Karlsruhe Institute of Technology

Algorithm Comparison for Autonomous Mobile Robots Scheduling

Master Thesis of

Jakob Wagenblatt

At the Department of Informatics
Institute of Theoretical Informatics

Reviewers:	T.T.-Prof. Dr. Thomas Bläsius
	Prof. Dr. Stefan Nickel
Advisors:	Dr. Katharina Glock
	Adrian Feilhauer

Time Period: January 11 2022 – August 01 2022

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, August 1, 2022

Abstract

With the advent of Industry 4.0 and increasing automation, the importance of autonomous mobile robots (AMR) for transporting goods in intralogistics is growing. Consequently, this has given rise to the development of new centralized and decentralized methods for scheduling and routing these robots. However, there is a lack of comparison between the different algorithms, which moreover often do not model important features of the real world. Therefore, we develop a discrete-event simulation framework allowing easy comparison of different scheduling algorithms for the Electric Multi-Agent Pickup and Delivery (EMAPD) problem by finding a general basis for modeling instances. The EMAPD is an online problem in which delivery tasks arrive continuously and have to be assigned to AMRs. Its solutions consists of collision-free routes that fulfill these tasks while considering battery constraints, charging as necessary and minimizing tardiness. We evaluate the performance of seven different schedulers experimentally on ten different scenarios varying in layout, size, number of robots and system load. Our experiments show that the performance of the schedulers varies wildly, depending on the scenario and especially the system load. We attribute this in part to the charging strategy used as a simple rule-based scheduler implemented for reference with a different charging strategy sometimes performs best. Overall, we rate an auction-based scheduler punishing tardiness best. On average, it produces good results and performs efficiently.

Zusammenfassung

Mit dem Aufkommen von Industrie 4.0 und der zunehmenden Automatisierung wächst die Bedeutung von autonomen mobilen Robotern (AMR) für den Transport von Gütern in der Intralogistik. Dies hat zur Entwicklung neuer zentraler und dezentraler Methoden zur Lösung des Planungsproblem und der Routenplanung für diese Roboter geführt. Es mangelt jedoch an einem Vergleich zwischen den verschiedenen Algorithmen, die zudem oft wichtige Eigenschaften der realen Welt nicht modellieren. Daher entwickeln wir eine ereignisdiskrete Simulation, die einen einfachen Vergleich verschiedener Planungsalgorithmen für das Electric Multi-Agent Pickup and Delivery (EMAPD) ermöglicht, indem wir eine allgemeine Grundlage für die Modellierung von Instanzen finden. Das EMAPD-Problem ist ein Online-Problem, bei dem Lieferaufträge kontinuierlich eintreffen und den AMRs zugewiesen werden müssen. Die Lösungen bestehen aus kollisionsfreien Routen, die diese Aufträge unter Berücksichtigung des Batterie Ladezustands erfüllen, bei Bedarf aufladen und die Verspätung minimieren. Wir bewerten die Leistung von sieben verschiedenen Planungsalgorithmen experimentell in zehn verschiedenen Szenarien, die sich in Layout, Größe, Anzahl der Roboter und Systemlast unterscheiden. Unsere Experimente zeigen, dass die Leistung der Planner je nach Szenario und vor allem je nach Systemauslastung stark schwankt. Wir führen dies zum Teil auf die verwendete Ladestrategie zurück, da ein einfacher regelbasierter Planungsalgorithmus, der als Referenz mit einer anderen Ladestrategie implementiert wurde, manchmal am besten abschneidet. Insgesamt bewerten wir einen auktionsbasierten Planungsalgorithmus, der Verspätung bestraft, am besten. Er liefert im Durchschnitt gute Ergebnisse bei kurzen Laufzeiten.

Contents

1. Introduction

The increasing importance of automation in intralogistics in undeniable. This development can be observed in all areas of industry, whether it is an almost fully autonomous Tesla Gigafactory, an Amazon warehouse relying heavily on autonomous delivery robots, or UPS using it to transport packages efficiently in their mail centers. The speed of this development has been increased by the COVID-19 pandemic. Due to social distancing rules, quarantines and lockdowns, company owners are further incentivized to automate their workflows. Fortune Business Insights projects the market size of autonomous mobile robots (AMRs) to grow from USD 1.97 billion in 2021 to USD 8.70 billion by 2028 [1].

This development has given rise to new centralized and decentralized methods for scheduling, routing and charging AMRs. The scheduling problem of AMRs is a variant of the Vehicle Routing Problem (VRP) with additional constraints such as having to consider the battery capacity and avoiding collisions. It is an optimization problem where, given a number of tasks and a fleet of robots, the task assignment is to be optimized by for example minimizing travel time or distance travelled. In centralized algorithms, AMRs are controlled by a central unit, while in decentralized algorithms they make their own decisions, based on information exchanged with other robots. The former has the advantage of having all the information available and being able to optimize the allocation globally, while the latter can come to – potentially worse – decisions more quickly [2]. In addition, decentralized systems are more scalable and better able to deal with dynamic situations [3]. However, the comparison of different approaches – especially of centralized and decentralized methods – is lacking [4, 5]. Furthermore, the nomenclature surrounding this subject is ambiguous, as e.g AMR and automated guided vehicle (AGV) are used synonymously, even though the former is the evolution of the latter [6, 2, 7]. AGVs rely on magnetic guide tapes for navigation, while AMRs can navigate freely, using sensors.

Therefore, we use a model allowing easy comparison of different scheduling algorithms and compare different centralized and decentralized approaches and algorithms from literature by implementing them and analyzing their performance. To achieve this goal, we use several metrics such as runtime, number of concurrently active tasks or tardiness. The algorithms range from simple rule-based systems to sophisticated heuristics. They are implemented using the strategy design pattern, allowing them to be easily exchanged. We develop our own pathfinding and collision avoidance algorithm that we use for all task assignment algorithms by combining a complete state-of-the-art algorithm with a heuristic.

The schedulers are evaluated using a discrete-event simulation. It simulates the robot locations, movement and status and provides this information as well as currently open tasks to the planning framework. We use five fundamentally different layouts often found in real-world applications in order to gauge the performance of the schedulers in different environments. We also evaluate the impact of changing the settings by varying the workload and the number of robots for a given layout. We use two different algorithms to create a graph from a given layout.

Outline

In Chapter 2, we formally define the studied problem and give a brief literature overview. Furthermore, we present two algorithms to transform a warehouse layout into a graph data structure. In Chapter 3, we introduce the schedulers we evaluate. We start by describing the two different charging strategies, then present the centralized and decentralized task assignment algorithms and finally describe the pathfinding and collision avoidance algorithm. In Chapter 4, we present the discrete-event simulation framework used to carry out the evaluation. In Chapter 5, we present the experimental setup and evaluate the performance of the schedulers qualitatively and quantitatively and discuss the results. Finally, we summarize our results in Chapter 6 and provide an outlook for possible future research.

2. Modeling

In this chapter we describe the model used in this thesis. In Section 2.1 we formally define the studied *Electric Multi-Agent Pickup and Delivery* problem as well as the problems it generalizes. Furthermore, when introducing the problems we also give a brief literature overview or refer to surveys of that problem. In Section 2.2 we describe the two ways we transform a given warehouse layout into a graph data structure.

2.1. Problem description

In this thesis we study the *Electric Multi-Agent Pickup and Delivery* (EMAPD) problem. It is the lifelong version of the *Multi-Agent Pathfinding* (MAPF) problem with additional constraints imposed by the need to charge the batteries of the agents.

In the following sections we first introduce the sub-problems that make up the EMAPD and then formally define it.

2.1.1. Electric Vehicle Routing Problem

The *Electric Vehicle Routing Problem* (EVRP) is an extension of the *Vehicle Routing Problem* (VRP) with additional battery constraints. Given a fleet of n vehicles, m targets $T = [t_1, \dots, t_m]$ and a distance matrix $\mathit{dima} \in \mathbb{R}^{(m+1)\times(m+1)}$, the goal of VRP is to find routes visiting all targets while minimizing or maximizing an objective function. The *dima* contains the distances from the start depot of the vehicles to all targets, the distance between targets and the distance from a target back to the depot. Common objective functions are makespan and flowtime. Makespan is defined as the time it takes all agents to reach their goal. Flowtime, which is also known as sum of costs, is the sum of all travel times. This problem was first introduced by Dantzig and Ramser [8]. As it is a generalization of the **NP**-hard traveling salesman problem it iself is also **NP**-hard [9].

Since refueling gas-powered vehicles does not take much time or may not even be necessary as the vehicles can be refueled at their depots, refueling stops are mostly not considered in literature. However, this changes when using electric vehicles instead. Charging an electric truck or AMR takes up a considerable amount of time, their battery capacity is currently quite limited and the necessary charging infrastructure still has to be established in most countries. Therefore, Gonçalves et al. [10] developed the EVRP, which takes battery constraints into account. Erdelić et al. [11] and Asghari and Mirzapour [12] provide an overview of variants, charging strategies and solution methodologies of the EVRP.

2.1.2. Multi-Agent Pathfinding

The input of a *Multi-Agent Pathfinding* (MAPF) problem is an undirected graph $G = (V, E)$, a list of k start locations of agents $S = [s_1, \dots, s_k]$ and a list of k target locations $T = [t_1, \dots, t_k]$ with $s_i \in V, t_i \in V$ for all $i \in \{1, \dots, k\}$. The goal of MAPF is to find a conflict-free route for each agent from its start location to its goal location that maximizes (or minimizes) an objective function. Routes may contain waiting times at nodes.

Similar to VRP, the most common objective functions are makespan and flowtime. Here, flowtime additionally includes the time spent waiting. Surynek [13] and Yu et al. [14] proved that solving MAPF optimally is **NP**-hard for both metrics.

In classical MAPF problems time is discretized, the length of each edge is set to one, agents either stay at their goal or disappear once they reach it and conflicts occur only if two agents either occupy the same vertex or travel simultaneously along the same edge. However, there are more complex variants that may use directed graphs or non-unit edge-length like 2^k neighbor grids with $k > 2$ or MAPF problems embedded in Euclidean space. Another adaption is to consider "large" agents with a specific geometry and volume [15]. In this case new conflicts have to be considered as agents might get too close to each other even when occupying different vertices.

MAPF solvers can be divided into two different classes: optimal and sub-optimal solvers. Due to the problem being **NP**-hard, in most cases optimal solvers can only be used for instances with few agents. Felner et al. [16], Ma [17] and Stern [18] survey the state of the art algorithms to solve the MAPF problem. Stern et al. [19] give an overview of the different variations of the MAPF problem and provide benchmarks. Salzman and Stern [20] discuss the challenges and opportunities in MAPF and Multi-Agent Pickup and Delivery (MAPD) problems.

2.1.3. Multi-Agent Pickup and Delivery

The input of a MAPD problem is a graph $G = (V, E)$, a set of m agents $R = \{r_1, \dots, r_m\}$, their start locations $l_i(0)$ with $i \in \{1, \dots, m\}$ and a set of k tasks $T = \{\tau_1, \dots, \tau_k\}$ consisting of a pickup location $p_j \in V$ and a delivery location $d_j \in V$, $j \in \{1, \dots, k\}$. A solution of a MAPD problem is an assignment of conflict-free routes to agents, so that all tasks are completed while maximizing or minimizing an objective function. A task τ_i is completed once an agent has picked it up at p_i and delivered it to d_i. Here, an agent may only execute one task at a time. In MAPD the set of tasks T is not static but new tasks keep arriving during run time, making it an online problem. As this is an extension of MAPF, MAPD is also **NP**-hard and therefore computationally intractable.

Algorithms for solving MAPD can be divided into centralized and decentralized schedulers (the latter sometimes called decoupled). Centralized schedulers use a single omniscient central entity to assign the tasks to the agents and calculate conflict-free paths. Decentralized algorithms, on the other hand, lack this entity. Instead, the agents assign themselves to tasks and calculate their own paths. As centralized schedulers are provided with more information, their solution quality also tends to be better. However, due to the overhead in communication and added complexity, the efficiency of centralized schedulers does not scale well with the number of agents. By contrast, decentralized schedulers scale well with the size of the problem, as their calculations are distributed and the need for communication is limited. Lon et al. [3] evaluated the performance of centralized and decentralized approaches in multi-agent systems.

MAPD solvers can further be subdivided into solvers that look at the problem holistically and solvers that divide the problem into assigning the tasks and then solving the resulting

MAPF problem. The latter therefore usually combine a VRP or Pickup and Delivery problem (PDP) solver and a MAPF solver to address the MAPD problem. In recent works, Malus et al. [5] developed a real-time order dispatching algorithm using multi-agent reinforcement learning. The surveys of De Ryck et al. [2] and Fragapane et al. [21] contain an overview of the state of the art AMR task assignment algorithms.

Ma et al. [22] were the consider the MAPD holistically and developed a heuristic in 2017. They presented two decentralized approaches, Token Passing (TP) and Token Passing with Task Swaps (TPTS), and showed that they solve all well-formed MAPD instances. They defined a well-formed MAPD instance to contain a finite number of tasks and more non-task endpoints than agents. Here, a non-task endpoint is a node where agents can wait indefinitely without causing conflicts. Their approach was able to solve instances with up to 200 agents in real-time (less than one millisecond per timestep). Since then, many different solvers for the MAPD problem were developed [23, 24, 25, 26] and Liu et al. also studied the offline MAPD problem [27]. The current research interest in this subject is further exemplified by the number of papers published this year while working on this thesis [28, 29, 30].

2.1.4. Electric Multi-Agent Pickup and Delivery

The EMAPD problem we study in this thesis is the MAPD problem with discretized time and edge lengths, disk-like "large" agents and a directed graph and its embedding in 2-dimensional Euclidean space. Conflicts between two agents occur, if at any time the distance between them is less than twice their radius. A task $\tau = (p, d, t_{\text{due}})$ in EMAPD is extended to include a due time t_{due}. The pickup $p = (\ell_p, t_p)$ and delivery $d = (\ell_d, t_d)$ are extended to not only include the locations ℓ_p and ℓ_d but also the processing times t_p and t_d to pickup or deliver the task respectively. Additionally, we are given a set of charging stations $C = \{c_1, \dots, c_o\}$ and have to consider the battery level b of the AMRs at all times. The batteries of the robots are modeled to discharge linearly while driving, loading and unloading and charge linearly while charging at a charging station. We model idle AMRs to not use up any charge. The goal of our EMAPD problem is to minimize tardiness.

2.2. Graph generation

As a graph representation of a warehouse or processing plant is not always given, we may first have to transform the warehouse's layout into a graph data structure. Here, the layout is given as an image where white pixels indicate free space and everything else indicates occupied space. In the following we present two different approaches to achieve this: a grid-based approach and an approach using the Optimized Directed Roadmap Graph (ODRM) algorithm by Henkel and Toussaint [31].

In both cases we discretize edge weights by setting them to the rounded up travel time. The travel time is defined as the Euclidean distance divided by the agent's speed.

2.2.1. Grid-based approach

Given a layout's image, we overlay it with a 1x1-meter-grid and add nodes wherever the grid lines intersect if the corresponding point is free. We then add edges between horizontally, vertically and diagonally neighboring nodes if the edge does not go through occupied space. Finally, we add nodes for all charging stations and workstations and connect them to the nearest feasible node.

2.2.2. Optimized Directed Roadmap Graph

Henkel and Toussaint designed ODRM specifically to ease pathfinding and collision avoidance for multi-agent systems. They achieve this by encoding the layout's structure and properties in the graph. This leads for example to edges parallel to walls and structures similar to one-way streets in tight environments or roundabouts at wider intersections or around blocks in order to avoid congestion.

ODRM starts by generating random nodes in the free space and constructing edges using Delaunay Triangulation [32]. The locations of the nodes and the directions of the edges are then optimized using random path queries and stochastic gradient descent.

3. Scheduling

In this chapter we introduce the schedulers evaluated in this thesis. We divide the EMAPD into the assignment and the MAPF problem and solve them separately.

In the following, we describe the general sequence of events of the scheduling algorithms. We visualize this program flow in the activity diagram in Figure 3.1. Some schedulers do not plan the task assignment every scheduling call due to their long runtimes. If planning is necessary, the scheduler plans the assignment of tasks to robots and schedules the charging stops. Depending on the scheduler, this either happens successively or simultaneously. Possible collisions between agents are not taken into consideration during task assignment and the planning of charging stops. Afterwards, the goal locations of the robots are passed to our MAPF solver which finds collision-free routes for each robot. Finally, the routes are assigned to the robots.

First, we describe the charging strategies we use in Section 3.1. Then, we present the centralized and decentralized schedulers used to solve the assignment problem in Sections 3.2 and 3.3 respectively. Finally, we describe the pathfinding and collision avoidance algorithm used to solve the MAPF problem in Section 3.4.

3.1. Resource management

Two different charging strategies were considered in this thesis. The first strategy recharges the battery fully once the battery level drops below a certain threshold. The second strategy computes the optimal charging point(s) along a given route and charges the battery partially to finish the route above a threshold. This strategy is based on a paper by De Ryck et al. [33]. We chose this charging strategy as De Ryck et al. also developed a task assignment algorithm [4] based on it, which we present in Section 3.3.1. Both charging strategies use an underlying charging station management system that allows the AMRs to reserve charging time slots at specific charging stations. However, unlike the model of De Ryck et al., we do not know exactly when a robot will start charging due to possible routing conflicts with other robots. Therefore, we reserve longer charging slots than strictly necessary by adding a routing uncertainty safety buffer of 20 seconds.

3.1.1. Full recharging

To ensure that tasks already started are not unnecessarily delayed, this strategy differentiates between AMR carrying cargo or not. The AMR is sent charging once the battery level drops

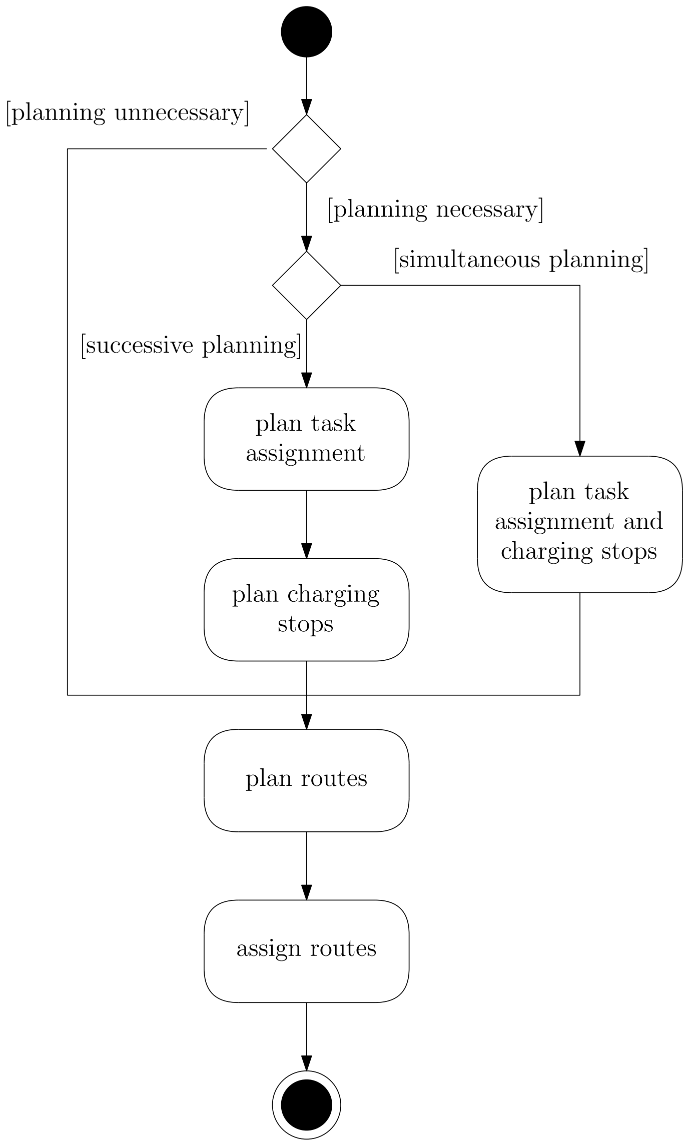


Figure 3.1.: Activity diagram showing the general program flow of the scheduling algorithms.

below 10% if the AMR is transporting goods or below 30% if it is not. These thresholds are chosen arbitrarily.

To select the best charging station, we perform the following steps: First the closest charging station to the AMR and its arrival time at the station are calculated. Then all available charging stations at the arrival time are considered. If all charging stations are occupied, the AMR has to wait, no matter which charging station it chooses. Therefore, all charging stations are considered. Out of these charging stations we choose the one that is closest to the AMR if the robot is not carrying cargo and the one with the smallest detour to the delivery location if it is.

3.1.2. Partial recharging

The original idea of De Ryck et al. was to minimize charging and travel time for a modified TSP problem while on a given route and instead use the charging opportunity at the depot to fully charge before the start and after the end of this route. However, since our routes neither start nor end at a depot, their approach had to be adapted. Instead, we iterate over the stops of the route and build up a list of potential charging stops. Once the charge of the robot drops below 20% we add the stop with the least detour from the list to the route and charge the robot for as long as possible. We then clear the list of potential charging stops and continue iterating. If at any point the rest of the route can be traversed without charging – i.e. finished with a charge of at least 20% –, we cancel the search.

To build up the list of potential charging stops we have to find the best charging opportunity between two stops s_i, s_{i+1}. Using a bidirectional A^* search, we can find the charging station with the shortest detour. We then go through the list of reservations at the charging station to find the first free slot that leaves the robot with at least 50% at the arrival at s_{i+1} in order to avoid pointless charging stops. If the robot cannot start charging right away we continue the A^* search and check the other charging stations while again only taking into consideration free charging slots that lead to at least 50% at stop s_{i+1}. We finally add the stop with the smallest sum of waiting time and detour to the list of potential charging stops. Additionally, we remove all potential charging stops from earlier iterations from the list that would lead to a charge of less than 40% upon arrival at s_{i+1}.

3.2. Centralized schedulers

When choosing which centralized schedulers to evaluate, we quickly realized that most algorithms in literature only solve the offline also known as one-shot assignment problem with e.g. Malus et al. [5] being the rare exception. However, the description of their algorithm was too brief to re-implement it. We therefore decided to implement a basic rule-based algorithm and two algorithms based on existing VRP solvers.

3.2.1. Rule-based

The rule-based scheduler is an adaption of the simple earliest due date first heuristic. The tasks are sorted by the difference of due date and the travel time from the pickup to the delivery location of the task. Then the tasks are assigned in order to the closest idle robot until either all tasks are assigned or no idle robots remain. This scheduler uses the full recharging strategy described in Section 3.1.1.

3.2.2. VRP solver

The following two schedulers are based on traditional VRP solvers that solve the VRP problem using heuristics. In order to use these solvers, we have to transform an instance of EMAPD into an instance of VRP. Given an instance of EMAPD with n robots $R = \{r_0, r_1, \dots, r_{n-1}\}$ and m open tasks $T = \{(p_0, d_0), (p_1, d_1), \dots, (p_{m-1}, d_{m-1})\}$, we create a $(n+m+1) \times (n+m+1)$ distance matrix *dima*. The start depot of robot $r_i \in R$ is set to i and the end depot of all robots is set to $n+m$. If robot r_i is already working on a task, we set $t_r[i]$ to the remaining time to finish this task, otherwise to 0. We also set the start location $s[i]$ of each robot $r_i \in R$ to its current location if it is not working on a task and the goal location of the task otherwise. For each task $\tau_j = (p_j, d_j) \in T$ the variable $t_{\tau_j} := \text{dist}(\ell_{p_j}, \ell_{d_j}) + t_{p_j} + t_{d_j}$ describes the time it takes to pick up the task, drive from the pickup location to the delivery location and drop the task off.

First, for each robot $r_i \in R$ we set the cost to travel from its start depot to the end depot to $t_r[i]$ as the robot at least has to finish the task it started already:

$$\forall r_i \in R : \textit{dima}[i][n+m] = t_r[i]$$

Next, we define the cost of assigning the first new task, which is represented by driving from the start depot to the first node. In this case, we set the cost of assigning that new task $\tau_j = (p_j, d_j) \in T$ to the remaining task time of the already initiated task $t_r[i]$ plus the time to travel from the robot's start location $s[i]$ to the start location ℓ_{p_j} of τ_j and performing the task:

$$\forall r_i \in R, \forall (p_j, d_j) \in T : \textit{dima}[i][n+j] = t_r[i] + \text{dist}(s[i], \ell_{p_j}) + t_{\tau_j}$$

We set the cost of assigning task τ_j after τ_i to the sum of travel time from the delivery location ℓ_{d_i} of τ_i to the pickup location ℓ_{p_j} of τ_j and the time t_{τ_j} it takes to perform τ_j:

$$\forall (p_i, d_i) \in T, \forall (p_j, d_j) \in T, i \neq j : \textit{dima}[n+i][n+j] = \text{dist}(\ell_{d_i}, \ell_{p_j}) + t_{\tau_j}$$

Finally, we set the cost to travel from a task to the end depot to zero as a robot does not have to visit any more locations after finishing the final task:

$$\forall (p_i, d_i) \in T : \textit{dima}[n+i][n+m] = 0$$

The remaining undefined values of the distance matrix are set to some large constant. The distance matrix is passed to a VRP solver, which provides an assignment of robots to tasks. This assignment is first passed to the partial recharging strategy described in Section 3.1.2 and its result to the pathfinding and collision avoidance algorithm (see Section 3.4.4) which in turn provides a valid solution to the EMAPD problem.

Due to their long run-times not all open tasks are considered and the schedulers are only executed once either a robot is idle or the number of tasks assigned to robots drops below a certain threshold.

We considered implementing and evaluating the algorithm by Quiroz et al. [34]. They use a genetic algorithm to solve the task assignment problem whenever new tasks arrive and ICBS and Prioritized Planning to solve the MAPF problem. As they do not use information from previous iterations and practically perform the transformation described above to generate the input data of their genetic algorithm, we refrained from using their approach and instead used optimized state of the art VRP solvers.

3.2.2.1. OR-Tools

This scheduler uses Google's OR-Tools software suite [35] to solve the VRP problem. The global span of a solution of OR-Tools is the maximum of the distances of the routes. In order to minimize the length of the longest route we set the cost coefficient of the global span (GlobalSpanCostCoefficient) to 100. In order to quickly gain a solution we set the first solution strategy to PARALLEL_CHEAPEST_INSERTION which iteratively inserts the cheapest node at its cheapest position. Otherwise we use the the default search parameters (DefaultRoutingSearchParameters).

3.2.2.2. HGS-CVRP

The VRP solver used in this scheduler is the Hybrid Genetic Search (HGS)-CVRP solver by Vidal et al. [36, 37]. Since it solves the Capacitated VRP, we need to assign a demand to each node as well as a capacity to each vehicle. This includes the nodes representing the vehicles depots, as they also have to be visited or rather routes need to start there. We set the demand of every node except the end depot to 1 to ensure that each task is assigned exactly once. We assume that in an ideal task assignment the tasks are distributed somewhat evenly between the robots and therefore set the vehicle capacity to $\lfloor \frac{m}{n} \rfloor + 2$, where m is the number of tasks and n is the number of robots. This is necessary because this solver minimizes flowtime rather than makespan and it might otherwise assign all tasks to a single robot. Setting it to $\lfloor \frac{m}{n} \rfloor + 2$ ensures that a robot may only be assigned to at most one more task than in an even distribution, as visiting the robot's starting depot decreases the capacity by one.

We use the default algorithm parameters except for limiting the number of iterations without improvement (nbIter) to 2000 to speed up the calculation and disabling SWAP* (useSwapStar). SWAP* is a recent improvement of the HGS-CVRP algorithm, however we cannot make use of it as it introduces a new neighborhood that requires the nodes as well as the depots to be positioned in Euclidean space.

3.3. Decentralized schedulers

Unlike the schedulers presented in the previous section, the task assignment for the following schedulers is not managed by a central entity. Instead, the AMRs negotiate among themselves to find the best task allocation.

We considered implementing a holistic scheduler, e.g. the one developed by Ma et al. [22]. However, we decided against evaluating these algorithms in this thesis as they oversimplify the problem by either assuming the graph to be an undirected grid with unit edge-length, the agents to not have a volume or supposing that non-task endpoints where agents can wait indefinitely exist and are known in advance.

We therefore settled on using and adapting the recently published algorithm of De Ryck et al. [4] as well as the SET-MASR algorithm [38], which was one of the first algorithms to solve the online assignment problem decentrally.

All schedulers in this section use the partial recharging strategy (cf. Section 3.1.2).

3.3.1. Basic auction

The algorithm introduced in this section was developed by De Ryck et al. [4]. It is an auction-based algorithm with a single central entity taking the role of auctioneer. However, it is still considered a decentralized scheduling algorithm as every AMR can take the role of auctioneer and the bid of each robot is calculated using only its own knowledge, i.e. its location and battery level, its list of assigned tasks and the new task.

The objective of the task assignment is to fulfill a combination of the following team objectives. The first team objective is MiniSum, whose goal is to minimize the sum of the travel time of the routes of all robots. The goal of MiniMax on the other hand, is to minimize the longest route. Therefore, MiniSum assigns new tasks to the robot with the shortest detour while MiniMax assigns tasks to have them evenly allocated. These objectives directly lead to MiniSum and MiniMax bidding rules where each bidding rule achieves its respective objective. The MiniSum bid c_{ms} is equal to the additional travel time gained by adding the new task to the route of the robot. The MiniMax bid c_{mm}, on the other hand, is defined as the total travel time of the robot after adding the new task. A combination of these objectives leads to an assignment rule that still takes locality into consideration while also making sure that no route gets too long. To combine them, De Ryck et al. introduced the parameter $\epsilon \in [0, 1]$ leading to the following calculation of the bid:

$$c_{\text{total}} = \epsilon \cdot c_{\text{ms}} + (1 - \epsilon) \cdot c_{\text{mm}}$$

This allows the user to decide whether to focus on an even task distribution by setting ϵ close to 0 or to minimize travel time and therefore energy consumption by setting ϵ closer to 1. In this thesis we want to minimize both objectives and therefore set ϵ to 0.5.

The tasks are offered up for auction in the order they arrive in. Each robot calculates c_{total} and submits their bid. The new task is then allocated to the robot with the lowest bid. In order to calculate the bid the best insertion position – i.e. the position leading to the shortest detour – has to be found. The heuristic we use iterates through all tasks already assigned to find this position. Therefore, we limit the maximum number of tasks assigned to a robot to 50.

We do not take charging into consideration when trying to find the optimal insertion position. However, we calculate the cost – i.e. the travel time – of the route after inserting possible charging stops.

3.3.2. Trading auction

This scheduler uses the previously described scheduler by De Ryck et al. as a base. After assigning the task to the robots, the AMRs start trading the tasks among themselves. This allows them to trade away tasks that may now lead to unnecessarily long detours or which may be better suited for other robots.

The AMRs take turns offering their worst task – i.e. the task leading to the largest detour – for auction. Then the other robots calculate the best insertion position in their route for this task and the additional cost of adding this task to their assignment. If the cost of one of the robots is smaller than that of the offering robot by a given percentage (in our case 3%), the task is traded to the lowest bidder. We add this mark up to avoid transactions that extend the run-time while not significantly improving the solution.

If a task is successfully traded away, the offering AMR can continue with the next worst task once it is its turn again. If it fails to trade away the task, the robot cannot offer its tasks again until it has accepted a task from another AMR. The algorithm stops once all robots are unable to offer up a task for trading. As each trade results in a decrease in total cost, this algorithm terminates.

We do not take charging and the battery level of robots into account during trading. Instead, we execute the partial recharging algorithm described in Section 3.1.2 once trading has finished. Due to the added complexity, we further limit the maximum number of tasks assigned to a robot to ten.

3.3.3. Tardiness penalty auction

A major problem of the algorithm of De Ryck et al. is that tasks can be continuously pushed back to the end of the tour of a robot, leading to tasks that may never be fulfilled. This scheduler tries to avoid these situations by punishing tardiness by means of including it in the cost calculation of a route.

The adapted cost calculation is done by figuring out the difference between estimated task completion time and due date of each task assigned to the robot if the task is assessed to be too late. These differences are then summed up, multiplied with a delay factor δ_{delay} and added to the travel time of the route to calculate the total cost of the route. Except for the cost calculation, this scheduler is the same as the De Ryck auction scheduler described in Section 3.3.1.

3.3.4. SET-MASR

This scheduler is based on the SET-MASR algorithm by Viguria et al. [38]. It is an auction-based algorithm in which subsets of already assigned tasks can be traded between robots. This has the advantage of offering additional context and information to the AMRs. Given a route of a robot and a subset of tasks, we calculate the cost of inserting these tasks into the route by greedily inserting each task at the position that leads to the shortest detour and summing up the travel times of the detours. Battery levels and potentially necessary additional charging stops are not considered during this calculation. Similar to the trading auction, the cost of assigning the tasks to a different AMR has to be smaller than the original assignment by at least 3% to avoid trades that do not lead to notably different results.

As an exhaustive search for the best subset of tasks to remove from a robot is not feasible even when the size of the subset is given, the subsets are also generated greedily. We iterate over all tasks in the route of the AMR and find the one leading to the largest detour. We then remove this task from the route, add it to the subset and repeat until the subset is of the required size.

In order to avoid offering the same subsets of tasks from the same robot multiple times during bidding, every robot has an announcement list of tasks. We start with a subset of size one and a random robot as auctioneer. It offers its worst subset of tasks – in this case a single task – to the other AMRs and receives the offers. If an offer leads to a smaller detour it then trades off the subset and the winning AMR adds the received task(s) to its announcement list. Otherwise the offering AMR removes all tasks in the subset from its announcement list. It then passes on the role of auctioneer to another robot. Once no more tasks are announced, the size of the subset is increased by one and the announcement lists of the robots are reset to their assigned tasks. We limit the number of tasks that can be assigned to a robot to 50.

As the original algorithm by Viguria et al. did not consider lifelong scheduling, we have to adapt it in order to deal with newly arrived tasks. We therefore add a dummy robot that trades these tasks off starting with the task that arrived first and continuing on in a FIFO fashion. We also limit the announcement list of each robot to tasks that arrived in the last minute to avoid having to redo the calculations of the previous time step.

3.4. Pathfinding and collision avoidance

We use a combination of a modified Continuous-time Conflict-Based Search (CCBS) [39, 40] and an adapted Cooperative A* [41] search to find conflict-free routes. CCBS is an extension of the Conflict-Based Search (CBS) algorithm by Sharon et al. [42] that guarantees optimal solutions without requiring discretized time. Furthermore, unlike basic CBS, it works on graphs with non-unit edge-length and "large" agents with a volume. We first describe CBS, CCBS and Cooperative A* and then present our algorithm.

3.4.1. Conflict-Based Search

CBS is an optimal and complete MAPF solver that minimizes flowtime and uses a two-level approach. At the high level it builds up a constraint tree (CT) based on the conflicts of the agents.

It starts by calculating a route for each agent without considering conflicts using A* [43]. Then it finds all conflicts between agents, i.e. points in time when agents either share a vertex or travel along the same edge in opposite directions. A conflict $c = (r_i, r_j, v, t)$ at time t and node v between agents i and j can be resolved by forbidding either agent i or agent j to be there at that time. Therefore, CBS chooses a conflict and resolves it by branching on the CT and adding a constraint forbidding i to be at v at time t on one branch and j to be there on the other branch. Then it runs the low-level search that takes the constraints imposed upon the agents from this and higher levels of the CT into account and tries to find a new solution consisting of a set of paths following these constraints for each new branch created. If a solution is found it is added to the open set. Note that this solution may still contain conflicts. Next, CBS chooses the solution with the lowest cost – i.e. the smallest flowtime – from the open set. It continues this procedure as long as conflicts remain in the solution with the lowest cost, otherwise it returns this solution. As all previously processed solutions contained conflicts and resolving conflicts cannot lead to lower flowtime, the solution returned is optimal. The low level search is a modified A* search that calculates the routes of each agent separately and operates on a three-dimensional search space (space and discretized time).

3.4.2. Continuous-time Conflict-Based Search

CCBS is an extension of CBS that allows solving MAPF instances for agents with arbitrary geometric shape on any directed graph and that allows agents' actions to have any duration. At the high level, CCBS still builds up a constraint tree, branches on conflicts and returns the first solution found without conflicts. However, a conflict in CCBS is not necessarily limited to a time and vertex but instead is a conflict between two actions of agents that lead to a collision.

CCBS resolves a conflict $c = (a_i, t_i, a_j, t_j)$ where agent i starts performing action a_i at time t_i and agent j likewise by adding *unsafe intervals* as constraints. An unsafe interval for agent i is defined as the time interval where performing action a_i leads to a conflict with agent j which is performing action a_j at time t_j. It starts at time t_i and ends once agent i can safely perform action a_i without causing a conflict with agent j performing a_j.

At the low level, CCBS uses an adapted safe interval path planning (SIPP) [44] algorithm. A maximal *safe interval* for a vertex v is defined as the interval in which staying at v does not lead to a collision for an agent but extending the interval in any way would. SIPP uses an A*-based search that complies with the safe intervals to find a valid path for a given agent.

The authors of CCBS adapted SIPP to work with unsafe intervals in order to use it as their low-level search. Given a constraint $(a_i, [t_i, t_i^u))$ that forbids agent i from performing action a_i in the interval $[t_i, t_i^u)$ they distinguish whether a_i is a wait action or not. If a_i is a wait action at node v, the safe intervals at node v are split to not allow waiting during that time. On the other hand, if a_i is a move action from v to v' and during the search the agent i arrives at time $t \in [t_i, t_i^u)$ at v, then it is forbidden from moving to v' until t_i^u and a wait action is added instead.

Andreychuk et al. [40] improved CCBS by e.g. adding Disjoint Splitting (DS) [45]. DS ensures that the search space is partitioned disjointly when branching during conflict resolution. They achieved this by adding positive constraints that – instead of forbidding – require the agent to perform the specific action during the specified interval. Given a conflict (a_i, t_i, a_j, t_j), DS branches by adding the negative constraint $\overline{(a_i, [t_i, t_i^u))}$ in one branch and the positive constraint $(a_i, [t_i, t_i^u))$ in the other.

3.4.3. Cooperative A*

Cooperative A* is an A*-based MAPF solver that plans the agents' routes one by one and reserves the paths of the previously planned agents as impassable. It does so by using a reservation table that contains the intervals when it is occupied for each vertex. An issue of this search is that its performance relies heavily on the order in which the paths of the agents are planned. Therefore, the author of [41] recommends using prioritized planning [46] to order the agents.

3.4.4. Our algorithm

There are several issues with CCBS that do not allow us to simply use it to solve the pathfinding problem of EMAPD.

The first issue is that CCBS works with continuous time while EMAPD does not. Therefore, we modify CCBS to discretize time by changing a given constraint $(a_i, [t_i, t_i^u))$ to $(a_i, [\lfloor t_i \rfloor, \lceil t_i^u \rceil))$.

Secondly, in classical MAPF two robots may not share the same goal location. However, here two robots may very well be assigned to different tasks with the same target location or target locations that are too close to each other. Hence, we need to resolve these target conflicts. We do so by first sorting the agents by the travel time to their target without considering collisions. Starting with the robot closest to its goal, we then check if this robot's goal is too close to another robot's goal. If that is the case, we reroute one of the robots to a different target. We continue with the other robots but only check for conflicts with robots that are further away from their target than them.

We decide which robot to reroute based on multiple criteria:

- If only one of the robots was rerouted before, we reroute it again. This is to minimize the total number of changed targets.
- If none of the robots was rerouted before but one of the robot just idles at its goal, we reroute this agent.
- Otherwise we just reroute the agent further away from its target.

Algorithm 3.1 shows the high-level pseudocode for resolving target conflicts.

When resolving a conflict between robot i and j where we reroute robot j, we only check for collisions with robots that are closer to their goal than i to avoid unnecessary collision checks.

In the following we show that the resulting assignment does not contain target conflicts. Assume that a conflict remains between agents r_a and r_b and that r_a is closer to its initial goal than r_b. Note that at least one of the robots was rerouted as the conflict would have been found otherwise. We distinguish two cases:

1. the goal of r_a was changed last
2. the goal of r_b was changed last

In the first case the last change to the goal of r_a must have happened after checking collisions between the two robots. However, after this check r_a might only have resolved target conflicts with agents further from the goal than r_b. Assume that the last reroute of r_a happened when resolving a conflict with r_c. As r_c is further from its goal than r_b, r_a would have checked for and resolved collisions with r_b, which is a contradiction to the assumption.

Algorithm 3.1: Resolving target conflicts

```
Input: Graph $G = (V, E)$, robots $R = \{r_0, \dots, r_{n-1}\}$, target locations
       $L = \{l_0, \dots, l_{n-1}\}$
Output: Non-conflicting target locations
1  sort $R$ by travel time to target
2  for $i = 0$ to $n - 1$ do
3      for $j = i + 1$ to $n - 1$ do
4          if not collide$(i, j)$ then
5              continue
6          if rerouted$(i) \oplus$ rerouted$(j)$ then
7              if rerouted(i) then
8                  reroute(i)   // only checks for collisions with robots $\leq j$
9              else
10                 reroute(j)   // only checks for collisions with robots $\leq i$
11         else if idle(i) then
12             reroute(i)       // only checks for collisions with robots $\leq j$
13         else
14             reroute(j)       // only checks for collisions with robots $\leq i$
```

In the second case the last change to the goal of r_b must also have occurred after checking for conflicts with r_a. However, afterwards r_b may only resolve target conflicts with robots further away from their goal than r_a, which would result in checking for and resolving conflicts with r_a. Therefore, the target assignment resulting from algorithm 3.1 does not contain any target conflicts.

Another issue of CCBS is that it solves the **NP**-hard MAPF problem optimally, leading to potentially unacceptable runtimes. Thus, after 4000 iterations of the high level search, we switch to a different MAPF-solver based on Cooperative A*. We set the rank of each agent to its travel time to its target, unless it idles at the target. In that case we set the rank to some large constant. We block the start location of each agent only for a limited time as we assume that they leave their start location quickly and do not want to unnecessarily block other agents. We then plan the route of the agents in ascending rank. If one of the robots is unable to reach its target, we differentiate two cases:

- If the robot is unable to wait indefinitely at any reachable vertex, we restart the entire search and block the start location of each agent forever, therefore enabling all robots to wait there indefinitely.
- Otherwise, we re-rank the robot to be of highest rank and run the pathfinding algorithm for it again. If its target is still unreachable, it instead travels to the node that is closest to its target at which it can wait indefinitely.

Furthermore, in order to save time we limit the search space of the algorithm by canceling the search after a number of time steps that depend on the optimal travel time to the target. This lowers the chances of finding a path to the goal but avoids searching the entire graph. Lastly, we change the collision detection algorithm to find conflicts between agents with a volume. This is done by saving the agent's moves in the reservation table and checking for conflicting moves using the subroutines provided by the CCBS implementation.

In summary, we run the modified CCBS search with Disjoint Splitting for 4000 iterations. If CCBS is unable to find a solution, we switch to the modified Cooperative A* search.

4. Simulation framework

In this chapter we introduce the discrete-event simulation framework used in this thesis to evaluate the scheduling algorithms described in the previous chapter.

In situ, the AMRs communicate their current positions and battery charge directly to the scheduler. The scheduler takes this information as well as the list of open tasks and the reservation table of the charging stations into consideration and assigns each robot a route. Cleanly separating the scheduler and the AMRs allows us to replace the latter with a simulation. Further utilizing this separation, we implement the strategy design pattern to easily exchange schedulers.

Given a scenario consisting of a graph, a set of AMRs, an end time and a set of arriving tasks, we simulate this scenario as a sequence of discrete-time-events. As a reminder: Task consist of a pickup location ℓ_p, a delivery location ℓ_d and a due time t_{due}. Graphs are generated from layouts by either overlaying them with a grid or by running the ODRM algorithm which generates a graph that is optimized to avoid conflicts between agents due to its structure. Charging stations are modeled as vertices and robots start charging immediately upon arrival. We simplify the charging process by assuming linear charge and discharge of the battery.

Figure 4.1 shows the main procedure for simulating a scenario in an activity diagram. All pending events are stored in a priority queue that is firstly sorted by the event's time and secondly by its priority. We distinguish between the following seven different event types, listed in decreasing priority:

- time update event
- arrival event
- task complete event
- begin task event
- new task event
- trigger plan event
- begin task after planning event

We first resolve time update events. They just set the simulation's and the scheduler's time to the event's time.

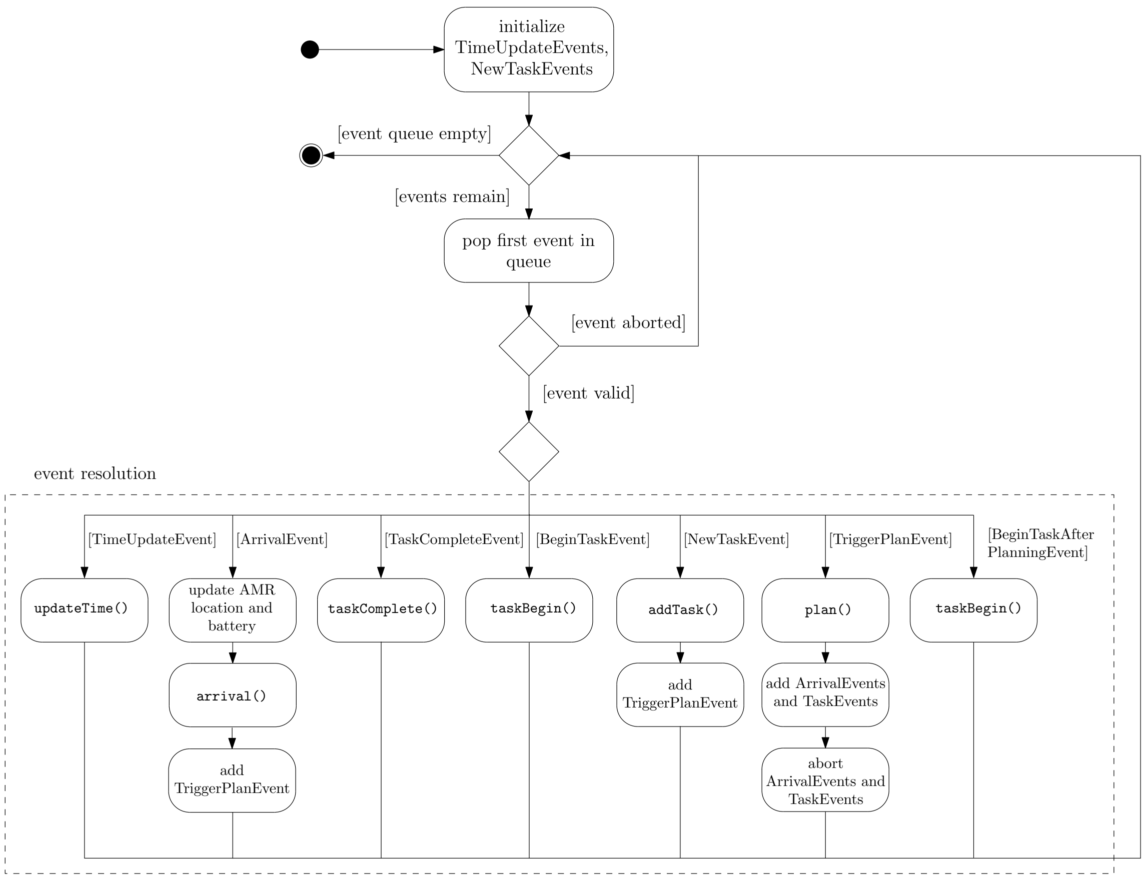


Figure 4.1.: Activity diagram showing the core mechanics of the simulation framework and event resolution based on Figure 14.3 in [47]. Function calls to the scheduler are written in typewriter style.

Next, we simulate the arrival of an AMR at its target location. We update the robot's battery and its current location and pass the information on to the scheduler by calling the `arrival()` function. As this robot needs a new target, we also schedule a new trigger plan event by adding it to the queue.

Completing a task and beginning to work on a task are announced to the scheduler by calling the `taskComplete()` and `taskBegin()` respectively.

Whenever new tasks arrive in the system, we inform the scheduler by calling `addTask()` and scheduling a trigger plan event.

The trigger plan event calls the `plan()` function of the scheduler, which in turn returns a list of routes that are assigned to the robots. If this leads to a different arrival time for an AMR than before, we abort the previous arrival event and schedule a new one. Similarly, we abort and (re-)schedule task complete events and begin task events as necessary. As the `plan()` function might assign a robot to a task starting at its current position, an AMR might start a task right after planning. In this case we add a begin task after planning event, which just calls the `taskBegin()` function.

We initialize the simulation by adding time update events for every time step until the end time of the scenario and begin task events for every task in the scenario. The simulation ends once either no events remain in the queue or the time of the next event is later than the end time of the scenario.

5. Experimental evaluation

In this chapter we evaluate the performance of the scheduling algorithms described in Chapter 3.
In Section 5.1 we describe the experimental setup consisting of the layouts, task distributions and number and type of robots used. In Section 5.2 we present the hardware and software used to evaluate the schedulers. In Section 5.3 we evaluate the performance of the schedulers' task assignments qualitatively and quantitatively and discuss our findings.

5.1. Experimental setup

5.1.1. Study design

We simulate all scenarios for 48 hours in one-tenth-of-a second time steps. We chose this time frame to be long enough that schedulers have a chance to settle in and go through multiple charging cycles but short enough that the simulation runs in a feasible amount of time. We stop adding new tasks after 44 hours in most scenarios to allow the robots to finish off the remaining tasks. In order to rule out inconsistencies during startup and wind down, we start evaluating the solution quality after one hour of simulation and stop at 44 hours of simulation unless otherwise stated.

We model the robots to be disk-shaped with a radius of 0.4 m and move at a speed of 2 m/s. Their battery is based on the battery of the Linde E-14 truck [48]. The percentage discharge and charge rates per second of the AMRs are displayed in Table 5.1. We model the robot to not use any charge when idle and to use the same amount of energy when driving empty as when driving while carrying a task. Furthermore, we model the robot's energy consumption to be the same when processing tasks as when it is driving. Therefore, when fully charged, the robots can perform actions for approximately 9 hours before the battery is depleted, and it takes 2 hours and 40 minutes to fully charge an empty battery. When generating robots, we place them randomly on the layout with a random battery charge between 40% and 100%.

Tasks can arrive at every timestep. When generating the start times of tasks, we use a log-normal distribution since it only provides positive values. Given the desired number of tasks per timestep n_t, we set both the mean m and variance v to n_t and transform m and v to the log-normal parameters μ and σ:

$$\mu = \log \frac{m^2}{\sqrt{v + m^2}} = \log \frac{n_t^2}{\sqrt{n_t + n_t^2}} \text{ and}$$

Operation	Percentage battery change per second
Processing	-3.118×10^{-3}
Driving	-3.118×10^{-3}
Charging	$+1.042 \times 10^{-2}$

Table 5.1.: Percentage discharge and charge rate per second of the AMRs.

$$\sigma = \sqrt{\log \frac{v}{m^2} + 1} = \sqrt{\log \frac{1}{n_t} + 1}$$

We start with time $t = 0$ and for every task generated add $\mathcal{LN}(\mu, \sigma)$ to t and set the task start time of the next task to $round(t)$. This creates n_t tasks per timesteps on average with a variance of n_t and allows multiple tasks to arrive at the same time.

We use the grid-based approach (cf. Section 2.2.1) to generate the graphs for every scenario but one. We tried using ODRM (cf. Section 2.2.2), but due to its long runtime we were unable to generate graphs with enough vertices to avoid collisions in most layouts. The graphs generated by ODRM worsened existing chokepoints and generated new ones which sometimes even led to deadlocks since the complete CCBS algorithm reached its iteration limit without finding a solution and the Cooperative A* algorithm was only able to find a solution in which all robots stay at their current location.

In order to gauge the performance of the schedulers, we created ten different scenarios using the following five different common warehouse layouts:

- two differently sized layouts based on matrix production
- one layout based on cross-docking
- one layout based on a car production line and its connected supermarket
- one layout based on a basic storage warehouse

These layouts differ in their use, the number and distribution of workstations, the flow of goods and therefore also the susceptibility to agent conflicts.

5.1.2. Matrix layouts

Figure 5.1 shows the two matrix production layouts. The large instance is taken from Disselnmeyer [49] who in turn adapted the original layout of Kabir and Suzuki [50]. It is 185 m×185 m with 41 workstations and 3 charging stations spread out on a 25 m×25 m grid with some bare spots throughout. We reduce the number of AMRs from 40 in Disselnmeyer's thesis to seven and ten in two scenarios each. This reduction is necessary, as only three charging stations are present and only one robot may charge at one station at a time. Disselnmeyer was able to run his simulation with 40 AMRs, as he did not take collisions between robots into account.

We created two different sets of tasks for this layout. In both sets the robots have five minutes to perform the task after it is first announced and the pickup and delivery locations are chosen at random. In the first set an average of seven tasks per minute are generated, leading to a total of 18475 tasks in 44 hours. In the second set we reduce the number of tasks to 11085 by removing 40% of the tasks of the first set.

We chose these two sets as the robots will be overwhelmed in the first set and just able to deal with the tasks in the second one. Therefore, we can see how the different schedulers handle these two system loads. Furthermore, by running each set with seven and ten robots, we may infer whether increasing the number of robots leads to an increase in performance.

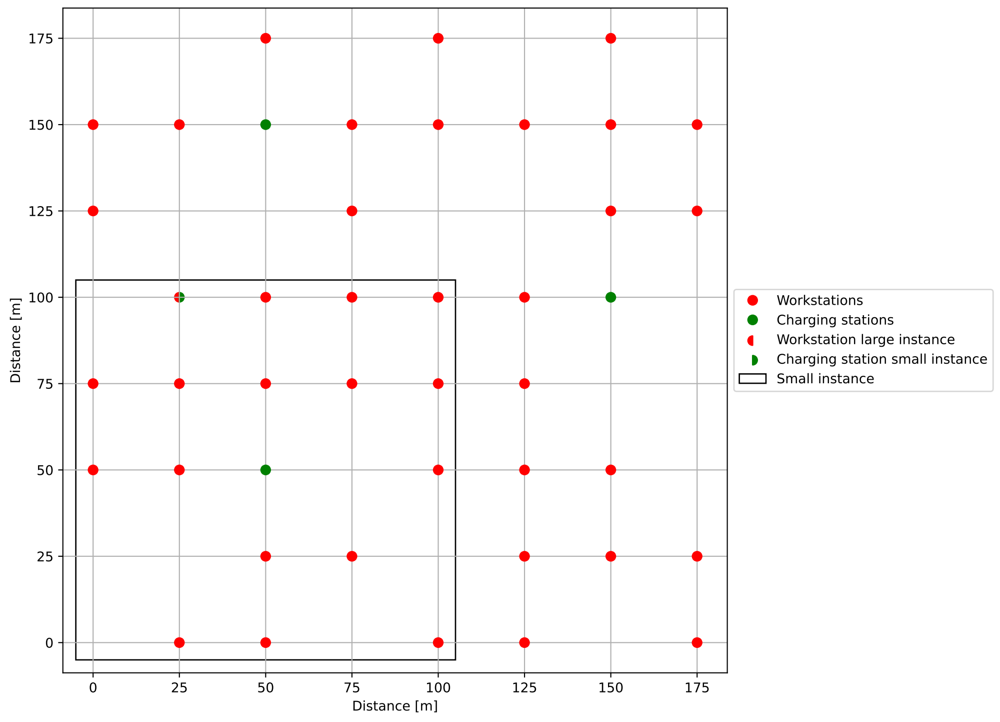


Figure 5.1.: The two matrix production layouts used for evaluation. The workstations are marked red, the charging stations green and the smaller instance is separated by the box in the bottom right. The station at $(25, 100)$ is a workstation in the large instance and a charging station in the small instance.

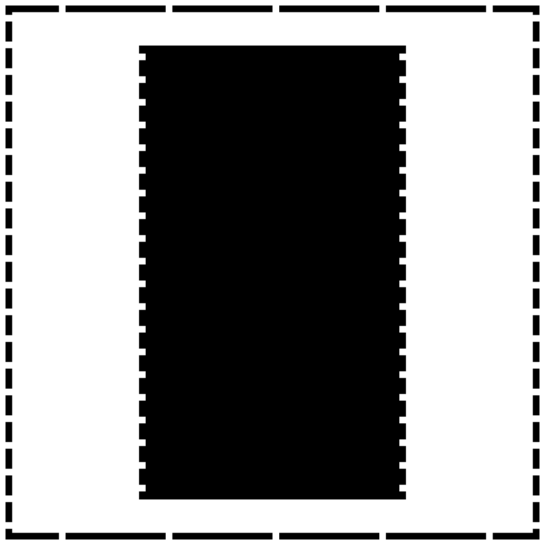


Figure 5.2.: The layout of the cross-docking warehouse. Charging stations are located at the top and bottom. Goods enter the system from the left, are redistributed and finally leave it on the right.

The small matrix production layout is 100 m×100 m with 16 workstations and two charging stations arranged on the same grid as the previous layout. On average, five new tasks are added per minute, resulting in a total of 13195 tasks. A task is due two minutes after being added to the system. We run this scenario with six robots.

5.1.3. Cross-docking layout

The third layout is based on cross-docking: a warehouse with little or no storage, where products enter the warehouse, are redistributed and leave the warehouse quickly. Figure 5.2 shows the layout we use. It is a 80m×80m warehouse with ten charging stations situated at the bottom and at the top. Products enter the warehouse from the left. AMRs transport them to the sorter in the middle, which they enter from the left. There, they are redistributed and repackaged as necessary. Afterwards, AMRs pick them up and transport them to the exits on the right.

We generate tasks requiring the AMRs to either pick up products at the entry and delivery them to the sorter or to transport them from the sorter to the exit of the warehouse. The exact locations are chosen at random. We run this scenario with 22 AMRs and generate on average 22 tasks per minute, resulting in a total of 58208 tasks.

5.1.4. KARIS layout

The fourth layout is based upon a car production line with a connected manufacturing supermarket, shown in Figure 5.3. It is taken from the final report of the federal German research project KARIS PRO [51]. The layout spans 52.2 m×85.6 m and models a quarter of the entire production line, the same area that is used for the research project.

There are 20 workstations placed at the production line in the bottom right and 100 shelves containing the required car parts in the supermarket in the top left. Six charging stations in total are placed above the supermarket and below the production line. This is an addition to the setup of [51], as they did not consider charging. Tasks require goods to be moved from a random shelf in the supermarket to a random workstation in the production line. We run this scenario with 13 robots and generate 17185 tasks or an average of 6.5 tasks per minute. Tasks are due 10 minutes after entering the system.

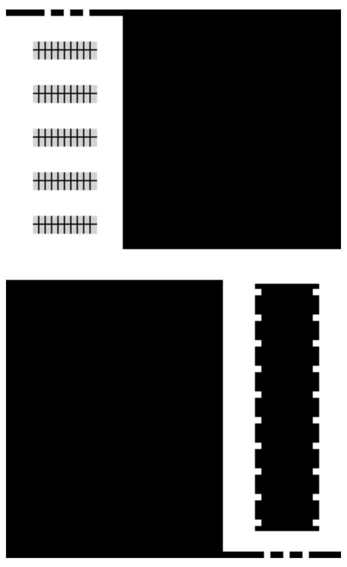


Figure 5.3.: The layout of the KARIS car production line and its manufacturing supermarket. The cars are assembled in the bottom right and the supermarket and its shelves are situated in the top left. Charging stations are placed above the supermarket and below the production line.

5.1.5. Slap layout

Finally, the last layout is of a basic storage warehouse called Slap pictured in Figure 5.5. This layout, as well as the task distribution and task due times are provided by the TU Dortmund. Its dimensions are 175 m$\times$74 m and it contains four entries, four charging stations, ten exits and 6504 shelves for storage in the grey area.

As the free space (indicated in white) is relatively small, we can use ODRM to generate a graph of the free space in one scenario. In the other two scenarios, this graph is grid-based. We do this in order to be able to compare the schedulers' performance on a graph that is optimized to reduce collisions (ODRM) with their performance on a grid-based graph.

In the following, we describe the graph of the shelf structure and ignore the rest of the layout. Shelves are arranged in blocks that are 4 shelves wide and which can be traversed but do not allow moving to adjacent blocks. The borders of blocks are indicated by the black lines in the depiction. Every shelf is modeled as a vertex and is connected to its horizontal and vertical neighboring shelves by edges if they are in the same block. We connect the graph of the free space and the graph of the shelf structure by adding connections between shelves at the edge of the shelf structure with their nearest neighbor in the graph of the free space.

All scenarios using this layout are run with nine AMRs. Tasks are either from an entry to a shelf or from a shelf to an exit. In two scenarios we use a task distribution provided by TU Dortmund containing 10281 tasks. They only differ in their graph representation of the layout: One uses a grid-based graph and the other a graph generated by ODRM. Figure 5.4 shows the task start times in a histogram. It is noteworthy that the number of new tasks increases extremely after approximately seven hours and 42 hours and that between hours 14 and 18 no new tasks are added. In the third scenario we use the same distribution but reduce the number of tasks by 25% by removing every fourth task to 7711 tasks. In all three scenarios, no new tasks are added after roughly 43 hours.

Table 5.2 summarizes the ten scenarios we use to evaluate the performance of the seven schedulers.

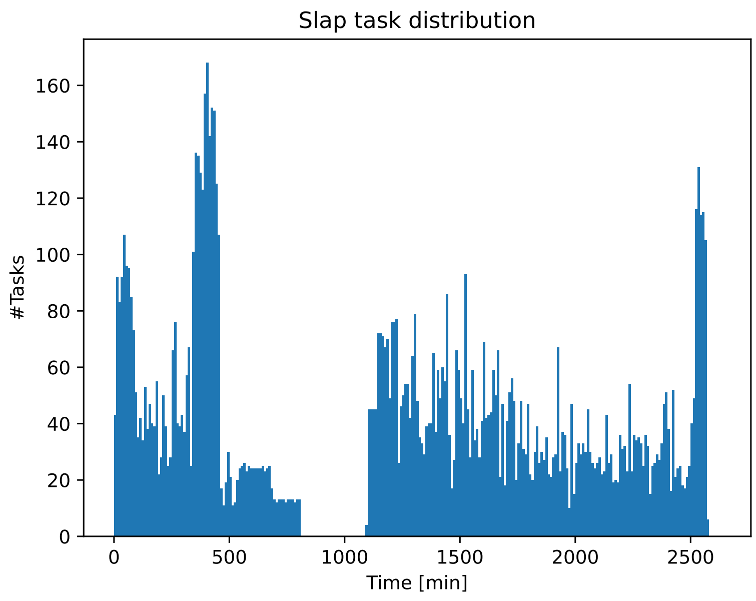


Figure 5.4.: Histogram showing the task start times of the Slap scenario with 10281 tasks

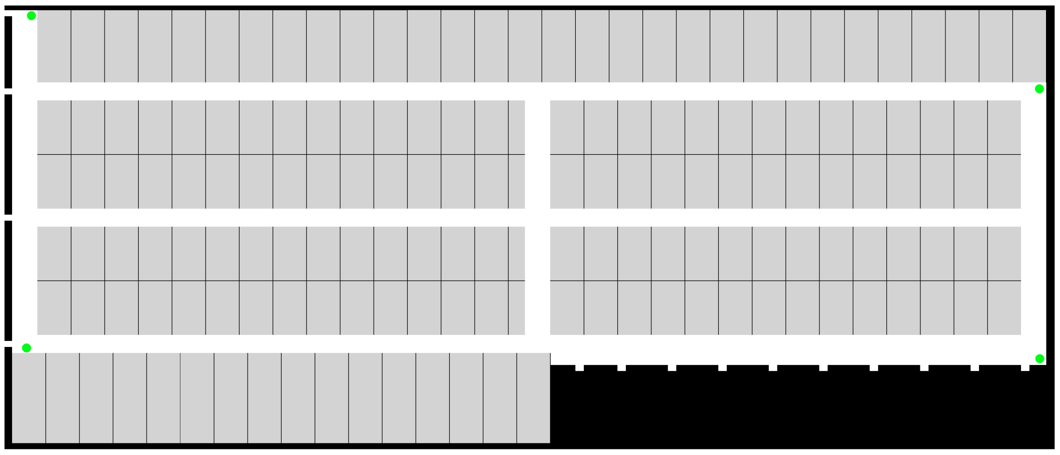


Figure 5.5.: Layout of the basic storage warehouse Slap. The charging stations are indicated in green. The entries are on the left and the exits on the bottom right. The gray areas are made up of storage shelves.

Scenario name	Layout	#AMR	#Tasks	Grid-based
Matrix 1	Matrix	7	18475	yes
Matrix 2	Matrix	10	18475	yes
Matrix 3	Matrix	7	11085	yes
Matrix 4	Matrix	10	11085	yes
Matrix S	Matrix small	6	13195	yes
Cross	Cross-docking	22	58208	yes
KARIS	KARIS	13	17185	yes
Slap	Slap	9	10281	no
Slap grid	Slap	9	10281	yes
Slap grid fewer	Slap	9	7711	yes

Table 5.2.: Overview of the ten different scenarios used

Scheduler		Rule	De Ryck	Trading	Penalty	SET	OR	HGS
Cross	#calls	150686	193229	183754	193390	175873	183272	184119
	t / call	2.0	7.8	15.0	7.8	216.2	397.5	429.3
	t / sec	1.7	8.8	16.0	8.7	220.0	421.6	457.4
Matrix 1	#calls	40043	39803	42499	40602	40078	48713	50602
	t / call	1.7	74.9	57.0	80.3	110.8	100.1	68.4
	t / sec	0.4	17.2	14.0	18.9	25.7	28.2	20.0
Matrix S	#calls	33415	31064	35462	37980	31695	40675	41569
	t / call	0.7	23.6	8.1	36.3	27.9	26.0	19.1
	t / sec	0.1	4.2	1.7	8.0	5.1	6.1	4.6
KARIS	#calls	53451	52730	55738	53686	52580	56900	56525
	t / call	0.1	5.1	23.5	3.8	21.7	345.5	52.3
	t / sec	0.0	1.5	7.6	1.2	6.6	113.8	17.1
Slap	#calls	26532	30368	28326	30354	30324	27302	27873
	t / call	0.6	28.6	4.1	29.0	43.7	61.2	14.9
	t / sec	0.1	5.0	0.7	5.1	7.7	9.7	2.4

Table 5.3.: The runtimes for task assignment for some of the scenarios. Runtimes of all scenarios are displayed in Table A.1 in the Appendix. For each scenario presented, we list the number of times the task assignment algorithm was called, the average time to assign the tasks when called and the average time spent per second of simulation. All times are in milliseconds.

5.2. Technical details

The collision avoidance algorithm is implemented in C++ 14 and compiled with vc14. The simulation and task assignment, as well as the charging strategy are implemented in Python 3.8. We use pybind11 to create Python bindings of the collision avoidance algorithm. The experiments for all schedulers but HGS-CVRP were run on a 6 core Intel i5-9600K computer with a clock speed of 3.7 GHz and 16 GB DDR4 RAM. Due to installation difficulties we were forced to run the HGS-CVRP scheduler on a 6 core AMD Ryzen 5 3600 computer with a clock speed of 3.6 GHz and 16 GB DDR4 RAM. None of the algorithms evaluated use parallelism.

5.3. Experimental results

In this section we evaluate the performance of the algorithms used in this thesis. In Subsection 5.3.1 we present and discuss the runtimes of the task assignment algorithms as well as the collision avoidance algorithm. In Subsection 5.3.2 we evaluate the solution quality of the schedulers for the scenarios introduced in the previous section.

5.3.1. Runtimes

We first evaluate the runtimes of the task assignment algorithms. Table 5.3 shows an excerpt of the runtimes of all algorithms, one for each layout. The full table is in Table A.1 in the Appendix. The #calls include the times when the scheduling algorithm is called but no new assignment of tasks is necessary, as all robots are still busy.

It is noticeable that regardless of the algorithm, the number of calls to the scheduling algorithm in the cross-docking scenario is at least a factor of three of all other scenarios. We assume that this is due to the high number of robots and the high task density in this scenario.

Scheduler		Rule	De Ryck	Trading	Penalty	SET	OR	HGS
Cross	#calls	149717	186511	182963	186626	175669	181937	183270
	t / call	34.1	16.8	17.6	17.2	19.4	26.5	28.9
	t / sec	29.6	18.2	18.6	18.6	19.8	27.9	30.7
Matrix 1	#calls	37167	39666	41854	40222	39892	46867	48706
	t / call	39.9	31.7	34.5	32.4	33.8	34.8	40.9
	t / sec	8.6	7.3	8.3	7.5	7.8	9.4	11.5
Matrix S	#calls	30334	30650	34606	36484	31262	38628	39671
	t / call	8.6	6.2	6.4	7.5	7.2	7.7	8.8
	t / sec	1.5	1.1	1.3	1.6	1.3	1.7	2.0
KARIS	#calls	52933	52730	54873	53686	52580	55289	55946
	t / call	46.5	46.9	55.8	52.7	57.6	70.6	60.8
	t / sec	14.2	14.3	17.7	16.4	17.5	22.6	19.7
Slap	#calls	25805	28048	27255	28191	27607	26410	27032
	t / call	5.7	4.9	4.8	4.9	5.3	5.6	6.5
	t / sec	0.9	0.8	0.8	0.8	0.8	0.9	1.0
Slap grid	#calls	30499	30269	31432	30250	29831	31613	31829
	t / call	12.8	13.0	11.7	10.9	14.5	17.2	12.8
	t / sec	2.3	2.3	2.1	1.9	2.5	3.1	2.4

Table 5.4.: The runtimes for pathfinding and collision avoidance for some of the scenarios. Runtimes of all scenarios are displayed in Table A.2 in the Appendix. For each scenario presented, we list the number of times the algorithm was called, the average time to find conflict-free paths when called and the average time spent per second of simulation. All times are in milliseconds.

As expected due to its simplicity, the runtime of the rule-based algorithm is the best by far in all scenarios with a maximum of 2ms per call in the cross-docking scenario and as little as 0.1ms per call in the KARIS scenario.

The runtime of VRP-based schedulers scales poorly with the size of the scenario, i.e. the number of AMRs. The same is noticeable for SET-MASR as the trading of subsets of tasks converges substantially slower when increasing the number of robots. The scheduler based on OR-Tools performs particularly poorly for the KARIS scenario. It is almost an order of magnitude slower than the other VRP-based scheduler HGS-CVRP for this scenario. Since we have no insight into the inner workings of OR-Tools, we can only make assumptions why this might be the case. One possibility is that the VRP problems generated by the tasks of this scenario are particularly difficult to solve using the heuristics of OR-Tools. HGS-CVRP could either be better equipped to handle these problems, or it simply provides a suboptimal solution more quickly, since we limited the number of iterations without improvements for this algorithm to 2000.

A call to the basic auction scheduler by [4] and the tardiness penalty scheduler take roughly the same time in most scenarios. This is to be expected as they use the same algorithm and only a different cost calculation method. Therefore it is particularly interesting that the penalty algorithm performs worse by a factor of about 1.5 for the small matrix instance. This can be explained by the solution quality of these two schedulers for this scenario. Using the basic auction scheduler, only 8144 of the 13195 tasks are fulfilled while 11977 tasks are fulfilled using the tardiness penalty auction. Thus the task queues of the AMRs are less likely to be full when calling the tardiness penalty scheduler, leading to a longer runtime. This also explains the increase in number of calls for the tardiness penalty scheduler in this scenario.

Next, we evaluate the runtimes of the pathfinding and collision avoidance algorithm. Table 5.4 shows an excerpt of the runtimes for some scenarios, one for each layout and an additional one to compare the difference in runtime between a graph grid-based graph and a graph generated by ODRM. With a runtime of on average at most 70.6 ms per call, our pathfinding and collision avoidance algorithm can be used for real-time applications.

Due to the slim corridor connecting the shelves with the production line, the runtime for the KARIS scenario is the slowest. Due to its size and therefore the size of its graph, the matrix layout is the second slowest. Surprisingly, finding paths and avoiding collisions on the cross-docking layout is not much slower than on the grid-based Slap scenario, even though the number of robots is significantly higher. We attribute this to the fact that the cross-docking layout is more or less split in two as tasks require robots to either move goods from an entry to the sorter or from the sorter to an exit. Therefore robots mostly stay in one half of the layout and do not cause conflicts with the robots in the other half. This hypothesis is supported by the rule-based scheduler, whose runtime per call is twice as high as, for example, the auction-based schedulers. Since it only assigns the closest and in most cases the only available free robot to the next task, the robots do not stay in one half of the layout, leading to more conflicts that need to be resolved.

As the runtime for finding collision-free paths on the graph generated by ODRM is more than twice as fast as on the grid-based graph for all but the HGS-CVRP scheduler, we recommend its use for sufficiently small layouts if runtime is a concern and if previously voiced concerns about more collisions are not relevant.

5.3.2. Solution quality

In this subsection we evaluate the solution quality of the task assignment algorithms. As previously stated, we start evaluating the algorithms after one hour of simulation and stop at 44 hours of simulation unless otherwise mentioned. Since we have ten scenarios and seven assignment algorithms, we limit ourselves to only discussing interesting findings. The median of all box plots in this thesis is marked in orange with the whiskers marking the first and third quartile.

Due to the long runtimes of the schedulers we were unable to run them all multiple times with different robot starting locations and initial battery levels or a slightly modified task distribution. In order to show that the results of the algorithms are largely independent of this, we ran the KARIS scenario three times with different robot start locations and initial battery levels.

5.3.2.1. KARIS layout

Figure 5.6 shows the cumulative tardiness of all schedulers of these three runs. While slight differences are noticeable, the overall performance of the algorithms remains the same.

With a cumulative tardiness that is at least five times as high as the next worst performing task assignment algorithm, the SET-MASR scheduler performs the worst by far. The VRP-based schedulers and the trading auction schedulers perform similarly on all three runs, with OR-Tools performing the worst of all three by the end with a cumulative tardiness of approximately 1×10^5 minutes. Best performing are the rule-based scheduler and the tardiness penalty scheduler. The tardiness of both of these schedulers oscillate a lot. We attribute this to robots being unavailable due to charging, leading to an increase in tardiness. This also explains the higher fluctuation and the lower oscillation frequency for the rule-based scheduler as robots charge longer but less frequently (cf. Appendices A.1, A.2).

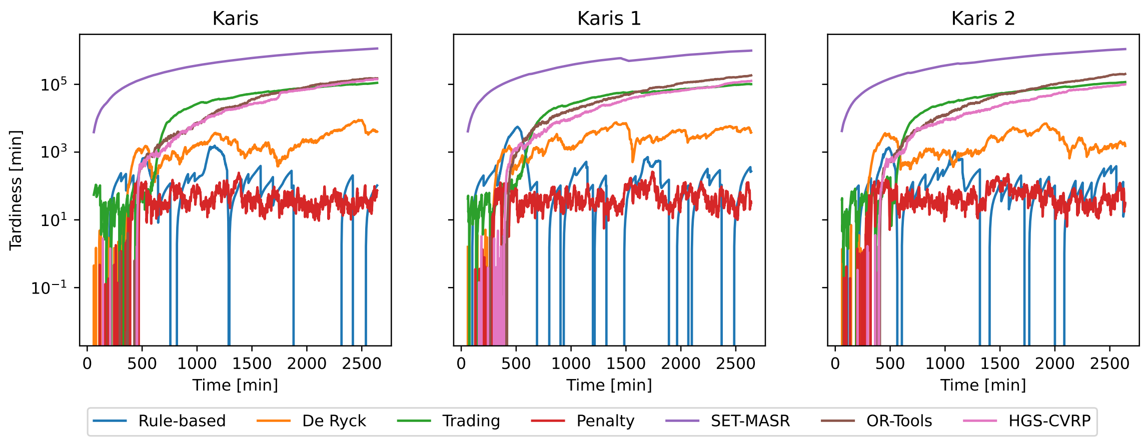


Figure 5.6.: Side-by-side comparison of the cumulative tardiness on a logarithmic scale of all schedulers for the KARIS scenario with modified robot start locations and initial battery levels

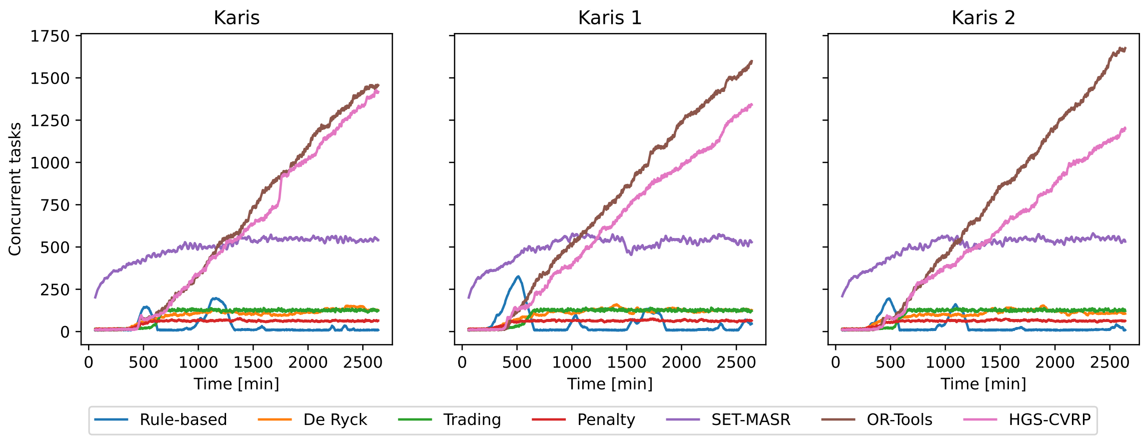


Figure 5.7.: Side-by-side comparison of the concurrent tasks of all schedulers for the KARIS scenario with modified robot start locations and initial battery levels

In Figure 5.7 we plot the number of concurrently open tasks for each scheduler. Similar to the cumulative tardiness, the results from the three runs differ slightly but are overall similar. Interestingly, the SET-MASR does not perform the worst in this metric and instead levels off at around 500 concurrently active tasks. The extremely high cumulative tardiness of the SET-MASR scheduler thus stems from tasks assigned to a robot early on that keep getting delayed. This is not the case for the VRP-based schedulers. Their number of concurrently active tasks keeps increasing and goes as high as 1700 in one of the runs for the OR-Tools scheduler. We therefore hypothesize that the cumulative tardiness of the VRP-based schedulers would eclipse that of the SET-MASR scheduler when running the simulation for a longer time period. The number of concurrently active tasks for the trading auction and the basic auction algorithm levels off at around 100 after 10 hours hours while the tardiness penalty scheduler stabilizes at roughly 50 tasks after approximately 9 hours. The number of concurrently active tasks for the rule-based scheduler stays at around zero for most of the simulation. However, for all three runs the number of active tasks spikes multiple times. The first peak occurs after approximately 500 minutes in all three runs. Differently sized peaks follow at irregular intervals. As mentioned above, we attribute this to the long charging times of the rule-based scheduler.

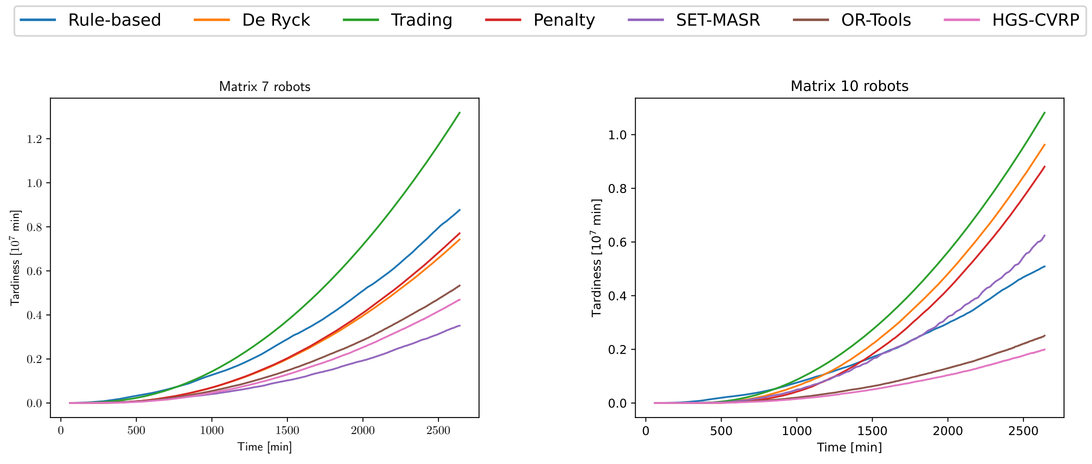
Since the very simple rule-based algorithm performs the best on this layout, we come to the conclusion that the charging strategy we use for the other algorithms is not suitable for this scenario. Therefore, due to its consistent performance, we recommend using the tardiness penalty scheduler for this and similar scenarios. We hypothesize that its performance could be further increased by using a different charging strategy. Owing to their poor performance and runtime, we recommend to avoid using the VRP-based algorithms.

5.3.2.2. Matrix layouts

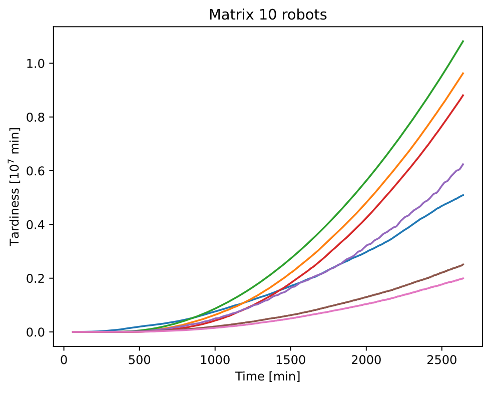
Figure 5.8 shows the cumulative tardiness of the seven schedulers for the four scenarios using the matrix layout. In the two scenarios with 18745 tasks the schedulers are overwhelmed and tardiness increases steadily over time. In both cases the two VRP-based schedulers perform similar to each other and among the best, with the HGS-CVRP scheduler performing slightly better. Their maximum cumulative tardiness decreases by about a factor of one half when increasing the number of robots. However, the performance of the SET-MASR algorithm decreases drastically when increasing the number of robots from seven to ten. Its maximum cumulative tardiness doubles from about 3×10^6 to just under 6×10^6 minutes. We cannot fully explain this decrease in performance but it might in part be due to an increase in conflicts between robots. The percentage of routes rerouted due to conflicts rises from 9.2% (2089 out of 22815) to 10.1% (1841 out of 18172) and the median additional travel time due to a conflict jumps from just over one second to more than 4 seconds. In both scenarios with 18475 tasks the trading auction scheduler performs the worst. We attribute the overall poor performance of this scheduler to the small task queue size of the robots, which is limited to ten for this scheduler and to 50 for the other decentralized schedulers.

Generally, the performance of the centralized schedulers improves drastically when increasing the number of robots while the performance of the decentralized schedulers either improves only slightly or even declines. We once again attribute this to the charging behavior of the schedulers. Therefore, we analyze the charging behavior in the following paragraph.

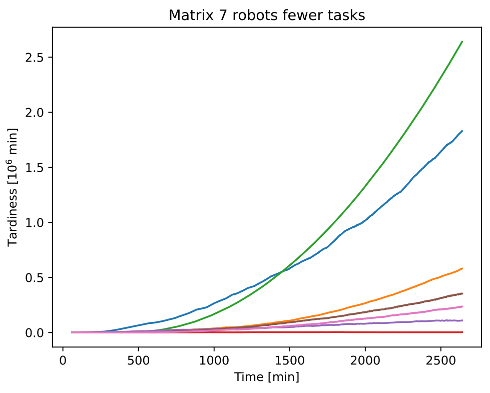
When using the VRP-based schedulers or the trading auction scheduler the robots charge more often and shorter as the maximum queue size of these algorithms is lower (cf. Appendices A.3, A.4). Assume that a robot is charging at a charging station and the schedulers assigns new tasks and therefore re-plans the charging stops. If the route is short, the robot is more likely to be able to finish it with a battery level of 20% or higher.



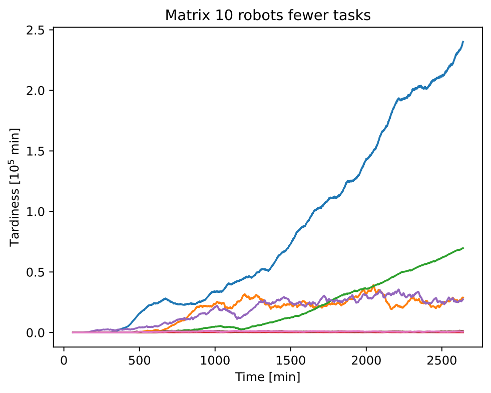
(a) Scenario with seven robots and 18475 tasks



(b) Scenario with ten robots and 18475 tasks



(c) Scenario with 7 robots and 11085 tasks



(d) Scenario with 10 robots and 11085 tasks

Figure 5.8.: Cumulative tardiness of all schedulers for the four scenarios using the matrix layout

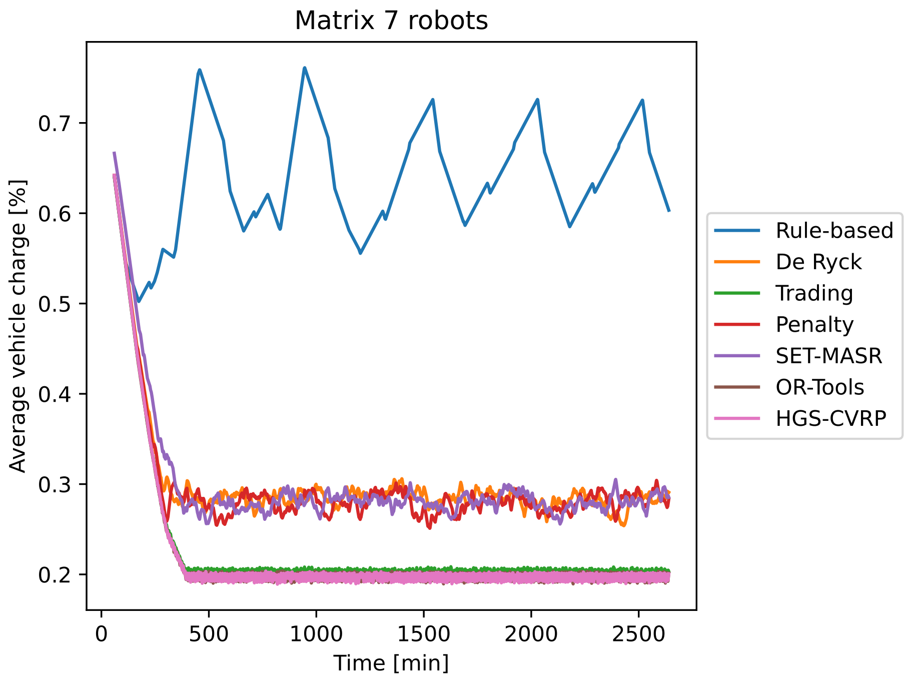
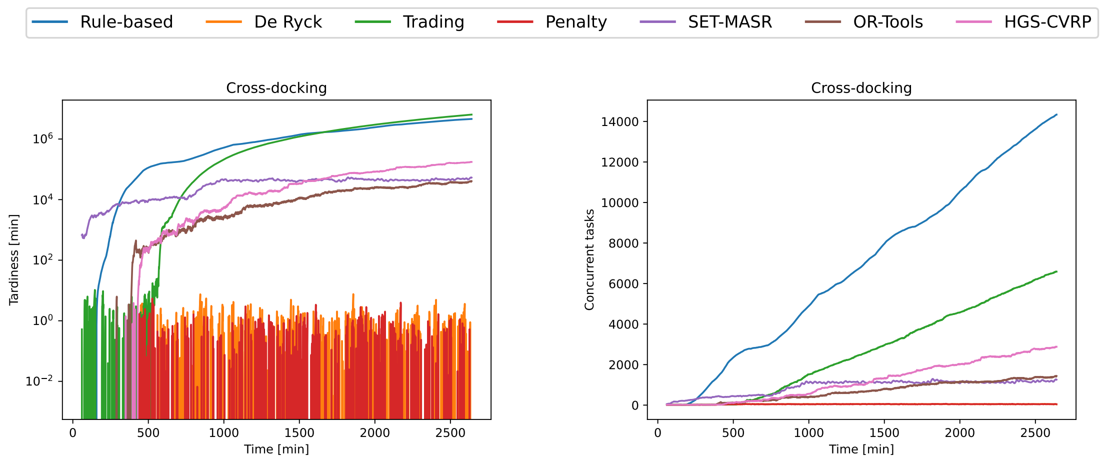


Figure 5.9.: The average cumulative percentage charge per AMR for each scheduler for the matrix scenario with 7 robots and 18475 tasks

Therefore it is more likely to stop charging and start driving right away. However, if the route is longer, the robot will likely have to charge more to finish the route above 20%. Thus it charges at its best charging stop along the route. As it is already at a charging station, it probably stays there longer and charges more. This phenomenon is particularly noticeable in Figure 5.9. It shows the average cumulative percentage charge per robot which oscillates wildly at about 30% for the SET-MASR algorithm, the basic auction algorithm and the tardiness penalty algorithm. The fluctuation of the VRP-based schedulers and the trading auction scheduler is lower and its average vehicle charge is at about 20% for most of the simulation. We attribute the shape of the graph of the rule-based scheduler to its different charging strategy to charge fully once the AMR's battery drops below a threshold.

This combination of longer but fewer charging stops leads to long waiting times at the charging stations when increasing the number of robots while keeping the number of charging stations the same (cf. Appendix A.5). The rule-based scheduler does not suffer from this phenomenon as its robots charge so seldomly that a slight increase in waiting time to charge does not add up. We attribute the poor performance of the trading auction scheduler to its overall poor performance, regardless of the scenario.

This changes when decreasing the number of tasks to 11085. In this case the system is no longer overwhelmed and given the right scheduler, all tasks can be finished in time. This is already the case for the tardiness penalty scheduler using only seven robots. However, the cumulative task tardiness still increases slightly over time for the SET-MASR scheduler, the VRP-based schedulers and the basic auction scheduler but significantly for the other two. When increasing the number of robots to ten, the overall performance of all schedulers improves drastically and the tardiness of the two VRP-based schedulers drops to zero. The trading auction scheduler, which was previously worst, now performs significantly better than the rule-based algorithm but its cumulative tardiness still increases over time.



(a) Cumulative tardiness in minutes on a logarithmic scale

(b) Number of concurrently active tasks

Figure 5.10.: Cumulative tardiness and number of concurrently active tasks of all schedulers using the cross-docking scenario

We therefore conclude that the charging strategy by de Ryck et al. [33] performs better under lower system load and that VRP-based schedulers perform relatively well on this layout, regardless of system load.

The schedulers perform similarly on the small matrix scenario as the matrix scenario with ten robots and 18475 tasks (cf. Appendix A.6). Thus, we refer to the findings and conclusions in the previous paragraphs.

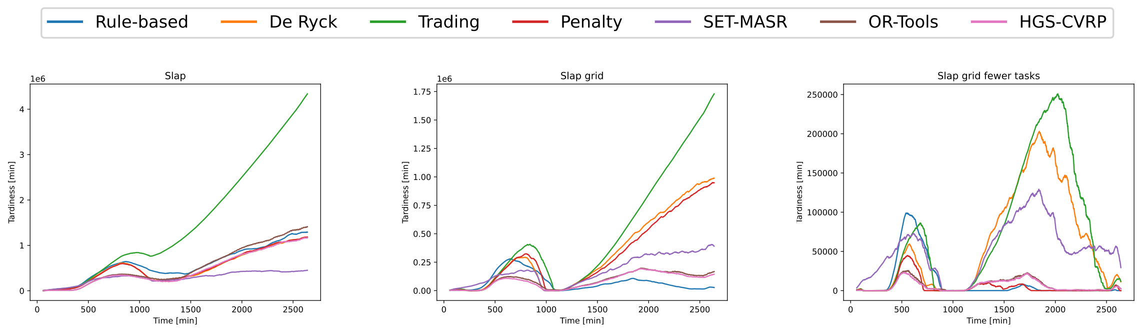
5.3.2.3. Cross-docking layout

Figure 5.10 shows the cumulative tardiness and the concurrent tasks of all schedulers when simulating the cross-docking scenario. Until around 500 minutes, all schedulers but the rule-based one and the SET-MASR scheduler are able to execute all tasks on time or nearly on time. From this point on, robots start having to be recharged and the performance of the trading auction scheduler and the VRP-based schedulers decreases dramatically. Their cumulative tardiness increases by several orders of magnitude in the span of a few hours and continues to grow. This is especially true for the trading auction scheduler, whose cumulative tardiness eclipses that of the rule-based scheduler after about 27 hours. However, its number of concurrent tasks is consistently half as high as that of the rule-based scheduler, which is indicative of tasks that keep getting delayed. Appendix A.7 shows the task age of active tasks after 44 hours of simulation and confirms this claim. The OR-Tools scheduler performs better than the HGS-CVRP scheduler but both their tardiness and their number of concurrently active tasks rise steadily throughout the simulation. The tardiness and the number of concurrent tasks of the SET-MASR scheduler levels off after 16 hours at about 5×10^4 minutes or 833 hours and 1200 tasks respectively. The basic auction scheduler of De Ryck et al. and the tardiness penalty scheduler are capable of performing the tasks mostly on time for the entire duration of the simulation. Therefore we recommend using either one of them for this and similar scenarios.

5.3.2.4. Slap layout

Next, we evaluate the performance of the schedulers for the scenarios using the Slap layout. Figure 5.11 shows the cumulative tardiness of all three scenarios using this scenario.

In all three scenarios and for all schedulers the cumulative tardiness increases rapidly after seven hours due to the large influx of tasks at that time. As no new tasks are added between



(a) Slap scenario using the graph generated by ODRM

(b) Slap scenario using a grid-based graph and 10281 tasks

(c) Slap scenario using a grid-based graph and 7711 tasks

Figure 5.11.: Cumulative tardiness of all schedulers for the three scenarios using the Slap layout

14 and 18 hours, the tardiness levels off again and even drops to zero for the two scenarios using the grid-based graph. The overall performance improves for all schedulers except for SET-MASR when switching from the ODRM graph to the grid-based graph. In the case of SET-MASR, the performance remains the same. This scheduler performs the best out of all schedulers on the scenario using the ODRM graph, hereinafter referred to as the ODRM scenario. In this scenario, the OR-Tools scheduler and the HGS-CVRP scheduler initially perform as well as the SET-MASR scheduler but their tardiness keeps increasing steadily, causing the OR-Tools scheduler to end with the second highest cumulative tardiness after 44 hours. The tardiness penalty auction scheduler and the basic auction scheduler perform the same for the entire duration and end up with a total cumulative tardiness of just under 1×10^6 hours, equaling that of the HGS-CVRP scheduler. The rule-based scheduler consistently performs slightly worse than the previously mentioned auction based schedulers and the trading auction scheduler performs by far the worst, ending with a tardiness of over 4×10^6 minutes.

The three auction-based schedulers perform the worst on the scenario using a grid-based graph and 10281 tasks, followed by the SET-MASR scheduler. The VRP-based schedulers and the rule-based scheduler perform the best with a cumulative tardiness of about 2.7×10^4 minutes after 44 hours. Due to the simplicity of the rule-based scheduler, we attribute this to the employed charging strategy.

When reducing the number of tasks, the performance of the rule-based scheduler and the VRP-based schedulers remain among the best. The VRP-based schedulers are particularly good at working off the large influx of tasks after seven hours while the rule-based scheduler struggles during that time period and has the highest cumulative tardiness. This is due to charging as the robots of the VRP-based schedulers only take short charging stops while some of the robots using the rule-based schedulers have to recharge fully. In addition to the previously mentioned schedulers, the tardiness penalty schedulers also performs well in this scenario. It is worse than the VRP-based schedulers at decreasing the tardiness after seven hours but performs consistently better than them thereafter.

All in all, we recommend using one of the VRP-based schedulers for this layout or the tardiness penalty scheduler for lower system loads. Regardless of the system load, we hypothesize that a different charging strategy than the partial recharging strategy by [33] might lead to better results for this layout and task distributions where tasks are added in irregular spurts. Best suited for these scenarios would likely be a charging strategy that changes its behavior according to the current and expected upcoming task load.

	Rule	De Ryck	Trading	Penalty	SET	OR	HGS
ODRM	16.1	13.9	18.7	14.3	14.3	18.4	17.7
Grid-based	14.0	11.2	15.1	11.2	11.0	16.2	15.2

Table 5.5.: Percentage of routes that are redirected due to conflicts with other robots

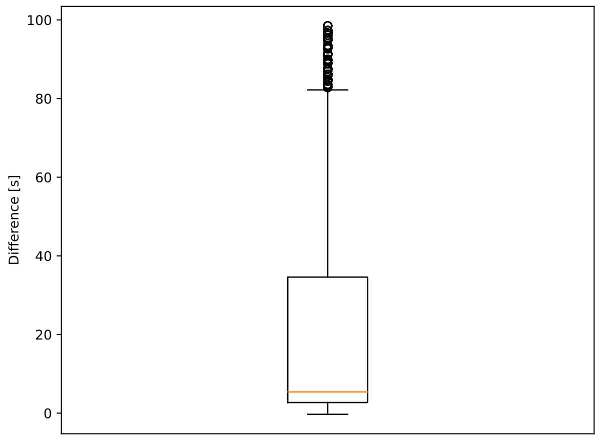


Figure 5.12.: Difference in travel time from pickup location to delivery location for the Slap layout using the grid-based graph and the graph generated by ODRM for the scenarios with 10281 tasks. Positive numbers indicate a shorter travel time of the grid-based graph.

Figure 5.12 shows the difference in travel time from pickup location to delivery location for the Slap scenarios with 10281 tasks. Using the grid-based graph, travel times are consistently lower, with a median of a few seconds but up to a difference of nearly 100 seconds. Longer travel times are acceptable if they lead to fewer potential conflicts and less and shorter detours. However, Table 5.5 shows that the opposite is true: Regardless of the scheduler, the percentage of redirected routes is higher using the ODRM graph than the grid-based graph. Furthermore, the length of the detours stays the same when changing the graph generation algorithm (cf. Appendix A.8). The only advantage of graphs generated by ODRM we noticed is a faster runtime of the pathfinding and collision avoidance algorithm. However, this is only true for smaller layouts. Thus we come to the conclusion that grid-based graphs are superior to graphs generated by ODRM for the EMAPD problem, especially as they can be computed quickly and easily.

6. Conclusion

In this thesis we compared seven different schedulers for the Electric Multi-Agent Pickup and Delivery problem.

Furthermore, we implemented a simulation framework that allowed us to simply and efficiently swap between different schedulers and scenarios.

First, we defined the EMAPD problem and the problems it generalizes in Chapter 2. In the EMAPD problem, AMRs move on a graph with a two-dimensional Euclidean embedding in order to fulfill continuously arriving delivery tasks while avoiding collisions, considering battery constraints and minimizing tardiness. In this chapter, we also gave a brief overview of the current state of the art and presented two algorithms to generate a graph from a given layout.

In Chapter 3, we presented the full and partial recharging strategy, introduced the centralized and decentralized task assignment algorithms and explained our choices. We also presented our pathfinding and collision avoidance algorithm which is a combination of an adapted CCBS algorithm and a modified Cooperative A* search.

We used a discrete-event simulation which we described in Chapter 4 to evaluate the schedulers. In Chapter 5 we first presented the ten scenarios used and then evaluated the schedulers' runtime and solution quality. Overall, the performance of the schedulers varied greatly depending on the system load, the number of robots and the layout. Since the simple rule-based scheduler performed best in some scenarios, we hypothesize that a different charging strategy might lead to better results. The SET-MASR scheduler performed well in scenarios with a high system load where not all of the tasks could be finished. However, it faltered in scenarios with a lower system load. In those scenarios either the VRP-based schedulers or the auction scheduler punishing tardiness performed best. Due to the long runtimes of the former two schedulers we recommend using the latter.

Future Work

Future research could focus on developing and evaluating different charging strategies for the EMAPD problem. Charging strategies that take the system load into account and try to act accordingly seem particularly promising. A reinforcement learning approach could be used to try to make assumptions about upcoming task arrival times, which could allow for better resource allocation.

Another open question is the impact of repositioning idle robots. This is particularly interesting in scenarios with a low task density but high urgency that allow little to no delay.

Finally, since the holistic MAPD solvers produce decent results on large scenarios, adapting them to solve the EMAPD problem seems promising. This would include lifting some of the constraints of these solvers such as requiring unit edge length or supposing that non-task endpoints known in advance exist.

Appendix

A. Figures and tables

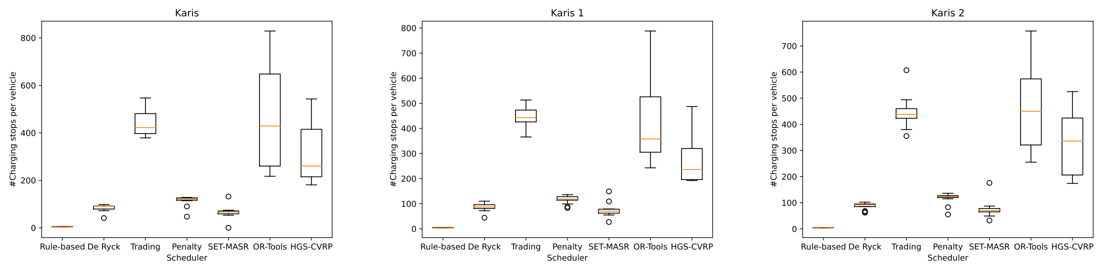


Figure A.1.: Side-by-side comparison of the number of charging stops of all schedulers for the Karis scenario with modified robot start locations and initial battery levels

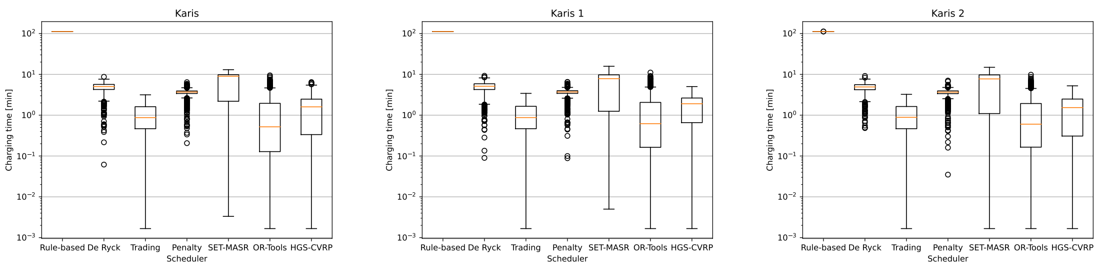
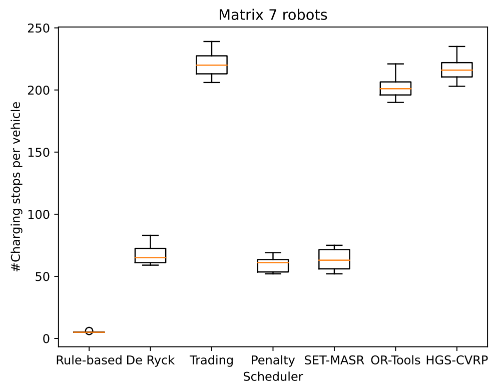
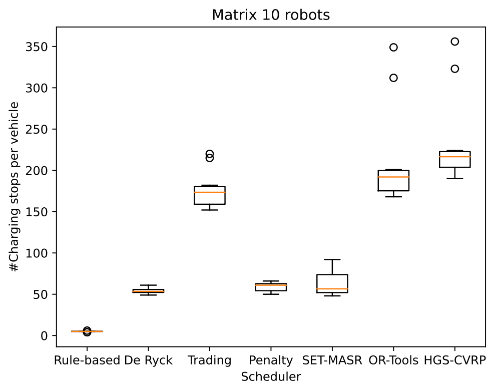


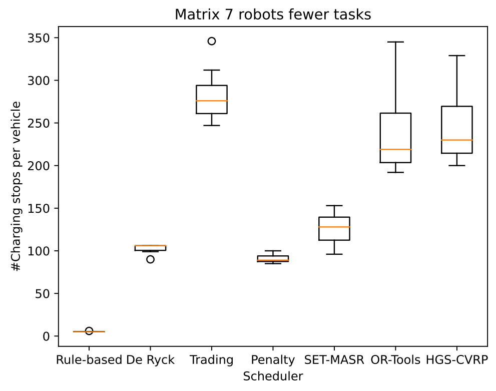
Figure A.2.: Side-by-side comparison of the charging times in minutes on a logarithmic scale of all schedulers for the Karis scenario with modified robot start locations and initial battery levels



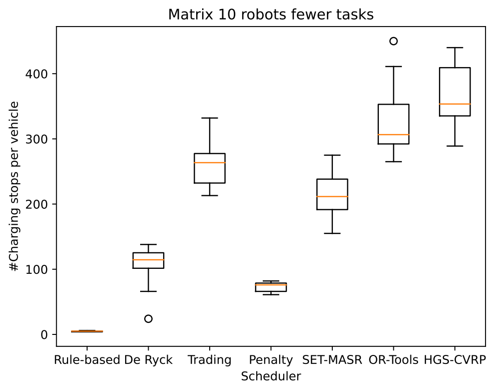
(a) Scenario with seven robots and 18475 tasks



(b) Scenario with ten robots and 18475 tasks

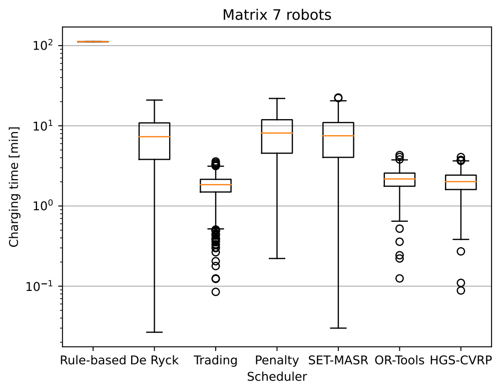


(c) Scenario with 7 robots and 11085 tasks

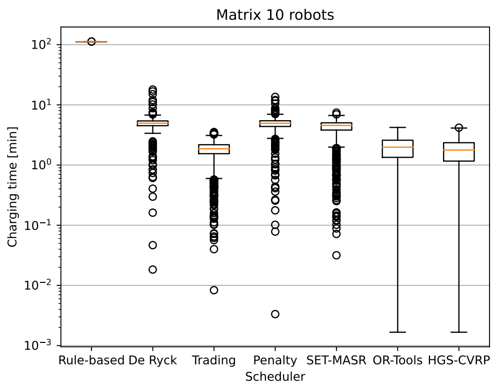


(d) Scenario with 10 robots and 11085 tasks

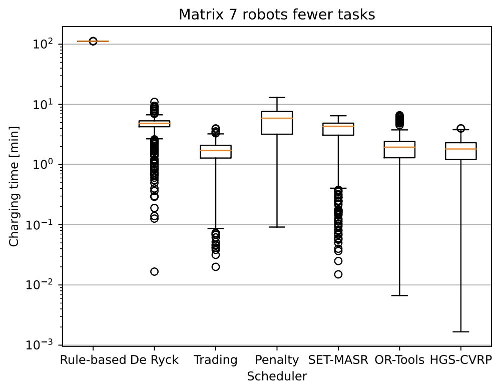
Figure A.3.: Side-by-side comparison of the number of charging stops of all schedulers for the four scenarios using the matrix layout



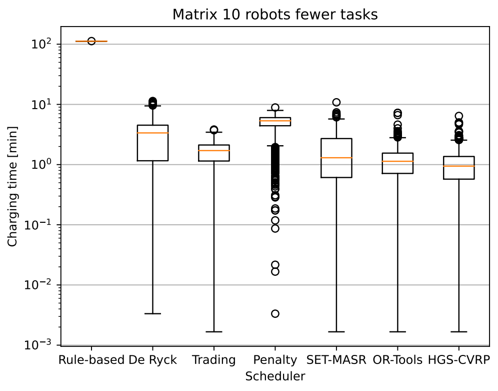
(a) Scenario with seven robots and 18475 tasks



(b) Scenario with ten robots and 18475 tasks

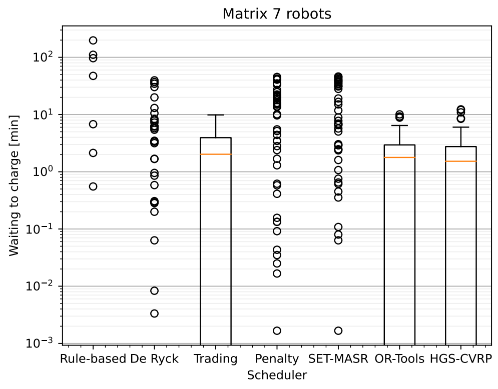


(c) Scenario with 7 robots and 11085 tasks

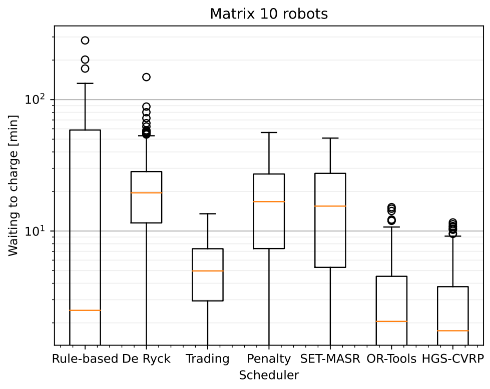


(d) Scenario with 10 robots and 11085 tasks

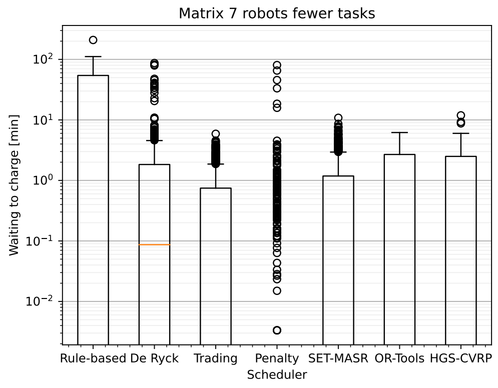
Figure A.4.: Side-by-side comparison of the charging times in minutes on a logarithmic scale of all schedulers for the four scenarios using the matrix layout



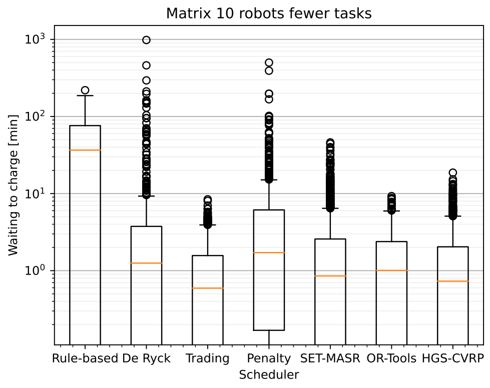
(a) Scenario with seven robots and 18475 tasks



(b) Scenario with ten robots and 18475 tasks



(c) Scenario with 7 robots and 11085 tasks



(d) Scenario with 10 robots and 11085 tasks

Figure A.5.: Side-by-side comparison of the waiting times to charge in minutes on a logarithmic scale of all schedulers for the four scenarios using the matrix layout

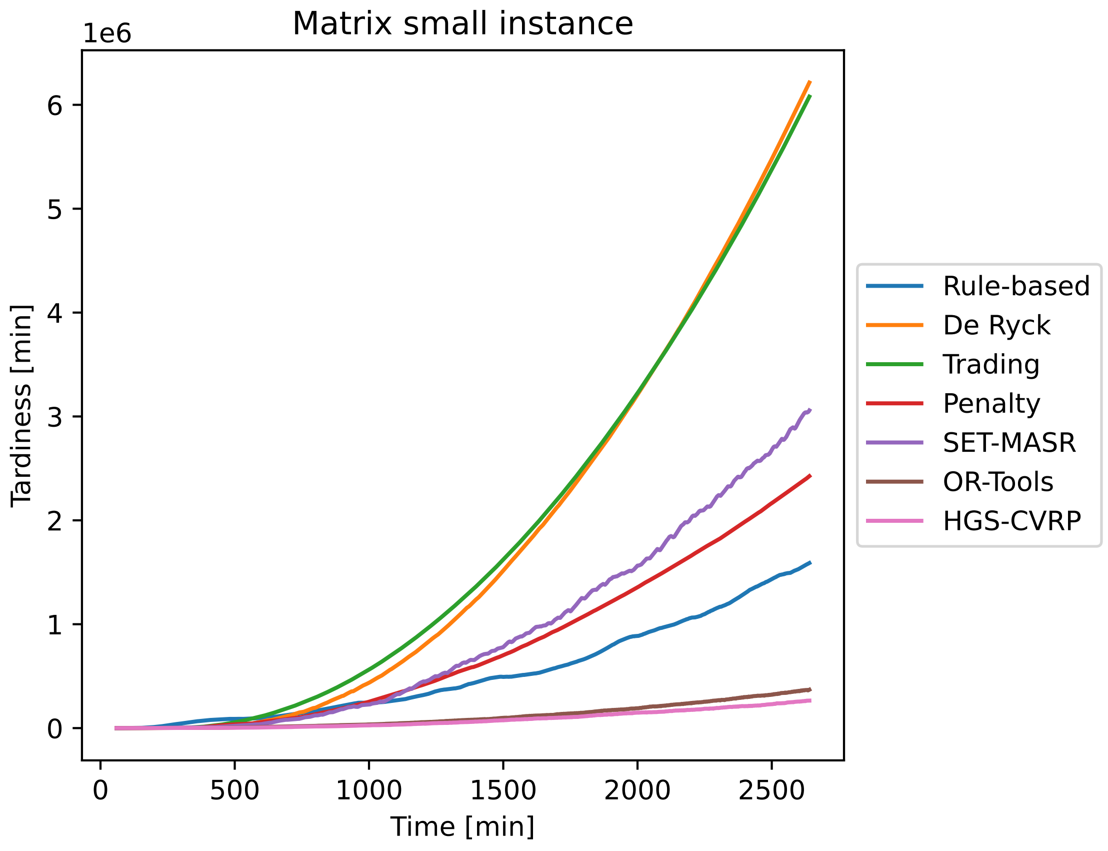


Figure A.6.: Cumulative tardiness of all schedulers using the small matrix scenario

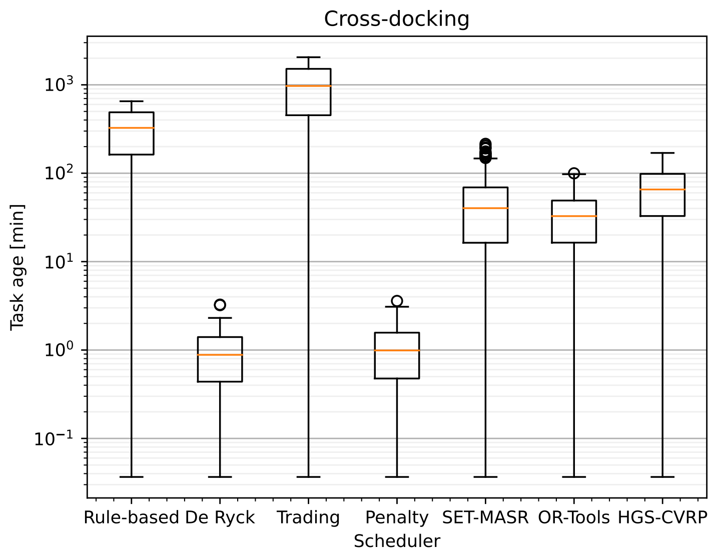
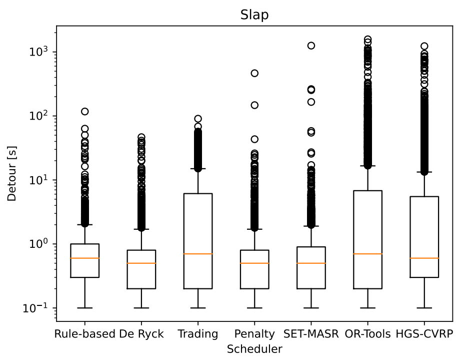
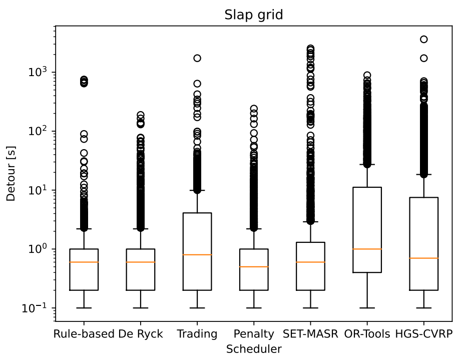


Figure A.7.: Task age of active tasks after 44 hours of simulation for the cross-docking scenario



(a) Slap scenario using the graph generated by ODRM



(b) Slap scenario using the grid-based graph

Figure A.8.: Additional travel time due to conflicts with other AMRs

Scheduler		Rule-based	De Ryck	Trading	Penalty	SET-MASR	OR-Tools	HGS-CVRP
Cross	#calls	150686	193229	183754	193390	175873	183272	184119
	t / call [ms]	2.0	7.8	15.0	7.8	216.2	397.5	429.3
	t / sec [ms]	1.7	8.8	16.0	8.7	220.0	421.6	457.4
Matrix 1	#calls	40043	39803	42499	40602	40078	48713	50602
	t / call [ms]	1.7	74.9	57.0	80.3	110.8	100.1	68.4
	t / sec [ms]	0.4	17.2	14.0	18.9	25.7	28.2	20.0
Matrix 2	#calls	34415	44184	38261	43825	43373	41339	42465
	t / call [ms]	1.9	108.3	15.5	113.5	137.4	42.0	28.7
	t / sec [ms]	0.4	27.7	3.4	28.8	34.5	10.0	7.0
Matrix 3	#calls	32173	34290	36789	34453	36191	38610	39125
	t / call [ms]	0.6	146.4	56.8	44.9	276.3	65.4	99.7
	t / sec [ms]	0.1	29.0	12.1	9.0	57.9	14.6	22.6
Matrix 4	#calls	26598	34090	32640	33878	34452	33481	34348
	t / call [ms]	0.9	86.6	35.9	72.5	116.1	45.6	41.8
	t / sec [ms]	0.1	17.1	6.8	14.2	23.2	8.8	8.3
Matrix S	#calls	33415	31064	35462	37980	31695	40675	41569
	t / call [ms]	0.7	23.6	8.1	36.3	27.9	26.0	19.1
	t / sec [ms]	0.1	4.2	1.7	8.0	5.1	6.1	4.6
KARIS	#calls	53451	52730	55738	53686	52580	56900	56525
	t / call [ms]	0.1	5.1	23.5	3.8	21.7	345.5	52.3
	t / sec [ms]	0.0	1.5	7.6	1.2	6.6	113.8	17.1
Slap	#calls	26532	30368	28326	30354	30324	27302	27873
	t / call [ms]	0.6	28.6	4.1	29.0	43.7	61.2	14.9
	t / sec [ms]	0.1	5.0	0.7	5.1	7.7	9.7	2.4
Slap grid	#calls	31374	31023	32293	31005	30668	32521	32620
	t / call [ms]	0.3	23.8	11.9	23.8	46.4	71.4	25.9
	t / sec [ms]	0.1	4.3	2.2	4.3	8.2	13.4	4.9
Slap grid fewer	#calls	24955	23886	25937	24392	24145	25548	25516
	t / call [ms]	0.2	26.5	18.5	21.6	56.2	75.1	52.9
	t / sec [ms]	0.0	3.7	2.8	3.0	7.8	11.1	7.8

Table A.1.: The runtimes for task assignment for all scenarios. For each scenario presented, we list the number of times the task assignment algorithm was called, the average time to assign the tasks when called and the average time spent per second of simulation.

Scheduler		Rule-based	De Ryck	Trading	Penalty	SET-MASR	OR-Tools	HGS-CVRP
Cross	#calls	149717	186511	182963	186626	175669	181937	183270
	t / call [ms]	34.1	16.8	17.6	17.2	19.4	26.5	28.9
	t / sec [ms]	29.6	18.2	18.6	18.6	19.8	27.9	30.7
Matrix 1	#calls	37167	39666	41854	40222	39892	46867	48706
	t / call [ms]	39.9	31.7	34.5	32.4	33.8	34.8	40.9
	t / sec [ms]	8.6	7.3	8.3	7.5	7.8	9.4	11.5
Matrix 2	#calls	26190	33626	34009	34583	34419	35240	36190
	t / call [ms]	24.1	24.1	22.2	21.6	21.1	22.6	26.8
	t / sec [ms]	3.7	4.7	4.4	4.3	4.2	4.6	5.6
Matrix 3	#calls	30920	34290	35743	34445	36123	37762	38541
	t / call [ms]	39.5	27.6	31.3	30.6	30.4	55.5	64.0
	t / sec [ms]	7.1	5.5	6.5	6.1	6.4	12.1	14.3
Matrix 4	#calls	21986	30421	28680	28677	30865	30003	31165
	t / call [ms]	27.2	22.6	22.1	21.6	20.5	21.7	28.1
	t / sec [ms]	3.5	4.0	3.7	3.6	3.7	3.8	5.1
Matrix S	#calls	30334	30650	34606	36484	31262	38628	39671
	t / call [ms]	8.6	6.2	6.4	7.5	7.2	7.7	8.8
	t / sec [ms]	1.5	1.1	1.3	1.6	1.3	1.7	2.0
KARIS	#calls	52933	52730	54873	53686	52580	55289	55946
	t / call [ms]	46.5	46.9	55.8	52.7	57.6	70.6	60.8
	t / sec [ms]	14.2	14.3	17.7	16.4	17.5	22.6	19.7
Slap	#calls	25805	28048	27255	28191	27607	26410	27032
	t / call [ms]	5.7	4.9	4.8	4.9	5.3	5.6	6.5
	t / sec [ms]	0.9	0.8	0.8	0.8	0.8	0.9	1.0
Slap grid	#calls	30499	30269	31432	30250	29831	31613	31829
	t / call [ms]	12.8	13.0	11.7	10.9	14.5	17.2	12.8
	t / sec [ms]	2.3	2.3	2.1	1.9	2.5	3.1	2.4
Slap grid fewer	#calls	24697	23463	25253	23656	23851	24917	24987
	t / call [ms]	9.7	13.2	12.2	11.5	11.2	16.3	16.8
	t / sec [ms]	1.4	1.8	1.8	1.6	1.5	2.4	2.4

Table A.2.: The runtimes for pathfinding and collision avoidance for all scenarios. For each scenario presented, we list the number of times the algorithm was called, the average time to find the paths when called and the average time spent per second of simulation.

Bibliography

[1] Fortune Business Insights, “Autonomous mobile robots market size, share & covid-19 impact analysis,” https://www.fortunebusinessinsights.com/autonomous-mobile-robots-market-105055, 2022, accessed 07-07-2022.

[2] M. De Ryck, M. Versteyhe, and F. Debrouwere, “Automated guided vehicle systems, state-of-the-art control algorithms and techniques,” *Journal of Manufacturing Systems*, vol. 54, pp. 152–173, 2020.

[3] R. R. Lon and T. Holvoet, “When do agents outperform centralized algorithms?” *Autonomous Agents and Multi-Agent Systems*, vol. 31, no. 6, pp. 1578–1609, 2017.

[4] M. De Ryck, D. Pissoort, T. Holvoet, and E. Demeester, “Decentral task allocation for industrial agv-systems with resource constraints,” *Journal of Manufacturing Systems*, vol. 59, pp. 310–319, 2021.

[5] A. Malus, D. Kozjek *et al.*, “Real-time order dispatching for a fleet of autonomous mobile robots using multi-agent reinforcement learning,” *CIRP annals*, vol. 69, no. 1, pp. 397–400, 2020.

[6] J. Fottner, D. Clauer, F. Hormes, M. Freitag, T. Beinke, L. Overmeyer, S. Gottwald, R. Elbert, T. Sarnow, T. Schmidt *et al.*, “Autonomous systems in intralogistics–state of the art and future research challenges,” *Journal LOGISTICS RESEARCH*, 2021.

[7] D. Giglio, “Task scheduling for multiple forklift agvs in distribution warehouses,” in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014, pp. 1–6.

[8] G. B. Dantzig and J. H. Ramser, “The truck dispatching problem,” *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[9] P. Toth and D. Vigo, *The vehicle routing problem.* SIAM, 2002.

[10] F. Gonçalves, S. R. Cardoso, S. Relvas, and A. Barbosa-Póvoa, “Optimization of a distribution network using electric vehicles: A vrp problem,” in *Proceedings of the IO2011-15 Congresso da associação Portuguesa de Investigação Operacional, Coimbra, Portugal*, 2011, pp. 18–20.

[11] T. Erdelić and T. Carić, “A survey on the electric vehicle routing problem: variants and solution approaches,” *Journal of Advanced Transportation*, vol. 2019, 2019.

[12] M. Asghari and S. M. J. Mirzapour Al-e-hashem, “Green vehicle routing problem: A state-of-the-art review,” *International Journal of Production Economics*, vol. 231, p. 107899, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925527320302607

[13] P. Surynek, “An optimization variant of multi-robot path planning is intractable,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, no. 1, 2010, pp. 1261–1263.

[14] J. Yu and S. M. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[15] J. Li, P. Surynek, A. Felner, H. Ma, T. S. Kumar, and S. Koenig, "Multi-agent path finding for large agents," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 7627–7634.

[16] A. Felner, R. Stern, S. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. Sturtevant, G. Wagner, and P. Surynek, "Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges," in *International Symposium on Combinatorial Search*, vol. 8, no. 1, 2017.

[17] H. Ma, "Graph-based multi-robot path finding and planning," *Current Robotics Reports*, pp. 1–8, 2022.

[18] R. Stern, "Multi-agent path finding–an overview," *Artificial Intelligence*, pp. 96–115, 2019.

[19] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. S. Kumar *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[20] O. Salzman and R. Stern, "Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020, pp. 1711–1715.

[21] G. Fragapane, R. De Koster, F. Sgarbossa, and J. O. Strandhagen, "Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda," *European Journal of Operational Research*, vol. 294, no. 2, pp. 405–426, 2021.

[22] H. Ma, J. Li, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *AAMAS*, 2017.

[23] H. Ma, W. Hönig, T. S. Kumar, N. Ayanian, and S. Koenig, "Lifelong path planning with kinematic constraints for multi-agent pickup and delivery," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 7651–7658.

[24] F. Grenouilleau, W.-J. van Hoeve, and J. N. Hooker, "A multi-label a* algorithm for multi-agent pathfinding," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 181–185.

[25] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 272–11 281.

[26] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, and P. J. Stuckey, "Integrated task assignment and path planning for capacitated multi-agent pickup and delivery," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5816–5823, 2021.

[27] M. Liu, H. Ma, J. Li, and S. Koenig, "Task and path planning for multi-agent pickup and delivery," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2019.

[28] K. Okumura and X. Défago, "Solving simultaneous target assignment and path planning efficiently with time-independent execution," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, 2022, pp. 270–278.

[29] T. Yamauchi, Y. Miyashita, and T. Sugawara, "Standby-based deadlock avoidance method for multi-agent pickup and delivery tasks," in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 2022, pp. 1427–1435.

[30] Y. Fujitani, T. Yamauchi, Y. Miyashita, and T. Sugawara, "Deadlock-free method for multi-agent pickup and delivery problem using priority inheritance with temporary priority," *arXiv preprint arXiv:2205.12504*, 2022.

[31] C. Henkel and M. Toussaint, "Optimized directed roadmap graph for multi-agent path finding using stochastic gradient descent," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 776–783.

[32] B. Delaunay *et al.*, "Sur la sphere vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, no. 793-800, pp. 1–2, 1934.

[33] M. De Ryck, M. Versteyhe, and K. Shariatmadar, "Resource management in decentralized industrial automated guided vehicle systems," *Journal of Manufacturing Systems*, vol. 54, pp. 204–214, 2020.

[34] A. C. L. Queiroz, H. S. Bernardino, A. B. Vieira, and H. J. Barbosa, "Solving multi-agent pickup and delivery problems using a genetic algorithm," in *Brazilian Conference on Intelligent Systems.* Springer, 2020, pp. 140–153.

[35] L. Perron and V. Furnon, "Or-tools," Google. [Online]. Available: https://developers.google.com/optimization/

[36] C. Vidal, "Rei, 2012 vidal t., crainic tg, gendreau m., lahrichi n., rei w," *A hybrid genetic algorithm for multidepot and periodic vehicle routing problems, Oper. Res*, vol. 60, no. 3, pp. 611–624, 2012.

[37] T. Vidal, "Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood," *Computers & Operations Research*, vol. 140, p. 105643, 2022.

[38] A. Viguria, I. Maza, and A. Ollero, "Set: An algorithm for distributed multirobot task allocation with dynamic negotiation based on task subsets," in *Proceedings 2007 IEEE International Conference on Robotics and Automation.* IEEE, 2007, pp. 3339–3344.

[39] A. Andreychuk, K. S. Yakovlev, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," *Artif. Intell.*, vol. 305, p. 103662, 2022.

[40] A. Andreychuk, K. Yakovlev, E. Boyarski, and R. Stern, "Improving continuous-time conflict based search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 220–11 227.

[41] D. Silver, "Cooperative pathfinding," in *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*, vol. 1, no. 1, 2005, pp. 117–122.

[42] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[43] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[44] M. Phillips and M. Likhachev, "Sipp: Safe interval path planning for dynamic environments," in *2011 IEEE International Conference on Robotics and Automation.* IEEE, 2011, pp. 5628–5635.

[45] J. Li, D. Harabor, P. J. Stuckey, A. Felner, H. Ma, and S. Koenig, "Disjoint splitting for multi-agent path finding with conflict-based search," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 279–283.

[46] J.-C. Latombe, "Motion planning: A journey of robots, molecules, digital actors, and other artifacts," *The International Journal of Robotics Research*, vol. 18, no. 11, pp. 1119–1128, 1999.

[47] K. D. Glock, "Emergency rapid mapping with drones: models and solution approaches for offline and online mission planning," Ph.D. dissertation, Karlsruher Institut für Technologie (KIT), 2020.

[48] "Linde li-ion 48v," https://www.linde-mh.com/media/Global-Content/08_Downloads/Linde-Li_ION-48V-EN.pdf, accessed 24-07-2022.

[49] M. Disselnmeyer, "Approximate dynamic programming for dispatching autonomous mobile robots in intralogistics," 2021.

[50] Q. S. Kabir and Y. Suzuki, "Comparative analysis of different routing heuristics for the battery management of automated guided vehicles," *International Journal of Production Research*, vol. 57, no. 2, pp. 624–641, 2019.

[51] Bundesministerium für Bildung und Forschung, "Karis pro – autonomer materialtransport für flexible intralogistik," http://karispro.de/Abschlussbericht%20KARIS%20PRO.pdf, accessed 07-07-2022.