
            if shortest path causes no encounters then
                Assign encounter-free path
                Book edges in encounter-free path
            end if
        end for
        Path allocation optimization
        for v in 𝒱_p do
            Assign paths to vehicles
            Book edges in path
        end for
    end for
    Environment update / Increment priority
    Purge old bookings
    Generate new orders and add to 𝒪
end for
```

B

Appendix 2

Layout	Small	Medium	Large
Rows	5	30	80
Groups per row	3	5	15
Parallels at ends	4	4	4
No. of nodes	55	390	1840

Table B.1: Size of the three different layouts that were used in the evaluation stage.

Parameter	Value	Meaning
Optimization Cycles	50	The number of times the optimization sequence was run
Paths per Vehicle	4	The number of path alternatives computed for each vehicle
Regular Buffer ϕ	5	The period of time in which if two vehicles were to access an edge in the same direction it would count as a regular encounter
Critical Buffer Φ	250	The period of time in which if two vehicles were to access a narrow edge in the opposite direction it would count as a critical encounter
Critical Encounter Cost λ	1000000	The cost in the Path Allocation Optimizer for generating a critical encounter
Optimization Interval	1	The number of timesteps simulated between each optimization computation
Cluster Size Scaling α	30	Factor used for scaling the size of the clusters
Timesteps	50	Length of simulation

Table B.2: The parameters used in the evaluation that were common for all test cases.

C

Appendix 3

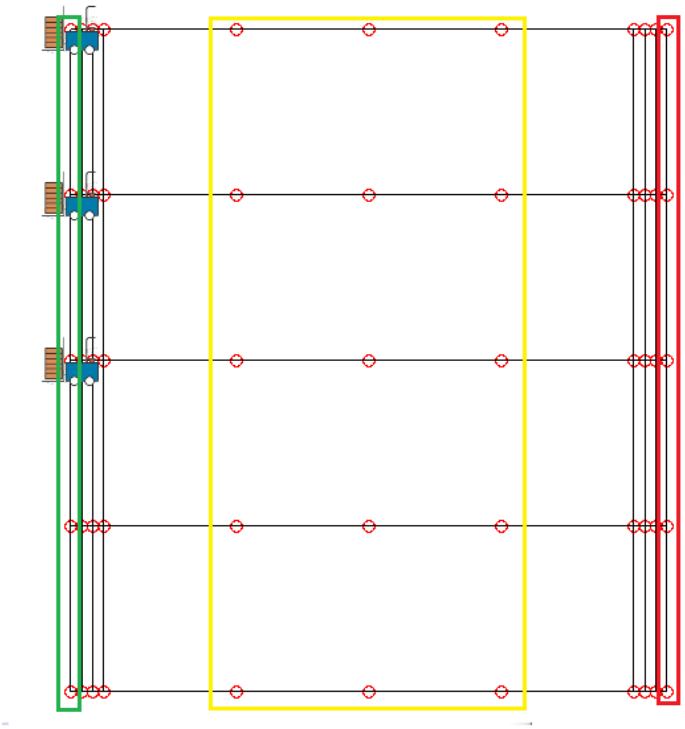


Figure C.1: Figure showing the simulation of a small layout. The nodes inside the green outline represent the source stations, the nodes inside the yellow outline represent the storage stations, and the nodes inside the red outline represent the delivery stations.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY