Master of Science Thesis

Multi-Agent Pickup-and-Delivery in a Distributed Baggage Handling System

W. Duchateau

Technische Universiteit Delft

TU Delft

Multi-Agent Pickup-and-Delivery in a Distributed Baggage Handling System

Master of Science Thesis

by

W. Duchateau

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on March 2nd, 2021 at 09:30.

Student number:	4369351	
Project duration:	11/2019 – 03/2021	
Thesis committee:	Dr. O.A. Sharpanskykh	TU Delft, Supervisor
	Dr. ir. M. Snellen	TU Delft, Chair
	Dr. ir. E. van Kampen	TU Delft, External Examiner

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Cover image: Vanderlande

TU Delft

Acknowledgements

Dear reader,

This report concludes my thesis research in order to obtain a Master of Science degree in Aerospace Engineering at the Delft University of Technology. The main part of this document, the scientific paper, presents a hybrid approach to provide coordination for a fleet of robots in a distributed baggage handling system. The proposed solution combines optimal conflict-free path finding and optimally assigning tasks to robots, and can be extended to other warehousing operations. The remainder of this report serves as supporting information to the paper. I would like to take this opportunity to thank all the people that have supported me over the last years.

First of all, I want to express my sincere gratitude to Dr. Alexei Sharpanskykh, my supervisor, who came up with the idea of my research subject and introduced me to agent-based modelling in the first place. He allowed me the freedom to fill in the research myself and, during our meetings, I could always count on his constructive and valuable criticism. His support and motivation have played a big role in bringing this thesis successfully to an end and since the subject was a perfect fit, it has been a great experience from the start.

Secondly, I want to thank all my friends for their support, for providing the necessary distractions during the past year and, off course, for making my years in Delft so wonderful. Thank you to Sam for the memorable experiences, for disconnecting me from my work and the support during our shared thesis period. Furthermore, I want to especially thank Jacoba, my girlfriend, for supporting me throughout my studies, undergoing my complaining from time to time and never failing to cheer me up.

Finally, I cannot express my immense gratitude towards my parents to give me the possibility of studying at the TU Delft and to always push me that extra bit further. Also a big thanks to my brother, Michiel, for his entertainment when I would be at home. They made this degree possible so it is a shame we cannot physically celebrate this special moment due to the pandemic. Luckily, the pandemic means that I still have to see a lot of people again and therefore have a reason to return to Delft.

Thank you all!

Wouter Duchateau
Delft, March 2021

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

A	Vargha-Delaney A-test value
ABM	Agent-based model
AGV	Automated guided vehicle
AMAPF	Anonymous MAPF
BDI	Believe-Desire-Intention
BHS	Baggage handling system
BP	Bypass
BPEA*	Basic PEA* (i.e. PEA*)
CA	Context-aware routing
CBM	Conflict-based min-cost-flow
CBS	Conflict-based search
CNP	Contract Net Protocol
CT	Constraint tree
DAO	Dragon Age: Origins
DiMPP	Improved DMAPP
DMAPP	Distributed MAPP
DMAS	Delegate MAS
ECBS	Enhanced CBS
ECBS-TA	ECBS with task assignment
EPEA*	Enhanced PEA*
FF	Fast plan generation through heuristic search
HWY	Highway
IA	Instantaneous assignment
ICTS	Increasing cost tree search
ID	Independence detection
IDMB	Improved distributed market-based

iECBS	Improved ECBS
isMA-RRT	Informed sampling MA-RRT
JA	Joint-state space
KPI	Key performance indicator
LCR	Le Lann, Chang and Roberts
MA	Multi-agent
MA-A*	Multi-agent A*
MA-CBS	Meta-agent CBS
MA-RRT	Multi-agent RRT
MAP	Multi-agent planning
MAPD	Multi-agent pickup-and-delivery
MAPF	Multi-agent path finding
MAPF+TA	Separate path finding and task assignment
MAPP	Multi-agent path planning
MAS	Multi-agent system
MR	Merge and restart
MR	Multi-robot
MRTA	Multi-robot task allocation
MT	Multi-task
NP	Nondeterministic polynomial time
OA	Optimal anytime
OD	Operator decomposition
PC	Prioritizing conflict
PEA*	Partial expansion A*
PI	Performance indicator
RRT	Rapidly exploring random trees
SIPPwRT	Safe interval path planning with reservation table
SR	Single-robot
ST	Single-task
TA	Task assignment
TA	Time-extended assignment

TAPF	Task assignment and path finding
TP	Token passing
TPTS	Token passing with task swapping
WP	Work package

Symbols

ΔT	Delay
η	Efficiency
μ	Mean
σ	Standard variance
$LB(n)$	Lowest cost node (unexpanded) within the constraint tree
r	Cost threshold
t_{coord}	Runtime required to provide coordination
$t_{ecbs-ta}$	Runtime required by ECBS-TA
t_{ecbs}	Runtime required by ECBS
t_{run}	Runtime required for a time step
$t_{service}$	Bag service time
t_{wait}	Bag waiting time
w_{ecbs}	ECBS suboptimality factor
w_{hwy}	Highway factor

Introduction

The research that is presented in this document is situated in baggage handling in airports. Generally, conventional conveyor systems are over-designed and only need to operate at full capacity for a couple of months during their lifetime of averagely 15 years. Because of this, flexible systems are being proposed in which robots transport luggage throughout the airport. These solutions allow robots to be taken out or be introduced to attend to varying capacity demands.

In such flexible systems, the robots optimally get assigned bags and they are required to find optimal, conflict-free paths towards these bags and subsequently to their destination. These two processes, task assignment and path finding, are generally optimized separately. Aiming to find the optimal combination of both can then lead to suboptimal solutions or to very extensive computations. Algorithms that combine task assignment and path finding do exist though. However, state-of-the-art solutions do not let the robots attend to a stream of tasks as the problem stops when each robot has reached its destination. On the other hand, state-of-the-art solutions that do let robots attend to a continuous stream of tasks, i.e. multi-agent pickup-and-delivery (MAPD), do not combine task assignment and path finding. In this research we therefore propose a hybrid MAPD algorithm that combines path finding and task assignment, based on existing algorithms.

This research was carried out at the Air Transport Operations department[1] under supervision of Dr. Alexei Sharpanskykh.

This thesis report is organized as follows : Part 1 contains the main part of this thesis project, the scientific paper. Relevant literature, supporting the research, is provided in Part II. Finally, Part III contains additional results and explanations.

[1] Aerospace Engineering faculty, Delft University of Technology

I

Scientific Paper

Multi-Agent Pickup-And-Delivery in a Distributed Baggage Handling System

W. Duchateau*, O.A. Sharpanskykh†

Delft University of Technology, Delft, The Netherlands

Abstract

Baggage handling systems in airports are generally over-designed and only need to work at full capacity for a couple of months during their lifetime. To cope with varying demand, flexible baggage handling systems are being developed which use Automated Guided Vehicles (AGV) to handle a required capacity. These AGVs transport bags throughout the baggage hall by optimally getting assigned bags to pick up and finding optimal, conflict-free paths to their destinations. Generally, in such systems or other warehousing applications, these two processes, task assignment and path finding, are optimized separately. This research therefore proposes a hybrid online Multi-Agent Pickup-and-Delivery (MAPD) algorithm that jointly optimizes both. To achieve this, two existing algorithms have been combined. First, the Improved Distributed Market-Based algorithm (IDMB) is used to provide optimal task assignment by letting AGVs communicate with each other. Second, to provide coordination, we implemented Enhanced Conflict-Based Search with Task Assignment (ECBS-TA). ECBS-TA, however, only lets AGVs attend to one task whereafter the algorithm stops. We therefore modified it into a lifelong version, letting the AGVs continuously move between pickup and drop-off points, to function as an MAPD algorithm. A Multi-Agent System (MAS) has been created and implemented in a conceptual baggage hall using highways to guide the AGVs. To analyse the hybrid MAPD algorithm, it has been compared to two other algorithms in which task assignment and path finding are performed separately. The analysis indicated that the hybrid MAPD solution was able to deliver better operational capacity at the expense of scalability. Its runtimes were found to be higher, but overall still ensured real-time applicability.

1 Introduction

Baggage handling systems (BHS) in airports are generally designed to last for 15 years. Considering that, according to Gittens [2019], the demand for passenger traffic is growing at a rate of 3.7% per year, such a system should be designed so that it can handle an increase of almost twice the initial demand during its lifetime. This implies that conventional baggage handling systems are dealing with a surplus capacity for 14 years since it is only fully operational in the 15^{th} year, disregarding peak periods. A very large system is being created to only work at its full capacity for a couple of months [Kersten, 2019].

*MSc Student, Air Transport and Operations, Faculty of Aerospace Engineering, Delft University of Technology

†Assistant Professor, Air Transport and Operations, Faculty of Aerospace Engineering, Delft University of Technology

To cope with this varying demand, Vanderlande [2019a] is developing FLEET, a flexible distributed baggage handling system in which automated guided vehicles (AGV) handle a required capacity. Peak demands can be dealt with by simply adding robots to the system after which they can be retracted and be put in use elsewhere [Kersten, 2019]. Similar systems have been existing for quite some time, think about Amazon warehousing (previously Kiva) [Wurman et al., 2008] where robots transport shelves to workers who then sort packages for shipment. FLEET deals with a highly dynamic system in which tasks appear randomly and that is capable of handling varying capacity demands. The robots that carry the bags throughout the airport are tasked with finding their most optimal route, their best task assignment, monitoring their battery level and taking care of separation with other robots. All of this in a distributed way while aiming for flexibility and scalability.

Finding routes and allocating tasks are generally two separate processes. When aiming to find the optimal solution, this can result in highly suboptimal solutions or lead to very extensive computations. Because of this, Ma and Koenig [2016] and Hönig et al. [2018], presented combined Task Assignment and Path Finding (TAPF) algorithms. These algorithms successfully strived to jointly optimize both processes. One issue remained however: a TAPF problem considers n robots that can attend to n tasks and the problem stops when each robot has reached its target. In automated warehouse logistics, or automated baggage handling systems, robots cannot stop working after delivering a task, they have to attend to a continuous stream of tasks. This is what the Multi-Agent Pickup-and-Delivery (MAPD) problem captures. The MAPD problem was first introduced by Ma et al. [2017] and considered *online* task assignment in which tasks appear randomly and only become available when they have entered the system. Although their proposed MAPD approach provided a plausible solution for an automated baggage handling system, this solution was not necessarily optimal since task assignment and path finding were solved separately. Besides online MAPD algorithms, *offline* solutions have been proposed by Liu et al. [2019]. In offline task assignment, robots can plan beforehand which tasks they will take on and subsequently optimize their paths with respect to all those tasks. The advantage of offline MAPD algorithms is that it provides significantly better solutions when compared to the online algorithms from Ma et al. [2017] and Ma et al. [2019]. However, in a real-time application where bags appear randomly, planning a sequence of tasks beforehand is not an option. Because of this, in combination with the lack of combined task assignment and path finding in state-of-the-art online MAPD solutions, this Master thesis addresses the following main research question:

How can a distributed baggage handling system be controlled in a flexible and scalable way by means of a multi-agent pickup-and-delivery algorithm which combines task assignment and lifelong path finding while considering AGVs' battery life?

In this paper, we propose a hybrid solution for MAPD problems by creating a lifelong TAPF algorithm based on the existing ECBS-TA mechanism by Hönig et al. [2018]. ECBS-TA has been selected because of its performance in runtime, success rate and scalability [Duchateau, 2020]. It uses Enhanced Conflict-Based Search (ECBS) [Barer et al., 2014] as Multi-agent Path Finding (MAPF) algorithm and the centralized Hungarian algorithm as task allocation method. As the research question indicates, we aimed to create a distributed algorithm to remove single-points of failure in the BHS and improve flexibility. We therefore propose to, instead of the centralized Hungarian method, use the Improved Distributed Market-Based (IDMB) algorithm. This algorithm was introduced by Trigui et al. [2014] and has proven to be able to approximate the optimal Hungarian algorithm within 3%. The environment in which we implemented the hybrid MAPD solution is a conceptual baggage hall by Vanderlande [2019b]. Furthermore, highways are implemented. These introduce

rules for controlling AGV flows and ease conflict resolution for ECBS(-TA). The proposed hybrid solution was compared to two mechanisms that solve task assignment and path finding separately (MAPF+TA). Both use ECBS for path finding but differ in task allocation mechanism: the first uses a simple auction (Auction+ECBS) and the second uses the IDMB algorithm (IDMB+ECBS). The three algorithms were evaluated based on real-time applicability, measuring runtimes and success rates, and operational capacity which was measured by service times and throughput. Moreover, the performance of the system and the AGVs was analysed when charging was included. The results showed that our approach was able to achieve higher throughputs and lower service times at the expense of scalability compared to the two MAPF+TA algorithms. Despite worse scalability, runtimes were on average kept sufficiently below 1 second to guarantee real-time applicability in a system with 60 AGVs. Further research should be performed to prove whether other warehousing environments (e.g. Kiva-like environments) would allow for improved scalability.

The structure of this paper is as follows: Section 2 first introduces the algorithms that have been combined in our hybrid MAPD solution. Secondly, the agent-based model (ABM) is presented in section 3 which provides a description of the baggage hall and the multi-agent system (MAS). Hereafter, section 4 elaborates on the experiments that were carried out to assess our algorithm along with their results. A discussion on our model and the results is provided in section 5. Finally, conclusions are drawn in section 6.

2 The hybrid MAPD mechanisms

The methodological approach consisted of two different algorithms: an algorithm that provided optimal task allocation, IDMB, and an algorithm to solve the TAPF problem, ECBS-TA. The second algorithm is an extension of ECBS, an existing MAPF solution. The implementation consisted therefore of three phases, as visualized in fig. 1, that ultimately lead to the distributed Multi-Agent Pickup-and-Delivery solution.

In section 2.1, we present ECBS, the path finding part of ECBS-TA. ECBS is a suboptimal solver for the MAPF problem which allows AGVs to safely move through the baggage hall by providing them with conflict-free paths. Secondly, IDMB is explained in section 2.2. It is a distributed auction-based method that allows agents to communicate to find the optimal task assignment. Hereafter, in section 2.3, we present the ECBS-TA algorithm and how we incorporated IDMB into it. Finally, to allow for lifelong path finding, meaning that AGVs continuously move back and forth between pickup and drop-off points, ECBS-TA was modified into our hybrid MAPD solution. Section 2.4 elaborates upon this final step.

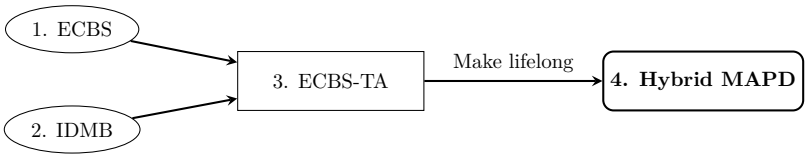


Figure 1: Scheme showing the implementation phases, resulting in the hybrid MAPD solution.

2.1 MAPF: Enhanced Conflict-Based Search

ECBS is based on Conflict-Based Search (CBS) [Sharon et al., 2015], an algorithm that provides and guarantees optimal solutions for MAPF problems by means of a constraint tree. CBS is a two-level algorithm that identifies conflicts between agents and resolves them by imposing constraints on each involved agent. The high level of CBS starts off with creating a root node which contains the shortest path for each agent, not taking into account any conflicts. It validates the paths and resolves a conflict between agents by generating child nodes, one for each involved agent, in which it adds a constraint that prohibits an agent to be at the location of conflict at the corresponding time point. The low level then uses an individual path planning algorithm, A* in our case, to identify new paths for the agents in conflict, taking into account the imposed constraints. This process is repeated until a conflict-free node is found.

As mentioned, CBS is an optimal solver for the MAPF problem. However, optimally solving MAPF problems has been proven to be NP-hard [LaValle and Jingjin, 2013], indicating that a problem might not be solvable in polynomial time. When the number of involved agents increases, the state space grows exponentially and makes optimal solvers like CBS not scalable. To allow for flexibility in the solution, bounded-suboptimal solvers were proposed. One of the most promising ones being Enhanced CBS (ECBS).

ECBS Enhanced CBS [Barer et al., 2014] is a bounded-suboptimal variant of CBS that exchanges some of its optimality for better performance in both runtimes and success rate. By not only focussing on optimal solution cost, but as well taking into account the amount of conflicts that agents are involved in, ECBS allows to find suboptimal solutions for which the cost lies within a user-defined suboptimality factor w_{ecbs}, i.e. *focal search*. Instead of validating the lowest cost node during the high-level search, focal search uses a second heuristic: the least amount of conflicts. All nodes for which the cost lies within the suboptimality bound are considered for expansion but, from those, the one with the least amount of conflicts is validated by the high level.

Windowed search To speed up ECBS, the user can define a conflict window in which conflicts are resolved. This conflict window is reduced when finding a solution takes too long. Using a conflict window means that ECBS has to be run frequently to resolve conflicts that occur after the defined time horizon.

Highways The performance of ECBS can be further enhanced by adding highways to the system, ECBS+HWY. These are pre-defined rules for managing agents and have the means to create streams of AGVs and so to reduce the amount of conflicts the coordination mechanism has to deal with. Highways are created by reducing the cost of edges to encourage AGVs to follow them. The user can define a highway factor w_{hwy} with which the cost of edges are lowered to define the level of encouragement [Cohen et al., 2015].

2.2 Task allocation: the Improved Distributed Market-Based algorithm

To implement task allocation in a distributed manner, we used the Improved Distributed Market-Based algorithm [Trigui et al., 2014]. Originally, the centralized Hungarian algorithm was used in ECBS-TA but ECBS-TA is not limited to using centralized algorithms only [Hönig et al., 2018]. Moreover, IDMB had shown to approximate the optimal Hungarian algorithm within 3%.

Figure 2 shows the three steps which the IDMB algorithm undertakes to obtain a (near-)optimal task assignment. It is based on negotiation between AGVs to obtain the optimal solution.

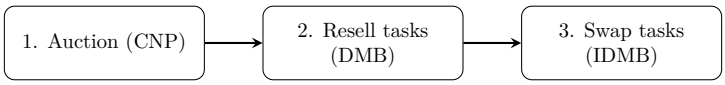


Figure 2: IDMB process overview

IDMB starts off with (1) Contract Net Protocol (CNP), which allows for competitive negotiation between agents [Khamis et al., 2015]. This means that, first, agents (AGVs) can make themselves available for tasks. Secondly, the auctioneer (infeed stations) can broadcast these tasks to the available AGVs and request specific information from them (bids). The auctioneer then selects the best bid and finally assigns the task to the winning agent (i.e. assigning the contract). In practice, this means that bags are auctioned by their corresponding infeeds to the closest available AGV.

Since IDMB allows AGVs to win multiple bids (i.e. bags) during one auction round, (2) an AGV with multiple bags assigned selects the task that suits it best, i.e. the closest one, and resells the remaining ones. At this point, each AGV has acted out of self-interest, not leading to an optimal task allocation. Therefore, (3) the AGVs that are assigned to a task communicate with each other to determine whether or not swapping tasks would benefit the cost of the solution.

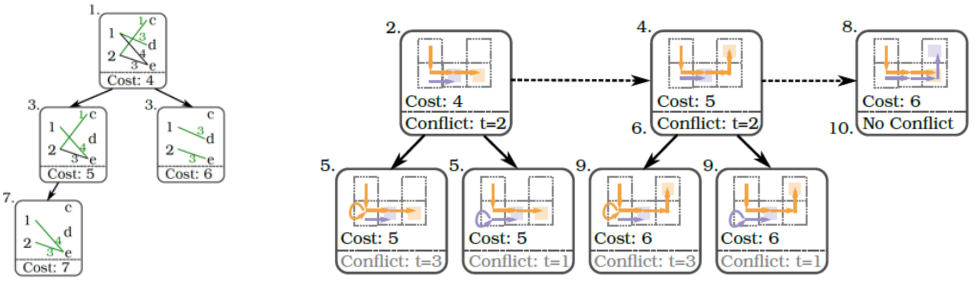
2.3 ECBS with Task Assignment

Motivated by warehouse automation, where robots were used to transport shelves to manned packaging stations [Wurman et al., 2008], Hönig et al. [2018] wanted to minimize the idle time of these human workers. They created an extension of the CBS framework to solve TAPF problems by minimizing solution cost: CBS with Task Assignment (CBS-TA). The key idea of CBS-TA is to create a search forest (fig. 3(b)) in which multiple constraint trees are expanded simultaneously and can be added on demand. The framework of CBS allowed the authors to extend their CBS-TA algorithm to a bounded and suboptimal version, ECBS-TA, which we adopted in our MAPD approach.

Root expansion The user of the ECBS-TA algorithm can decide how often a new tree is added. There are three ways to do so: (1) As few as possible, 'MinRoot', (2) regularly, 'CBS-TA Style' (i.e. whenever the expanded node is a root node), and (3) as many as possible, 'MaxRoot'. For scalability reasons, we chose to continue with the **MinRoot** approach. In this approach, each new root node defines a threshold, i.e. minimum node cost, at which the next constraint tree should be added: $r = w \cdot LB(n)$, w is the user-defined suboptimality factor for ECBS, while $LB(n)$ is the lowest cost node in the current constraint tree(s). When, in the constraint tree(s), the cost of the lowest cost node exceeds the defined threshold, a new root node (and constraint tree) is created by invoking the next-best task assignment. Alternatively, in the MaxRoot approach, all additional root nodes (within the suboptimality bound) are added for which the cost is smaller than or equal to $r = w \cdot LB(n)$.

Next-best task assignment The next-best task assignment mechanism also makes use of a constraint tree to find the next-best assignment, an example is shown below in fig. 3(a). In this constraint tree, the root node consist of the task allocation provided by IDMB. A next-best task allocation is found by creating child nodes which contain (1) assignments that must be part of the solution and (2) assignments that cannot be part of the solution. In each child node, one of the assignments of the parent node is disallowed and re-auctioned. To provide ECBS-TA with optimal next-best assignments, IDMB is partly invoked within the created child nodes: the AGV that won the repeated auction can ask the other AGVs in the node to swap tasks.

Example Figure 3 provides an example of the working principle of CBS-TA. The process starts with (1) finding the optimal task assignment in fig. 3(a). In this example, robots 1 and 2 are respectively assigned to tasks d and c. For this assignment, (2) the first CBS root node is created and expanded as shown in fig. 3(b). Because the expanded node was a root node, (3) next-best task assignment is performed resulting in two new optional task assignments. For the next-best assignment $\{1 \mapsto e,\ 2 \mapsto c\}$, (4) a new root node is created. CBS then starts with (5) expanding the lowest cost nodes and reaches the second root node while doing so. (6-8) The second root node is expanded, the next-best task assignment is therefore calculated and a new root node is added for $\{1 \mapsto d,\ 2 \mapsto e\}$. (9) CBS continues with expanding nodes (including those from step 5) and (10) eventually finds a conflict-free solution.



(a) Next-best task assignment constraint tree

(b) Search forest

Figure 3: The working principle of CBS-TA for 2 tasks and 3 agents [Hönig et al., 2018].

The working principle of ECBS-TA does not differ much from that of CBS-TA. The first difference is that nodes are expanded using the aforementioned focal search. The second one is that CBS-TA always adds a new root node when another root node has been expanded (CBS-TA Style). ECBS-TA provides additional flexibility by adding the MinRoot and MaxRoot methods.

2.4 Hybrid MAPD

We have now described the algorithms that are used in our MAPD approach. However, the AGVs in the BHS need to attend to a continuous stream of tasks and ECBS(-TA) had therefore to be modified to allow for lifelong path finding (and task assignment).

To let AGVs continuously move between pickup and drop-off points, we used an approach in which ECBS(-TA) was invoked frequently throughout a simulation. When during a time step at least one AGV received a new target cell, by either finishing its current task or getting assigned a new bag, ECBS(-TA) was invoked but only new shortest path(s) were identified for that/those AGV(s). During ECBS(-TA) these new paths were validated with respect to each other and with respect to previously coordinated paths of other AGVs. On the other hand, ECBS(-TA) was invoked as well when an AGV became stationary. This is when it had arrived at a buffer spot and started waiting for a task, the AGV stayed at this location until it received a new target. ECBS(-TA) then validated existing paths with respect to the 'obstacle'.

A more extensive explanation of the workflow of a time step is provided in section 3.4, after the introduction of the created agent-based model.

3 Agent-based Model

An agent-based model was created in order to assess the performance of our MAPD algorithm in the baggage handling system. The infrastructure of this baggage handling system is presented in section 3.1. We developed a multi-agent system model in which AGV agents were required to transport as many bags as possible, section 3.2 introduces the agents in the MAS together with their properties and most important interactions. Then, in section 3.3, we explain how coordination was provided among the agents and we finally elaborate upon the workflow of our MAPD-based approach in section 3.4.

3.1 Environment

Our baggage hall lay-out is inspired by a concept of Vanderlande. In that concept, AGVs are not directly subjected to opposing traffic due to pre-defined streams of AGVs. This is represented by the right-hand side of the environment shown in fig. 4, in which the arrows indicate the AGV flows. We extended the lay-out with an area in which AGVs can charge and where they can wait for luggage to appear, represented by the left-hand side of the same figure. The entire baggage hall is a 60×13 grid and can manage a maximum amount of 70 AGVs.

The direction in which AGVs can move is represented by the overlaying graph in fig. 4. Apart from specific edges, all edges in the graph are bi-directional. This is done to allow AGVs to move in all directions during conflict-solving. However, to create the aforementioned AGV stream, several edges have been made uni-directional. These are the ones at the infeed stations (white arrows) and the ones at the small gates that connect the central hallway with the outer corridors (vertical arrows in between the chutes). AGVs thus enter the central hallway via the infeed stations, pass one of the 6 gates and return via the outer corridors where they deliver their bags. Moreover, some edges have been removed. This can be seen on the right-hand side of the infeeds, where the edges between the first cells are removed. We removed these to avoid congestion at the entrance of the hallway caused by AGVs that directly want to switch sides to line up for their gates. This gives AGVs the time to clear the center by moving to the outer lanes on their side of the hallway when required.

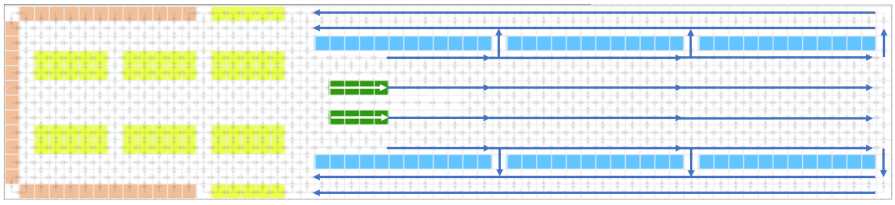


Figure 4: Baggage hall lay-out based on a concept of Vanderlande, including infeed stations (green), chutes (blue), charging stations (red) and buffer zones (yellow). The arrows indicate the pre-defined AGV flow and the gates are represented by the six vertical arrows.

3.2 Agents

To create the MAS, the infrastructural elements and the AGVs have been turned into agents with each their functionality. The most important ones are the infeed agents, that initiate task

assignment, and the AGV agents of which the behaviour decides upon the performance of the system. Furthermore the chutes, bags, buffer spots and charging stations are all represented by agents as well but mainly process and contain information that is communicated with the AGVs. This section highlights the properties that define the performance of the algorithm. We first address the infeed agents in section 3.2.1 after which the AGV agents are presented in section 3.2.2.

3.2.1 Infeed agents

Each infeed station has an infeed agent that is positioned on its most right cell. These agents are responsible for generating bags and running auctions for them. The infeed agents can be called upon at three different moments: (1) When a bag has to be auctioned, (2) during task reselling in DMB, and (3) when finding next-best task assignments in ECBS-TA. The infeed agents therefore need to be aware in which stage they are called and which AGVs can or cannot be considered during an auction. The *Auctioning property* is described below.

Auctioning property: As mentioned, the auction can be run at three different moments. The first is whenever an unassigned bag is present on the infeed station. This can be a bag that has just been generated (*bag generation property*) or a bag that had been generated in a previous time step but had not been assigned yet due to no AGVs being available at that time. The infeed agents initially start locating AGVs within a small search radius. This search radius gradually increases depending on whether identified AGVs have reported to be available or not (*check AGV availability property*). If available AGVs are found within the search radius, the infeed agent asks those AGVs to report their bid: their distance to the infeed and their energy level. The infeed agent then selects the closest AGV that has a sufficiently high energy level to complete the delivery. It should be noted that an AGV can win multiple bids if multiple auctions are run during a single time step. The second and third moment are respectively during task reselling and during next-best task assignment in ECBS-TA. In both of these, bags are possibly re-auctioned while taking into account constraints imposed on the involved infeed agent in which specific assignments are excluded. During DMB, an AGV that has won multiple bids resells the tasks that take the longest to reach. For these tasks, the auction is then run again by the corresponding infeed agents while neglecting the originally assigned AGV. During ECBS-TA, the next-best task assignments are found by re-auctioning the involved bags. The auction is run for the bag (and infeed station) that could not be part of the solution in a child node, neglecting the AGVs that were disallowed in its parent node.

3.2.2 AGV agents

The AGV agents have many properties and are involved in the most agent interactions. They are involved in the task assignment, they have to keep track of their battery level and find conflict-free paths to their dynamically changing targets. Dynamically changing implies the possibility of rapidly changing targets resulting from task swapping, altered task assignments from ECBS-TA or needing to charge their battery.

Task reselling property: If multiple bags are auctioned by different infeed agents, it is possible that an AGV ends up with multiple tasks. When this is the case, the AGV selects the task that suits it best (the closest one) and resells the remaining ones by re-invoking auctions for those tasks. Since the AGVs act out of self-interest when reselling tasks and do not keep the global goal in mind, i.e. minimizing bag waiting times, IDMB allows for swapping tasks between AGVs. Pseudocode for task reselling can be found below in algorithm 1.

Algorithm 1 Task reselling (DMB)

Input: A list of AGVs that have received a task in the current time step, DMB_list
Output: Assigned tasks based on self-interest
$bids\_won \leftarrow$ Tasks assigned to agv after CNP
while DMB_list not empty **do**
  for agv in DMB_list **do**
    if $agv_{bids\_won} > 1$ **then** ▷ AGV has more than 1 task assigned → resell
      Assign best bid to self ▷ Based on distance to infeed and energy level
      $sell\_bids \leftarrow$ List with tasks to resell
      for bid in $sell\_bids$ **do**
        Unassign bid from self
        $bag, inf \leftarrow$ Bag to resell & corresponding infeed agent
        $agvs\_available \leftarrow$ List with available AGVs identified by inf
        if $agvs\_available$ **then**
          $inf.auction(bag, resell)$ ▷ Re-auction and add winner to DMB_list
        else
          Bag will be re-auctioned next time step
    else
      Assign best bid to self
    remove agv from DMB_list

Task swapping property: Task swapping can occur in two stages in a simulation step: (1) After a regular auction and potentially reselling tasks and (2) during next-best task assignment in ECBS-TA. Both differ in which AGVs are considered for swapping. In the first option, the AGVs that have been assigned to a bag in the current time step ask all AGVs that are assigned to the other infeed whether it would be beneficial for minimal waiting times of bags to swap tasks. In the second option, the newly added AGV in the next-best task assignment node (fig. 3(a)) considers the other AGV(s) in the same assignment node to swap tasks. The pseudo-code for task swapping is shown below in algorithm 2.

Algorithm 2 Communication between agents to check for task swapping (IDMB)

Input: Local information of AGV agent asking for task swapping, AGVs to consider for task swap
Output: List of possible task swaps, $swap\_tasks$, for considered AGV
$self \leftarrow$ AGV agent asking for task swapping
$infeed_{self} \rightarrow$ Infeed assigned to $self$
$agvs\_for\_swapping \leftarrow$ AGVs assigned to other infeed than $infeed_{self}$
$swap\_tasks \leftarrow$ empty list ▷ Store AGVs with which to swap
for agv **in** $agvs\_for\_swapping$ **do**
  $cost_{tot} \leftarrow$ Total original path cost of both agents
  $swap\_cost_{tot} \leftarrow$ Total path cost of both agents when swapping
  if $swap\_cost_{tot} < cost_{tot}$ **then**
    $swap\_tasks \leftarrow$ Add agv ▷ Consider AGV for swapping
return $swap\_tasks$
if $swap\_tasks$ **then**
  $agv\_swap \leftarrow$ agv with lowest $swap\_cost_{tot}$
  $task\_swap(self, agv\_swap)$ ▷ Exchange bag & infeed properties

Charging property: The *Charging property* is an interaction between an AGV and a charging station agent. After deciding it has to go charge (*attery monitoring property*) and after having completed its task, the AGV reserves one of the available charging stations. The AGV selects the closest one available, reserves it and updates its target to the charging spot. If no charging station is available, it invokes its *idle property* to claim a buffer spot while waiting for a charging station to become available.

3.3 Coordination among agents

This section elaborates on how coordination is performed between the AGV agents. First we address an additional agent which is responsible for coordination in section 3.3.1. Hereafter, the highways that have been implemented in the environment are presented in section 3.3.2.

3.3.1 Coordination agent

To provide conflict-free paths for the AGVs, a coordination agent is added to the system which contains and runs the ECBS(-TA) algorithm. The coordination agent is called upon depending on whether (1) a certain amount of time has passed, (2) AGVs have received a new goal or (3) AGVs have become stationary. Based on whether new goals were obtained due to the task allocation mechanism, the agent invokes ECBS or ECBS-TA. The coordination agent communicates with the AGVs to let them provide the required information: the task allocation, stationary AGVs, previously coordinated paths of AGVs and AGVs that require a new path. After coordination, the coordination agent communicates the computed conflict-free paths and, in case of ECBS-TA, possibly updated task assignments to the corresponding AGV agents.

3.3.2 Highways

During preliminary research, we looked at which positions highways could contribute the most to ECBS(-TA). Due to expected bottlenecks in front of and behind the infeed stations and right behind the small gates, we decided to place the highways at those locations. They are visualized by the grey lines in fig. 5. The central highways had been created in order to line-up AGVs as early as possible when entering the infeed station and to postpone the crossing of AGVs as long as possible in order to avoid congestion behind the infeed stations. Those at the outer corridors had the means to not let AGVs, when they were returning from a delivery, obstruct the gateways by enforcing them to move to the most outer highway.

Figure 5: Infrastructure from section 3.1, showing the highways represented by the grey lines.

3.4 Model time step

Figure 6 finally shows the workflow of a time step in our MAPD approach. A time step starts, depending on the capacity and current time step, with bags appearing at the infeed stations. Based on whether bags and AGVs are available for auction, bags are assigned to the AGVs. If not, the task assignment process is skipped. Task allocation starts off with an auction in which the closest AGV gets the bag assigned (CNP). This initial task allocation is then optimized, using IDMB, in order to minimize the time that bags have to wait at an infeed station, ultimately maximizing the throughput of the BHS. During the next step, AGVs update their information (e.g. executing function, battery level, target,...) and they determine whether they require to be coordinated or not. If not, they continue their operation and keep following their current path. If they do, ECBS or ECBS-TA is invoked based on whether task allocation was performed earlier during the time step. Conflict-free paths (and updated task assignments) are computed and communicated to the corresponding AGVs. The AGVs subsequently move one step along their path, concluding the time step for the agents.

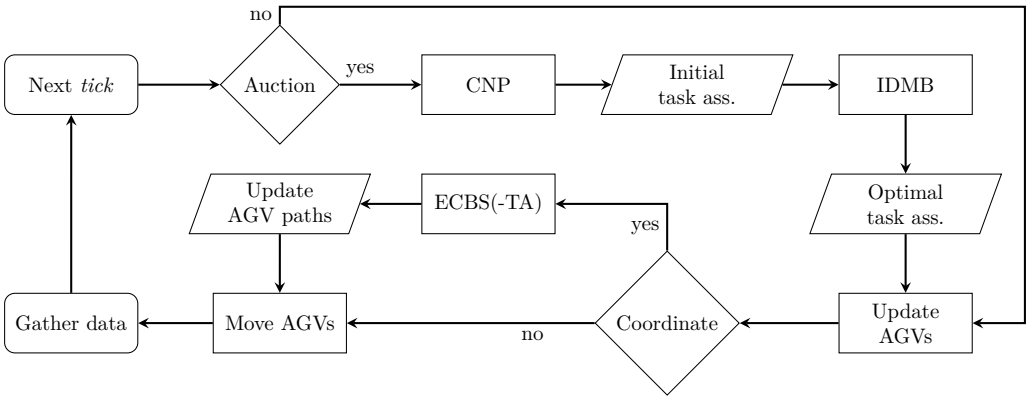


Figure 6: Overview of a simulation step.

4 Experimental Analysis

The multi-agent system model was developed and implemented in Python 3.7, using the novel ABM programming environment MESA. Simulations were run on a Windows 10 laptop with 8 GB RAM and an Intel Quad-Core i7-7700HQ processor. During development, continuous verification of the code was required to assure no compiler errors would occur during the analysis phase. These could originate from programming errors but also from agents that were not receiving the required information during simulation steps.

This section elaborates upon the simulation phase of our research. The simulation plan in section 4.1 first addresses the global set-up of the analysis of our MAPD algorithm. Secondly, in section 4.2, we elaborate upon the performance indicators that were important for assessing the algorithmic efficiency as well as the operational capacity of the baggage handling system. Then, sections 4.3 to 4.5 present the experiments that were carried out together with their results.

4.1 Simulation plan

The created model was simulated based on temporal discretization. This means that time was not continuous, but rather discretized in *'ticks'*. At every tick, the model execution would advance one time step in which each agent was activated. The overview of a simulation step was shown in fig. 6, section 3.4.

We compared our MAPD algorithm to two alternate methods that used the same components (i.e. CNP, IDMB & ECBS) but which both did not combine task assignment and path finding. The first method ran the CNP algorithm, explained in section 2.2, after which the task allocation was fixed and ECBS provided coordination based on the received targets (Auction+ECBS). The second method first provided the optimal task allocation using IDMB and then ran ECBS (IDMB+ECBS). Comparisons were performed using the Vargha-Delaney A-test [Vargha and Delaney, 2000]. These statistical values (< 1) represent the probability that there is a difference between two sets. Threshold values are 0.29, 0.36 and 0.44 which respectively indicate large, medium and small differences (i.e. observed values are more likely to be lower)[Sharpanskykh, 2020]. The same intervals apply to values above 0.5 (i.e. observed values are more likely to be higher). A value of 0.5 indicates that there is no difference between the two sets. A-test values were computed for IDMB+ECBS with respect to Auction+ECBS and for our MAPD approach with respect to IDMB+ECBS.

Three experiments were carried out for every method. The first experiment, laid out in section 4.3, determined the best ECBS(w_{ecbs})+HWY(w_{hwy}) configuration. The second one assessed the scalability of each of the algorithms and is presented in section 4.4. The final experiment, section 4.5, investigated the global performance of the system when charging of AGVs was included. Before attending to the experiments, section 4.2 elaborates upon the performance indicators that were used to evaluate the algorithms.

4.2 Performance indicators

The algorithms were assessed on two levels. On the one hand, we needed to test the algorithmic efficiency which included computational resources, but on the other hand the operational capacity of the system had to be evaluated as well. The performance indicators (PI) for both levels are elaborated upon in this section. Since the three experiments were meant to assess different aspects of our algorithm, specific PIs for each experiment and their corresponding hypotheses are laid out in the corresponding sections (sections 4.3 to 4.5).

4.2.1 Algorithmic efficiency

The first set of PIs that we used throughout the analysis were those that determined the efficiency of the algorithms. First of all, we had to keep in mind the **success rates**. These indicated the amount of times simulations were successfully completed and they were determined by the ability of ECBS(-TA) to find a solution. Whenever during a simulation ECBS(-TA) exceeded a computation time of 1 minute, the simulation was deemed unsuccessful. Furthermore, we needed to evaluate the **real-time applicability** of the system. To allow the system to operate in real-time, the runtime of a simulation step should not exceed 1 second. We therefore evaluated whether, over the course of a simulation, the **average step time** remained below 1 second per tick. Not only the runtimes played a role to assess the algorithms, also the cost of the solutions that were provided by ECBS(-TA) were taken into account. The **solution cost** is the sum of all individual path costs in a conflict-free solution provided by ECBS(-TA). Note that, in a path, highway cells cost less than regular cells with respectively costs of $1/w_{hwy}$ and 1. When comparing different highway factors,

the solution cost might not allow for a fair comparison. Therefore, we also looked at the average **delay** ($\boldsymbol{\Delta T}$) the AGVs were subjected to per trajectory. When an AGV received a new target, it calculated the shortest path to its destination which it compared to its actual path when having reached its destination. The difference in path length subsequently indicated the delay. The final, and one of the most important measures during this research, was the **scalability** of the system. This evaluated the amount of AGVs the system could handle while remaining real-time applicable and while providing success rates towards 100%.

4.2.2 Operational performance

Next to the algorithm efficiency, the operational performance of the system was tested. The first PI was the average **waiting time** of bags, the time between entering the system and being picked up by an AGV. In the same way, also their **service time** was being recorded. This is the time between entering the system and being delivered. Decreasing both parameters would normally lead to a higher **throughput**, the total number of bags that had been delivered at the end of a simulation. Alternatively, the operational capacity can be analysed [Olijslager, 2018]. This indicates the amount of bags that were picked up from the infeed stations and subsequently determines the efficiency ratio of the system: the operational capacity compared to the number of bags that had entered the system (i.e. system capacity), η = operational capacity/system capacity.

4.3 Experiment 1: ECBS+HWY factors

A previous study by Cohen et al. [2015] showed that ECBS can strongly benefit from adding highways to a system. If they capture the problem structure well, they can significantly reduce the amount of conflicts. When tested in Kiva-like environments, this resulted in lower runtimes and solution costs. However, we were dealing with a more dense system, where crossing AGVs occur more frequently. The simulations therefore had to show in what extent our lay-out could benefit from the implemented highways (see section 3.3.2).

4.3.1 Experimental set-up

The performance of ECBS(w_{ecbs})+HWY(w_{hwy}) is based on the two user-defined factors: the suboptimality factor for ECBS, w_{ecbs}, and the highway factor, w_{hwy}. Since there is no relation between both, this first experiment was performed to determine their best combination.

Set-up Preliminary analysis of the MAPD algorithm had shown that 50 AGVs were able to attend to a regular system capacity of 1800 bags/hour[1], with not all AGVs constantly transporting bags. The system configuration with 50 AGVs was therefore chosen as a baseline. To fully exploit the ECBS+HWY performance for 50 AGVs, a system capacity of 3600 bags/hour was used to continuously have the 50 AGVs on the move. We simulated approximately 5 minutes of operation (300 *ticks*) without charging of AGVs, to fully focus on how well the coordination mechanism performed.

Variables To find the optimal combination of suboptimality factor and highway factor, we respectively considered $w_{ecbs} \in \{1.1, 1.3, 1.5\}$ and $w_{hwy} \in \{1.5, 1.9\}$. We decided to limit the flexibility of ECBS to factor 1.5 in order to restrict the suboptimality of the solution. The highway factors were based on the work of Cohen et al. [2015], disregarding very low values ($w_{hwy} = 1.1$) as these did not deliver real-time applicable runtimes during preliminary testing.

[1]Typical average conveyor lines generally operate around a capacity of 1800 bags/hour. During maximum peak, this can increase to 3600 bags/hour [Glidepath, 2017].

PIs The performance of this first experiment was assessed based on algorithmic efficiency. More specifically, we looked at how often ECBS+HWY was able to successfully come up with a solution and how long it averagely took to find that solution. Furthermore, the obtained solution costs and delay would give an indication of the suboptimality of the computed paths.

Hypotheses Because higher suboptimality factors allow for more flexibility during coordination, it was hypothesized that high values for w_{ecbs} would deliver lower runtimes at the expense of higher solution costs and delays. On the other hand, $w_{hwy} = 1.9$ was expected to result in lower delays but higher runtimes compared to $w_{hwy} = 1.5$. The reason for the latter hypothesis was the fact that AGVs would be more encouraged to follow the highways, leading to additional conflicts around them.

4.3.2 Results

Table 1 summarizes the set-up for this experiment. For each method and every combination of w_{ecbs} and w_{hwy}, at least 100 simulations were run, taking around 84 hours to complete.

Table 1: Experimental set-up for experiment 1.

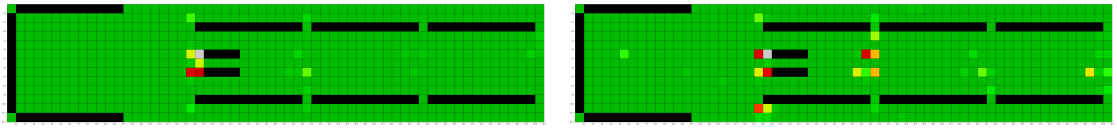
Set-up	**Independent variables**	**PIs**
50 AGVs	$w_{ecbs} \in \{1.1, 1.3, 1.5\}$	Success rate
capacity = 3600 bags/hour	$w_{hwy} \in \{1.5, 1.9\}$	Runtimes
300 *ticks*		Solution costs
Charging = *False*		Delay

The results of the first experiment, presented in table 2, showed that our hybrid MAPD approach, apart from runtimes, performed better than the Auction+ECBS algorithm but was clearly outperformed by IDMB+ECBS as the A-test values confirm. Looking at success rates, average solution costs and average delay for both IDMB+ECBS and the MAPD algorithm, we could deduce that the shared task allocation mechanism played a significant role in the performance of ECBS and ECBS-TA. This can be explained by the way in which tasks are assigned by CNP and by IDMB as elaborated upon below.

Table 2: The results of experiment 1 in which the bold numbers indicate the best results per row. 'A' presents the A-test values for IDMB+ECBS and our Hybrid MAPD algorithm with respect to respectively Auction+ECBS and IDMB+ECBS.

		Auction + ECBS				**IDMB + ECBS**				**Hybrid MAPD**			
w_{ecbs}	w_{hwy}	Suc.	t_{coord} [s]	Cost	ΔT [*ticks*]	Suc.	t_{coord} [s]	Cost	ΔT [*ticks*]	Suc.	t_{coord} [s]	Cost	ΔT [*ticks*]
1.1	1.5	.97	0.192	845.67	0.215	**1.**	**0.135** A: 0.16	**840.83** A: 0.43	**0.152** A: 0.10	**1.**	0.223 A: 0.95	843.46 A: 0.53	0.155 A: 0.52
	1.9	.97	0.225	845.15	0.192	**1.**	**0.161** A: 0.19	**843.76** A: 0.42	**0.143** A: 0.11	.99	0.249 A: 0.94	849.09 A: 0.58	0.144 A: 0.53
1.3	1.5	.98	0.207	844.43	0.213	**1.**	**0.136** A: 0.12	**840.17** A: 0.41	**0.153** A: 0.10	.99	0.221 A: 0.94	844.0 A: 0.59	**0.153** A: 0.49
	1.9	.97	0.217	852.05	0.19	**1.**	**0.161** A: 0.16	**847.46** A: 0.42	**0.137** A: 0.09	.99	0.263 A: 0.95	849.15 A: 0.52	0.141 A: 0.52
1.5	1.5	.96	0.200	842.76	0.215	**1.**	**0.135** A: 0.14	844.47 A: 0.52	**0.148** A: 0.07	**1.**	0.225 A: 0.96	**842.60** A: 0.47	0.163 A: 0.62
	1.9	.98	0.222	851.57	0.197	.98	**0.163** A: 0.16	**847.47** A: 0.42	**0.138** A: 0.07	**1.**	0.259 A: 0.94	848.40 A: 0.53	0.143 A: 0.54

Looking at the heat maps in fig. 7, one can see that traffic is highly concentrated in front of the infeed stations. This is especially the case for the Auction+ECBS algorithm in fig. 7(a) as in the rest of the baggage hall not many conflicts are observed. Comparing this to fig. 7(b), the latter shows that conflicts are more spread throughout the hall. This is caused by emerging AGV flows resulting from the task allocation mechanism. The auction mechanism (CNP) assigns bags to AGVs that are closest to the corresponding infeed stations. AGVs can therefore be located at the upper half of the environment and get assigned to the lower infeed and vice versa. This locally increases the amount of conflicts ECBS has to deal with in front of the infeed stations as crossing traffic arises. What emerges from IDMB on the other hand, is that in general AGVs that are coming from the upper chutes get assigned to the upper infeed and AGVs coming from the lower chutes get assigned to the lower infeed. Crossing traffic in front of the infeed stations therefore reduces significantly. For this reason, conflicts arising in Auction+ECBS simulations resulted in higher solution costs and delays, compared to the other two algorithms, and resulted in significantly higher computation times for ECBS compared to the MAPF+TA algorithm using IDMB.



(a) Heatmap Auction+ECBS

(b) Heatmap hybrid MAPD

Figure 7: Heat maps showing the spread of conflicts in the baggage hall. The intensity of the colours in the heatmaps (from green, no conflicts, to red) are defined relative to the maximum amount of conflicts that had to be solved at the corresponding location. It can thus not be assumed that our hybrid mechanism in fig. 7(b) had to solve more conflicts in the central hallway.

Considering the runtimes of the three mechanisms, ECBS+HWY dealt well with the amount of AGVs and the lay-out of the system. Our MAPD approach did however deliver the highest computation times required by ECBS(-TA). These were attributed to the additional auctions and task swapping in the next-best task assignments and the additional constraint trees that were created. Since we were dealing with a very dense system, compared to a Kiva lay-out, conflicts between agents were often present in different constraint trees. The same conflicts were therefore solved multiple times. It was however considered a possibility that increasing the amount of AGVs would cause more difficulties for the MAPF+TA algorithms since these allowed for less flexibility due to not considering alternative task allocations. Especially Auction+ECBS was expected to scale worse as this experiment showed that ECBS had more trouble coordinating the AGVs with a non-optimal task assignment. This would be further investigated in experiment 2.

Deciding upon which ECBS+HWY configuration to continue with was based on the values in table 2. Despite $w_{hwy} = 1.9$ providing AGVs with less delay, both runtimes and solution costs were consistently higher. This indicated that a highway factor of 1.9 reduced the cost of highways to such an extent that it lead to additional conflicts for ECBS to deal with, confirming our second hypothesis. The first hypothesis was that higher values for w_{ecbs} would perform better in terms of runtime but worse in terms of solution costs and delay. Our results could however not confirm this hypothesis as there were no notable and consistent differences between the suboptimality factors in combination with $w_{hwy} = 1.5$. We therefore decided to continue the second experiment with the least suboptimal configuration, ECBS(1.1)+HWY(1.5), as a baseline.

4.4 Experiment 2: Exploring scalability

The goal of our research was eventually to have a scalable MAPD algorithm in order to (temporarily) allow more AGVs in the environment when required by the system capacity. The outcome of this second experiment would therefore be the number of AGVs at which the algorithm remained real-time applicable.

4.4.1 Experimental set-up

When more AGVs become involved, runtimes of IDMB and ECBS(-TA) increase significantly due to the exponential increase of, respectively, required communication and search space (NP-hardness). By not letting an AGV agent consider all other AGVs when swapping tasks (algorithm 2), the amount of required communication, and therefore required runtime, had already partly been minimized. However, to ease solution finding for ECBS(-TA) when the number of AGVs increases, we wanted to assess whether the coordination mechanism would benefit from allowing higher sub-optimality.

Variables As mentioned before, the second experiment used the outcome of the first experiment as baseline. This was a system configuration with 50 AGVs and ECBS(1.1)+HWY(1.5). In this experiment, we considered more AGVs, #AGVs $\in \{60, 70\}$, and used suboptimality factors $w_{ecbs} \in \{1.1, 1.5, 1.9\}$ to evaluate in what extent higher suboptimality affected the system performance. The highway factor of 1.5 remained unchanged.

PIs During this experiment, we did not only look into the algorithmic efficiency. The operational performance of the system also became involved. For the algorithmic efficiency, the difference with experiment 1 was that we now split up runtimes in total runtime per step and the required computation time by the coordination mechanism. Considering the operational performance, we looked into waiting and service times of bags and the amount of bags the system was able to process.

Hypotheses As mentioned in the previous experiment, we expected that the MAPF+TA algorithms would experience more difficulties during coordination, compared to our MAPD approach, when the number of involved AGVs increased. It was therefore hypothesized that our MAPD approach would deliver better runtimes regarding coordination than the other two methods. Furthermore, we hypothesized that both our MAPD algorithm and IDMB+ECBS would deliver higher operational capacity than Auction+ECBS as both share the (initial) task allocation method, IDMB.

4.4.2 Results

Table 3 summarizes the parameters and performance indicators for the second experiment. At least 50 successful simulations were run for each combination of the independent variables, leading to about 111 hours of simulating.

Table 3: Experimental set-up for experiment 2.

Set-up	**Independent variables**	**PIs**
w_{hwy} from experiment 1	$\#AGVs \in \{60, 70\}$	Success rate
capacity = 3600 bags/hour	$w_{ecbs} \in \{1.5, 1.7, 1.9\}$	Runtimes
300 *ticks*		Solution costs
Charging = *False*		t_{wait}, $t_{service}$
		Throughput

We first evaluated the algorithmic efficiency of all mechanisms. The results of this experiment, provided in table 4, followed the trend from experiment 1 in section 4.3 where IDMB+ECBS outperformed the other algorithms. Considering the average total step times, the increments when adding 10 AGVs were in every case almost equal to the increments in required computation times for ECBS(-TA). This meant that adding robots did not have a big impact on the task allocation mechanisms but mainly influenced coordination: Required computation times for ECBS(-TA) in all three mechanisms at least doubled each time 10 AGVs were added. The impact on coordination translated further to decreasing success rates.

As hypothesized, we expected that the MAPF+TA algorithms would experience more difficulties during coordination compared to our MAPD algorithm. This was however not entirely the case. IDMB+ECBS was able to outperform both our MAPD algorithm and Auction+ECBS due to the AGV flows that emerged[2] from its optimal task assignment as explained in section 4.3. On the other hand, between our MAPD approach and Auction+ECBS, difficulties for Auction+ECBS significantly increased with the amount of AGVs. The difference between computation times for coordination with our MAPD approach became smaller and our algorithm was eventually able to match those of Auction+ECBS. Our first hypothesis was therefore partly confirmed. More notable was that the required computation times for ECBS in Auction+ECBS increased to such an extent that, despite its significantly longer task allocation process, with 70 AGVs, IDMB+ECBS was able to deliver lower total runtimes.

Table 4: Results of experiment 2: Algorithmic efficiency. The table provides the average values over the 50 successful simulations. The values in bold indicate the best result per AGV-w_{ecbs} combination, whereas 'A' represents the A-test statistical value.

		Auction + ECBS				**IDMB + ECBS**				**Hybrid MAPD**			
AGVs	w_{ecbs}	Suc.	t_{run} [s]	t_{coord} [s]	Cost	Suc.	t_{run} [s]	t_{coord} [s]	Cost	Suc.	t_{run} [s]	t_{coord} [s]	Cost
50	1.1	0.97	**0.326**	0.192	845.67	**1.**	0.526 A: 0.99	**0.135** A: 0.16	**840.83** A: 0.43	**1.**	0.575 A: 0.80	0.223 A: 0.95	843.46 A: 0.53
60	1.1	0.82	**0.585**	0.431	979.94	0.94	0.695 A: 0.76	**0.294** A: 0.21	**976.53** A: 0.45	**0.96**	0.879 A: 0.91	0.512 A: 0.92	983.575 A: 0.59
	1.5	0.82	**0.583**	0.431	986.99	**1.**	0.708 A: 0.78	**0.286** A: 0.16	982.88 A: 0.45	0.98	0.875 A: 0.96	0.512 A: 0.97	**976.82** A: 0.44
	1.9	0.78	**0.606**	0.457	**977.74**	0.94	0.693 A: 0.77	**0.266** A: 0.10	980.19 A: 0.54	**0.96**	0.880 A: 0.92	0.505 A: 0.97	981.51 A: 0.54
70	1.1	0.37	1.092	0.967	1039.01	**0.69**	**0.966** A: 0.34	**0.615** A: 0.14	**1027.95** A: 0.42	0.57	1.356 A: 0.91	1.032 A: 0.92	1033.21 A: 0.53
	1.5	0.62	1.145	1.010	1036.27	**0.69**	**1.072** A: 0.45	**0.648** A: 0.18	**1021.04** A: 0.40	0.67	1.323 A: 0.78	1.014 A: 0.89	1029.51 A: 0.56
	1.9	0.55	1.186	1.044	1041.49	**0.79**	**1.024** A: 0.31	**0.611** A: 0.12	**1021.61** A: 0.37	0.63	1.367 A: 0.85	1.056 A: 0.92	1034.00 A: 0.57

Table 5 provides more insight in the performance of the coordination mechanism in our MAPD algorithm. It specifies the average runtimes of ECBS and ECBS-TA separately, how many times both are invoked and how often the task assignment changed because of ECBS-TA. Comparing t_{ecbs} from table 5 with t_{coord} for IDMB+ECBS in table 4, running ECBS in our MAPD approach always took longer. One would however expect them to be approximately equal as no additional constraint trees needed to be expanded. The explanation for this behaviour is the following: When ECBS-TA provides a solution with an alternative task assignment, the task allocation might be less optimal

[2]AGVs coming from the upper half of the environment move to upper infeed and vice versa leading to less conflicts in front of the infeed stations.

than the one provided by IDMB. This can be attributed to the implemented windowed search to lower computation times for ECBS(-TA), as explained in section 2.1. The windowed search only validates paths within a defined time horizon which causes ECBS-TA to solve TAPF instances only within that time window. The paths of AGVs that are involved in the next-best task assignment might therefore not be optimized entirely until the infeed stations where the most conflicts occur. Due to the possible suboptimal task assignment, it is therefore likely that the aforementioned problem[3], regarding task allocation causing increased amounts of conflicts at the infeed station, returns. On the other hand, the extensive runtimes of ECBS-TA were the consequence of a combination of the baggage hall lay-out[4] and the same conflicts having to be resolved in different constraint trees.

Table 5: Average ECBS(-TA) performance of the Hybrid MAPD algorithm, showing the individual average runtimes of ECBS and ECBS-TA together with the amount of times they are invoked. The last column displays the number of new task allocations resulting from ECBS-TA.

AGVs	t_{ecbs} [s]	$t_{ecbs-ta}$ [s]	ECBS/ECBS-TA [#]	Next-best TA [#]
50	0.157	0.263	102/173	11
60	0.337	0.636	119/162	41
70	0.701	1.330	133/150	66

Besides the algorithmic efficiency, the performance regarding operational capacity was analysed of which the results are shown in table 6. Here, our MAPD algorithm showed its strengths. Although not by much, it outperformed both MAPF+TA algorithms in every aspect. The difference with IDMB+ECBS was generally very small but the A-test values were almost always in favour of our MAPD algorithm, with small to medium differences between both. Comparing them to Auction+ECBS, on the other hand, clearly demonstrated the influence of IDMB. Large differences (i.e. threshold of 0.29) arose from the A-test values for all PIs and, as the average results indicated, they were able to let AGVs attend to bags more quickly (with less delay) and eventually handle more bags. Our second hypothesis was thereby confirmed.

Table 6: Results of experiment 2: Operational performance. Bold values and 'A' again respectively indicate the best results and A-test values.

		Auction + ECBS				**IDMB + ECBS**				**Hybrid MAPD**			
AGVs	w_{ecbs}	t_{wait} [s]	t_{serv} [s]	ΔT [*ticks*]	# bags	t_{wait} [s]	t_{serv} [s]	ΔT [*ticks*]	# bags	t_{wait} [s]	t_{serv} [s]	ΔT [*ticks*]	# bags
50	1.1	39.27	68.12	0.215	204.4	**37.60** A: 0.30	**66.51** A: 0.33	**0.152** A: 0.10	**207.0** A: 0.64	37.72 A: 0.52	66.68 A: 0.52	0.155 A: 0.52	206.9 A: 0.50
60	1.1	25.26	56.48	0.256	235.2	23.79 A: 0.28	**55.12** A: 0.32	0.202 A: 0.17	237.7 A: 0.63	**23.76** A: 0.5	55.23 A: 0.53	**0.195** A: 0.41	**238.0** A: 0.52
	1.5	26.05	57.38	0.257	232.5	23.87 A: 0.20	55.21 A: 0.24	**0.187** A: 0.10	237.2 A: 0.75	**23.63** A: 0.46	**54.89** A: 0.46	0.195 A: 0.55	**237.6** A: 0.54
	1.9	25.49	56.66	0.265	235.1	24.09 A: 0.28	55.43 A: 0.33	0.189 A: 0.68	237.4 A: 0.62	**23.63** A: 0.43	**55.11** A: 0.47	**0.184** A: 0.45	**238** A: 0.52
70	1.1	20.46	52.83	0.234	245.9	19.61 A: 0.30	51.84 A: 0.32	0.204 A: 0.29	246.9 A: 0.56	**19.26** A: 0.40	**51.59** A: 0.45	**0.183** A: 0.37	**248.3** A: 0.60
	1.5	20.50	52.85	0.227	245.7	19.97 A: 0.32	52.14 A: 0.30	0.199 A: 0.29	**246.8** A: 0.59	**19.51** A: 0.32	**51.75** A: 0.40	**0.185** A: 0.41	**246.8** A: 0.51
	1.9	20.79	53.18	0.235	245.2	19.86 A: 0.22	52.01 A: 0.27	0.187 A: 0.15	246.7 A: 0.63	**19.24** A: 0.30	**51.49** A: 0.37	**0.177** A: 0.40	**247.6** A: 0.58

[3]AGVs coming from the upper half of the environment move to the lower infeed and vice versa, increasing the amount of conflicts ECBS(-TA) has to deal with at the infeed stations (section 4.3.2).

[4]Previous research [Hönig et al., 2018] used a Kiva-like environment in which pickup points were more spread throughout the infrastructure.

The final experiment continued with the best-performing, real-time applicable configuration for each method. As none of them allowed for real-time application with 70 AGVs - IDMB+ECBS with $w_{ecbs} = 1.1$ only had a success rate of 69% -, we decided to use 60 AGVs.

4.5 Experiment 3: Global performance

The final experiment introduced charging in the AGV process in a full operation of approximately 30 minutes real-time. We looked more extensively at the performance of individual AGVs and the emerging properties which came forth from charging. This experiment would have to determine whether our hybrid MAPD algorithm indeed improved upon separate task assignment and path finding algorithms.

4.5.1 Experimental set-up

The previous experiment showed that none of the algorithms could provide real-time applicability with 70 AGVs, as runtimes below 1 second could not be guaranteed. For comparison of our MAPD and the two MAPF+TA approaches, we continued with the best performing ECBS+HWY configuration for each mechanism using 60 AGVs.

Set-up To see how charging would affect the system performance, the maximum battery level of the AGVs was chosen so that two global charging periods would arise. All values regarding energy consumption were decided upon arbitrarily and are shown below in table 7.

Table 7: Energy consumption specifications for AGVs

Max. battery level	Min. energy threshold[5]	Moving	Stationary	Charging
720	100	-1/*tick*	-0.5/*tick*	+4/*tick*

The suboptimality factors that we used for the three methods were $w_{ecbs} = 1.7$[6] for Auction+ECBS and $w_{ecbs} = 1.5$ for IDMB+ECBS and our MAPD approach. All of them were used in combination with $w_{hwy} = 1.5$ as these delivered the best performance in experiment 2.

PIs Including charging in the AGV process would introduce new complexities for the algorithms as the charging periods would increase the traffic density in the left part of the baggage hall. We therefore analysed the algorithmic efficiency throughout the whole simulation period. Additionally, we looked into the emerging behaviour due to charging. This included, among others, the amount of active AGVs and charging AGVs and how these affected the operational capacity.

Hypotheses Because of the limited amount of charging stations, many AGVs were expected to be stationary during periods of charging. It was hypothesized that the computation time for coordination during these periods would not increase significantly, relative to the rest of the operation. However, as more AGVs would be near the infeed stations during those periods, the required time for task assignment was expected to increase when more and more AGVs finished charging. On the other hand, we presumed that our MAPD algorithm would benefit from the higher concentration of AGVs as optimizing task assignment and path finding would become more complex. We expected that this would result in lower service times for bags compared to the MAPF+TA algorithms.

[5]Energy level at which AGVs become aware of needing to go charge.

[6]Despite not covered in section 4.4, simulations were also performed for $w_{ecbs} = 1.7$. This delivered better results in terms of runtimes and solution costs compared to $w_{ecbs} = 1.1$ with values of respectively 0.561 (against 0.585) and 978.1 (against 979.94). Furthermore, delays of 'only' 0.244 ticks were experienced, compared to 0.256.

4.5.2 Results

The set-up of this final experiment is summarized in table 8. An operation of 30 minutes real-time (1800 ticks) was simulated. We therefore restricted the amount of simulations to 20 (successful ones), requiring 32 hours of simulating.

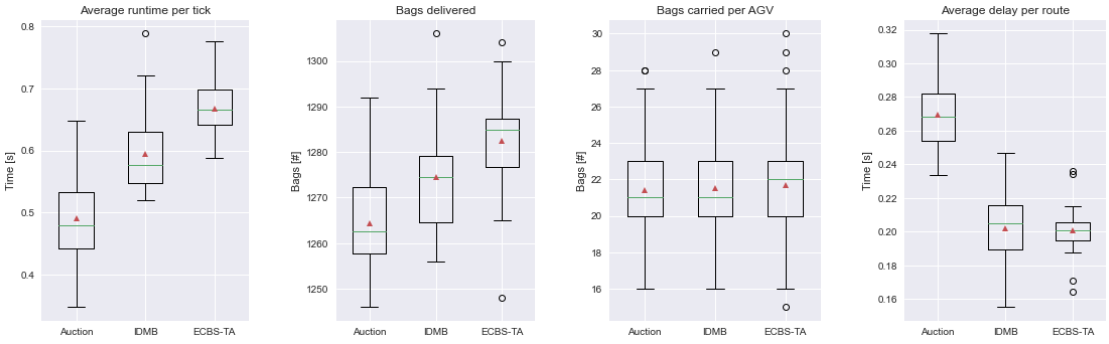
Table 8: Experimental set-up for experiment 3.

Set-up		**Ind. variables**	**PIs**
60 AGVs	1800 *ticks*	None	Runtimes
w_{ecbs} from experiment 2	Charging = *True*		$t_{service}$, ΔT
$w_{hwy} = 1.5$	AGV energy = 720		Throughput
capacity = 3600 bags/hour			AGV performance

Table 9 provides the main results of the third experiment. The order of best performance in terms of average runtimes remained unchanged compared to experiment 2, but the difference in operational performance between the three algorithms became more apparent. As expected, our MAPD approach was able to pickup bags at a higher rate and, as the box plots in fig. 8 show, overall more bags were delivered and delays were more consistently lower. Looking at the amount of bags transported by each AGV in fig. 8(c), all three algorithms seemed to perform similarly. However, the slightly higher median and average in the MAPD box (denoted as ECBS-TA in the plot) explain its higher throughput.

Table 9: The performance of the different algorithms in a system with 60 AGVs, a system capacity of 3600 bags/hour and a simulation period of 30 minutes real-time.

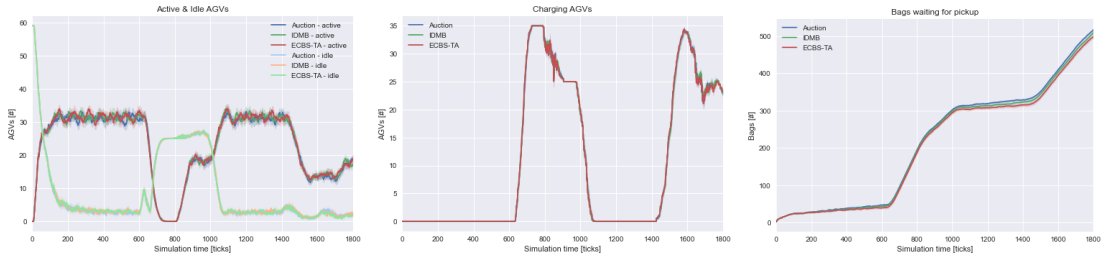
	t_{run} [s]		t_{serv} [*ticks*]		*#bags*		ΔT [*ticks*]	
Auction	$\boldsymbol{\mu = 0.49}$	$\sigma = 0.07$	$\mu = 222.7$	$\sigma = 5.9$	$\mu = 1264$	$\sigma = 10.5$	$\mu = 0.27$	$\sigma = 0.02$
IDMB	$\mu = 0.59$	$\sigma = 0.07$	$\mu = 219.8$	$\sigma = 7.0$	$\mu = 1274$	$\sigma = 12.2$	$\mu = 0.20$	$\sigma = 0.02$
MAPD	$\mu = 0.67$	$\sigma = 0.05$	$\boldsymbol{\mu = 215.3}$	$\sigma = 7.1$	$\boldsymbol{\mu = 1282}$	$\sigma = 11.7$	$\boldsymbol{\mu = 0.20}$	$\sigma = 0.02$



(a) Average runtime. (b) Throughput. (c) Bags per AGV. (d) Delay per route.

Figure 8: Box plots indicating the consistent improvement of our MAPD algorithm over the MAPD+TA algorithms in operational capacity and delay.

As mentioned, the amount of bags that were transported throughout the simulations per AGV was similar for each of the methods. It was therefore expected that no differences would arise when assessing AGV behaviour. This is confirmed by the similar graphs in fig. 9 which show the behaviour of AGVs throughout the simulation periods. Figures 9(a) and 9(b) first display the amount of AGVs that were active (i.e. delivering a bag), the amount of idle AGVs (i.e. not allocated to a task) and the amount of charging AGVs per time step, with the two large peaks indicating the aforementioned global charging periods. As soon as the first AGVs became aware of their draining batteries, around step 600, the amount of active AGVs dropped to zero and the bags started to pile-up at the infeed stations as shown in fig. 9(c). In this figure, it can be seen how the blue and green lines (those for of the MAPF+TA algorithms), slowly started rising above the line of our hybrid MAPD solution. This visualizes the slightly higher rate at which our proposed model was able to let AGVs pickup bags due to jointly optimizing task allocation and path finding, confirming our hypothesis.

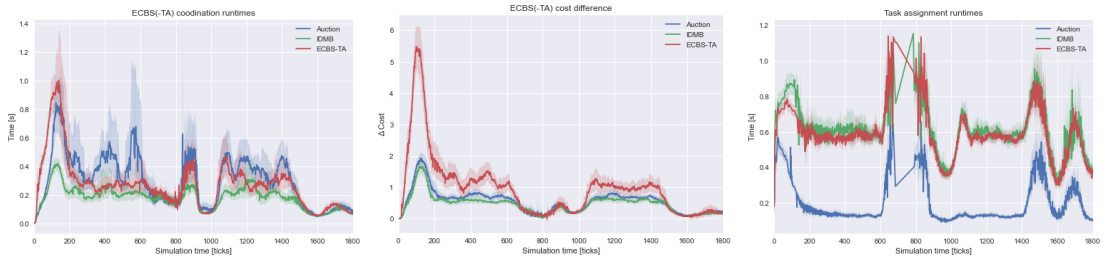


(a) Active & idle AGVs (b) Charging AGVs (c) Unattended bags

Figure 9: Influence of charging on AGV usage and operational capacity.

Figure 10 shows the behaviour of the task allocation and coordination mechanisms throughout the simulation period. Recalling table 9, one could observe that the average runtimes of the three algorithms were remarkably lower than those of experiment 2 with differences of around 0.1s for the MAPF+TA algorithms and 0.2s for our MAPD algorithm. This becomes clear when figs. 10(a) and 10(b) are considered. In both graphs, clear peaks can be distinguished within the first 300 ticks with maxima occurring at around step 100. Looking at fig. 9(a), this is the moment at which all AGVs are active. Bags appeared at a rate of 3600 bags/hour, so averagely 1 bag was auctioned per tick. This created a lot of traffic in the buffer area. As the high runtimes and solution costs in figs. 10(a) and 10(b) indicate, ECBS(-TA) struggled with this high amount of concentrated traffic. During the rest of the simulation period, ECBS(-TA) runtimes and solution costs significantly lowered, causing the average step times in table 9 to decrease with respect to experiment 2.

Delays roughly followed the same trend as fig. 10(b), indicating that the more AGVs were on the move, the higher solution costs and delays became. Overall, our hybrid MAPD algorithm delivered the highest solution cost and one might expect that this goes paired with higher delays. However, due to the highly suboptimal task assignments provided by CNP, AGVs in the Auction+ECBS method were still subjected to the highest delays due to the conflicts emerging at the infeed stations. Higher solution costs in our MAPD algorithm can be attributed to the next-best task assignment in ECBS-TA. When a solution considered an alternative task allocation, additional AGVs could be included in the solution: both the originally and the newly assigned AGV needed to be coordinated to respectively an updated destination and the assigned infeed station.



(a) ECBS(-TA) runtimes (b) Δ Solution cost (c) Task allocation runtimes

Figure 10: Moving averages (and corresponding standard variances) of the coordination and task allocation performance showing their runtimes and the difference between the solution cost and the initial root cost as a result from ECBS-TA. To clarify the graphs, time windows of respectively 30, 60 and 30 ticks have been used.

Initially, all AGVs were given the same battery level, charging therefore only started affecting the simulations at around the 600^{th} time step when the first AGVs needed to go charge. At that point, all AGVs started to move to the buffer/charging area in order to charge their battery. The traffic density therefore rose significantly and one could expect that the initial high peaks in ECBS(-TA) runtimes and solution costs would therefore return. However, as we hypothesized, this did not occur. Because there were only 35 charging stations, the other AGVs needed to wait at their reserved buffer spot for charging stations to become available. Most of the AGVs were therefore stationary. Because of this, ECBS(-TA) did not experience any problems as AGVs only, one by one, slowly started to operate again.

On the other hand, bags kept appearing at the infeed stations. The infeed agents therefore kept communicating to the AGVs that bags were ready to be picked up. When less and less AGVs were reporting to be available, the infeed agents increasingly had to expand their search area, requiring more and more time to allocate tasks. This is what the peaks in fig. 10(c) indicate. One might notice the gap around step 700. Because no bags were allocated due to AGVs charging and none of the other AGVs having sufficient energy, runtimes were not registered. Note that the initial smaller peak was caused by the fact that all AGVs were situated in the buffer areas. Many AGVs could therefore be considered during task assignment, increasing the required communication between the infeed agents and the AGV agents (CNP) and among the AGV agents themselves (IDMB).

From this experiment, we can conclude that our hybrid MAPD algorithm does outperform separate MAPF+TA approaches in terms of operational capacity. Higher throughputs were guaranteed and AGVs were consistently subjected to lower delays. Considering algorithmic efficiency, our algorithm did deliver higher runtimes but these were kept sufficiently below 1 second to allow for real-time applicability in the BHS with 60 AGVs.

5 Discussion

This research showed that combining optimal task assignment and path finding in a multi-agent pickup-and-delivery problem has the ability to provide better operational capacity in a distributed baggage handling system, however, at the expense of higher runtimes.

The experiments indicated that our MAPD algorithm delivered significantly higher runtimes than the other algorithms. This can be attributed to several factors. First of all, we used the IDMB algorithm which requires a significant amount of communication between agents (exponential with the amount of involved agents), substantially increasing the time it takes to find a task assignment. This required time had partly been reduced by identifying AGVs with which tasks could be swapped upfront to avoid unnecessary communication. Secondly, the next-best task assignment played a small role in required computation time as well. When tasks were re-auctioned, communication was again required in order to optimize the next-best task assignment. A third reason can be attributed to the next-best task assignment as well, but this time because of the fact that multiple constraint trees were used. As all AGVs move towards the same area (the infeed stations), conflicts would often, if not always, re-appear in the additional constraint trees. This led to the same conflicts having to be resolved multiple times. This is part of the reason why we did not choose the CBS-style or MaxRoot approach in ECBS-TA, but continued with the MinRoot method. This way, the amount of additional constraint trees would be reduced. We therefore propose further research to address different ways to combine IDMB and ECBS-TA. A first option could be to not use IDMB entirely by not allowing task swapping before invoking ECBS-TA and using solely the next-best task assignment to optimize task allocation. The initial task assignment could then be restricted to CNP only or to DMB (i.e. only reselling). On the other hand, the next-best task assignment could be altered as well by keeping it relatively simple and therefore only re-auctioning tasks without task swapping within an assignment node.

Furthermore, the ECBS(-TA) algorithm did not perform as expected as the Auction+ECBS and our hybrid MAPD approach did not allow for real-time application with ECBS(-TA) runtimes exceeding 1 second. ECBS in the IDMB+ECBS approach was however able to provide solutions in only around 0.6 seconds, but the IDMB runtimes caused total step times often to exceed 1 second. The difference in coordination runtimes, presented in section 4.4, clearly indicated why this was happening: IDMB caused the paths of AGVs to interfere less by creating AGV flows from the upper chutes to the upper infeed and the lower chutes to the lower infeed. This showed the bottleneck of the system and could potentially be mitigated by altering the lay-out of the left-hand part of the environment (e.g. create a large buffer area that acts as a waiting line for AGVs to pick up bags) and reducing the movement freedom of AGVs in front of the infeed stations to restrict interfering traffic. Doing this would however limit the contribution of combined task assignment and path finding as the movement freedom in this baggage hall environment is necessary to profit from our MAPD approach. Issues with the lay-out can further be confirmed by the fact that when using low highway factors ($w_{hwy} = 1.1$), ECBS did not allow for real-time application with only 50 AGVs in the system.

The global performance in section 4.5 showed the influence of charging on the operational capacity of the system. With 2 bags appearing every 2 ticks (3600 bags/hour), bags piled up very quickly when AGVs were charging. In real-life however, not all AGVs would have exactly the same energy level at the start. This would prevent the global charging periods from occurring and therefore reduce the bag pile-up. Furthermore, the charging level of the AGVs did not represent that of current real-life AGVs as it was merely used to include charging in the simulations while keeping simulation times reasonable. Additional simulations were carried out and showed that without charging, with 60 AGVs and at a system capacity of 3600 bags/hour, efficiencies around 94% (instead of 71%) could be obtained in an operation with a timespan of 30 minutes.

Finally, the created multi-agent system does not fully capture a real-life operation as time was discretized. Seconds were simulated as ticks at which the simulation advanced with one step. During

each step, groups of agents were activated in a particular order which brought the environment to its next state. As we wanted to focus on combining task assignment and path finding, kinematics were not within the scope of this research and robots therefore moved at a speed of 1 cell per tick.

6 Conclusion

This research proposed a hybrid approach for the Multi-Agent Pickup-and-Delivery problem in a distributed baggage handling system. As baggage handling systems in general only need to operate at full capacity for a couple of months during their lifetime, flexible baggage handling systems are being developed which use AGVs to handle a required capacity. Solutions for such systems are captured by the MAPD problem. In state-of-the-art online MAPD algorithms, task assignment and path finding are implemented as two separate processes. Studies have shown that combining both, covered by TAPF problems, can increase solution optimality and/or scalability. Yet, these TAPF problems do not allow AGVs to attend to a continuous stream of tasks as it stops when each robot has a conflict-free path assigned to its target.

A agent-based model was created to simulate a baggage handling system which incorporated a distributed solution for the MAPD problem. The proposed hybrid solution used IDMB as optimal task allocation mechanism and ECBS-TA (TAPF) to provide coordinated paths and alternative task assignments. To create an MAPD solution, the ECBS-TA algorithm was modified into a lifelong version of the TAPF problem. The hybrid MAPD algorithm was eventually carefully assessed in three experiments. We first evaluated which combination of suboptimality and highway factor suited our environment best. No significant difference arose between the suboptimality factors. The highway factors on the other hand, clearly had an impact on the real-time applicability of the algorithms. A factor of 1.1 was not able to sufficiently enforce AGVs to follow the highways whereas a factor of 1.9 enforced the AGVs too much, causing additional conflicts to occur around the highways. Secondly, we explored the scalability of the system. Testing different suboptimality factors with increasing amount of AGVs showed that, to remain real-time applicable, all algorithms were limited to 60 AGVs. Due to its optimal task allocation, our MAPD algorithm was not able to outperform the MAPF+TA mechanism using CNP (Auction+ECBS) in total runtimes. ECBS and ECBS-TA, in both our approach and IDMB+ECBS, did however find solutions more quickly than Auction+ECBS. This was the result from AGV streams towards the infeed stations emerging from the IDMB algorithm. Apart from runtimes, a major aspect in the second experiment was a first assessment of the operational capacity of our hybrid algorithm. It steadily outperformed the other MAPF+TA algorithms as bags were picked up at a faster rate and AGVs experienced less delays. Finally, in the third experiment, longer simulations were run which included charging of AGVs. This showed a clear hierarchy between the three algorithms with our approach significantly outperforming the other two based on service times and throughput.

Our hybrid MAPD algorithm has shown to outperform the MAPF+TA approaches in terms of operational capacity. Despite inferior scalability, runtimes guaranteed the algorithm to be real-time applicable as these were, on average, kept beneath 1 second in a system with 60 AGVs. The decrease in scalability was mainly caused by the required runtimes for IDMB and the baggage hall lay-out. Further research should therefore point out whether IDMB and ECBS-TA could be integrated in a more scalable way and whether our MAPD approach could benefit from other warehouse environments.

References

Barer, M., Sharon, G., Stern, R., and Felner, A. (2014). Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *Proceedings of the Seveth Annual Symposium on Combinatorial Search (SoCS 2014)*, pages 19–27.

Cohen, L., Uras, T., and Koenig, S. (2015). Feasibility Study: Using Highways for Bounded-Suboptimal Multi-Agent Path Finding. In *Conference: Eigth Annual Symposium on Combinatorial Search*. Association for the Advancement of Artificial Intelligence.

Duchateau, W. (2020). Multi-agent pickup and delivery in a distributed baggage handling system, a literature review. Master's thesis, Delft University of Technology.

Gittens, A. (2019). ACIs World Airport Traffic Forecasts reveal the drivers of air transport demand growth on the path to 2040. Accessed in March 2020.

Glidepath (2017). Design of Baggage Handling Systems. Presentation. Accessed in 2019.

Hönig, W., Kiesel, S., Tinka, A., Durham, J., and Ayanian, N. (2018). Conflict-Based Search with Optimal Task Assignment. In Dastani, M., Sukthankar, G., André, E., and Koenig, S., editors, *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018)*.

Kersten, E. (2019). Interview by Circulaire Maakindustrie: Vanderlande - FLEET. `https://circulairemaakindustrie.nl/case/vanderlande-fleet/`.

Khamis, A., Hussein, A., and Elmogy, A. (2015). *Cooperative Robots and Sensor Networks 2015*, chapter Multi-robot Task Allocation: A Review of the State-of-the-Art, pages 31–51. Springer International Publishing.

LaValle, S. and Jingjin, Y. (2013). Structure and Intractability of OptimalMulti-Robot Path Planning on Graphs. *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 27(1):1443–1449.

Liu, M., Ma, H., Li, J., and Koenig, S. (2019). Task and Path Planning for Multi-Agent Pickup and Delivery. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019)*.

Ma, H., Hönig, W., Kumar, T., Ayanian, N., and Koenig, S. (2019). Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery. *Association for the Advancement of Artificial Intelligence*.

Ma, H. and Koenig, S. (2016). Optimal Target Assignment and Path Finding for Teams of Agents. In Thangarajah, J., Tuyls, K., Jonker, C., and Marsella, S., editors, *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*.

Ma, H., Li, J., Kumar, T., and Koenig, S. (2017). Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In Das, S., Durfee, E., Larson, K., and Winikoff, M., editors, *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)*.

Olijslager, V. (2018). Operationalizing a High-capacity Decentralized Baggage Handling system. Master's thesis, Delft University of Technology.

Sharon, G., Stern, R., Felner, A., and Sturtevant, N. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66.

Sharpanskykh, A. (2020). Lecture: Agent-based simulation. analysis of simulation results. model validation. Presentation. Delft University of Technology.

Trigui, S., Koubaa, A., Cheikhrouhou, O., Youssef, H., Bennaceur, H., Sriti, M., and Javed, Y. (2014). A Distributed Market-based Algorithm for the Multi-robot Assignment Problem. *Procedia Computer Science*, 32:1108–1114.

Vanderlande (2019a). FLEET. Accessed in December 2019.

Vanderlande (2019b). Vanderlande. Accessed in December 2019.

Vargha, A. and Delaney, D. (2000). A Critique and Improvement of the "CL" Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educatinoal and Behavioral Statistics*, 25(2):101–132.

Wurman, P., D'Andrea, R., and Mountz, M. (2008). Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1):9–20.

II

Literature Study
previously graded under AE4020

1

Introduction

When a new baggage handling system is being designed in an airport, this system is generally designed to last for 15 years. Considering that the demand for air traffic is growing due to an increasing amount of passengers of almost 4% per year [32], the system should be designed to withstand the peak after those 15 years. This means that a conventional baggage handling system is dealing with a surplus capacity for 14 years since the full capacity is only reached in the 15^{th} year, disregarding peak demands during the period. A very large and expensive system is thus being created to only work at its full capacity for a couple of months.

To cope with this varying demand, Vanderlande aims to create a flexible distributed baggage handling system in which robots handle the required capacity, FLEET. Peak demands can be dealt with by simply adding robots to the system after which they can be retracted and be put in use elsewhere [33]. Similar systems do exist, think about warehousing [5] where robots transport shelves to workers which sort packages for shipment. However, FLEET is dealing with a highly dynamic system where tasks appear randomly and one that is capable of handling varying capacity demands. The robots that carry the bags throughout the airport will be programmed to find their most optimal route, their best task assignment and, not to forget, to take care of separation with other robots. All of this in a distributed way while aiming for flexibility and scalability.

This report contains critical reviews on state-of-the-art multi-agent system methods that deal with task allocation and path planning for robots. Ultimately, trade-offs will be performed to find the most suitable method to fulfil the requirements of FLEET. This literature review begins with looking at baggage handling systems in general in chapter 2. What are advantages or disadvantages of conventional systems and how does FLEET cope with these? Furthermore, a somewhat similar system, Amazon's warehouse, will be presented to see what the commonalities are between both. Since a multi-agent system is considered to be used in FLEET, in which each robot becomes an autonomous agent, the choice for using a multi-agent system is elaborated on in chapter 3. Chapter 4 then introduces the concepts of task allocation and path finding in multi-agent systems. First, both are presented individually, however, there are ways to optimize them both by performing them simultaneously. Two alternatives to combining task allocation and path planning are explained after. The chapter concludes with setting up criteria for the trade-offs and KPIs for the final system. Subsequently, chapter 5 presents algorithms to tackle the combined task allocation and path finding problem. In the same chapter, one of those algorithms will be selected based on the trade-off criteria, improvements will be suggested and taking into account the requirements, a new solution will eventually be proposed. Finally, the research objective and research planning based on the new solution are presented in chapter 6.

2

Baggage Handling

Because of the increasing demand in capacity and customer expectations in aviation, the need for a next-generation baggage handling system (BHS) is growing. Current conventional conveyor systems are not always reliable and are expensive to design because of the space and infrastructure they require. This chapter starts of with elaborating upon current baggage handling operations and the need for a next-generation system in section 2.1. Hereafter, Section 2.2 introduces Vanderlande's idea to automate baggage logistics in airports, BAGFLOW. A part of BAGFLOW is FLEET, a flexible and scalable distributed baggage handling system that aims to eliminate the drawbacks of conventional systems. Finally, since the working principle of FLEET is similar to current warehousing solutions, Kiva Solutions (Amazon robotics) is discussed in section 2.3.

2.1. Conventional baggage handling systems at airports

Since it directly affects airport performance, for passengers as well as for airlines, baggage handling is a crucial process in the activities of a terminal. When the BHS is poorly managed, baggage might be damaged, arriving late or might even be lost, leading to passenger dissatisfaction and damaging public images of airlines [34]. Furthermore, it can lead to significant cost increments: Inefficient baggage handling might induce longer turnaround times, increasing operational costs considerably [1].

Baggage handling starts when a passenger drops off his/her luggage at the check-in and ends when that piece of luggage has entered the aircraft. As can be seen in figure 2.1, three separate baggage handling processes happen: (1) An outbound baggage flow which goes from the check-in to the departing flight, (2) an inbound baggage flow that comes to the baggage claim area from a landed flight and (3) a transfer baggage flow which transports the luggage from transfer passengers between gates [1].

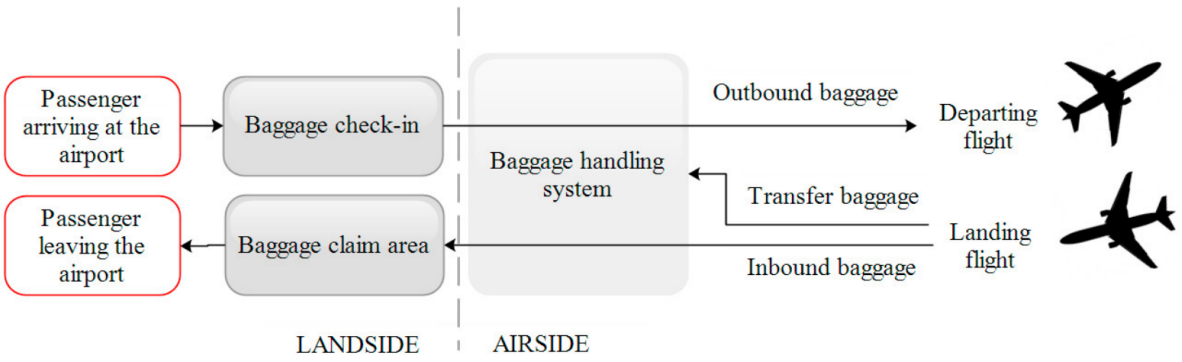


Figure 2.1: Baggage handling in airports: outbound, transfer and inbound baggage flows [1].

When one imagines a baggage handling system in an airport, a whole network of conveyor belts comes to mind. Miles of these are needed to transport one piece of luggage through the airport. Conventional systems

exist of fixed conveyor belts. They require first of all a lot of space with sufficient structural integrity to carry all the systems and have to be specifically designed for each airport.

The size (and type) of a BHS depends on several factors, the most evident being the passenger flow. The passenger flow considers the peak number of passengers and the amount of inbound, outbound and transfer passengers. Also the type of flights that fly from and to an airport are important to know: how frequent they fly, the amount of checked baggage, check-in times and delay tolerance of the airlines. Furthermore, in the design of a BHS, the type of terminal plays a role (it influences how long passengers are willing to wait), together with the security and the project type (whether it is a new system or a modification).

Considering the space needed by a BHS, different requirements have to be met [35]. It should be easy accessible for maintenance, walking space has to be provided for e.g. unjamming bags, structural integrity is required and thus the building has to be adapted to the appropriate structural architecture. Another point is that conveyor systems are expensive and, since the they are fixed, they are not always easy to expand or modify. Moreover, sorting of the luggage can still happen manually in small to medium sized airports and is not ergonomically responsible. The final point is that if a conveyor system fails, the whole process is affected.

In peak periods, these conveyor belts are sometimes being used beyond their capacity and however they are being designed for it, conveyor belt systems have proven to be bottlenecks [1, 5]. To make them more expandable and flexible, Vanderlande has developed an automated baggage logistics solution which is discussed in the next section, section 2.2.

2.2. Automated baggage logistics

As a sustainable and ergonomic alternative for conventional baggage handling systems, Vanderlande has developed FLEET. FLEET consists of robots that handle individual bags and transport them throughout the airport in the most optimal way. Section 2.2.1 presents the overarching baggage processes, BAGFLOW, where FLEET is part of. Hereafter, the operational concept of FLEET itself is thereafter explained more in depth in section 2.2.2.

2.2.1. Vanderlande BAGFLOW

Vanderlande is an international market leading company that is specialized in the automation of logistic processes. It is a supplier for airport logistics automation, for the parcel industry and for automation of processes in warehouses [36]. The focus in this literature study is situated in the automation of baggage handling in airports, which is tackled by their concept BAGFLOW. BAGFLOW is *'a scalable, modular and innovate concept that integrates the complete journey of a bag'* [37]. Its goal is to create a stable system that occupies less space (with respect to the conventional conveyor belt system) while handling more luggage and reducing costs.

One of BAGFLOW's components is FLEET which uses automated guided vehicles (AGVs) to transport individual pieces of luggage inside the baggage hall to their correct drop-off points while supporting screening and sorting.

2.2.2. FLEET

FLEET is a solution for automated baggage handling in airports which is both flexible and sustainable. It uses intelligent AGVs to transport bags from the baggage hall up to the gates and replaces the current conveyor systems as presented in the previous section (section 2.1).

Each AGV in FLEET carries one piece of luggage and determines its optimal way through the airport. Since one AGV carries one bag, the need for total redundancy disappears. If one

Figure 2.2: A FLEET AGV with a small conveyor on top to deliver and retrieve bags to/from a fixed conveyor [2].

of the robots fails and another robot notices this, the robot in question notifies all other robots and the failed robot is considered as an obstacle which is to be avoided during route planning.

Furthermore, FLEET reduces the need for manual handling. Since the AGVs are capable of retrieving bags from a belt, figure 2.2, the AGV's can deliver (i.e. sort) the bags by moving to corresponding drop-offs.

FLEET also adds expandability and flexibility to baggage handling systems. If an AGV needs to go into maintenance, it can easily be removed without having to interrupt the system. Other AGVs can simply continue their work. During periods of higher demand, one can add a robot to the network which will automatically integrate in the system by communicating with the other robots.

Much more benefits are offered by FLEET, a list of the most important ones is provided below [3, 29]:

- **Flexibility:** Adding an AGV and changing routes is easy which adds scalability. In case an AGV fails, other vehicles can simply bypass it.
- **Prioritizing:** FLEET is capable of handling bags with priority.
- **Maintenance:** This follows from *Flexibility*. Since removing and adding agents can be done easily, the system can continue working when an AGV does not function appropriately.
- **Space utilisation:** No additional infrastructure is needed (except for the infeed and chutes) so any open space can be used.

Working principle

Figure 2.3 shows a warehouse-like environment in which the AGVs move around to collect and pick up luggage. In this environment, the AGVs (represented by circles) transport the luggage between the infeed (1) and the chutes (2) while avoiding collisions and deadlocks among each other. While transporting the bags, the robots are required to consider their battery life, these can be recharged at the charging stations beneath the chutes (3) [3].

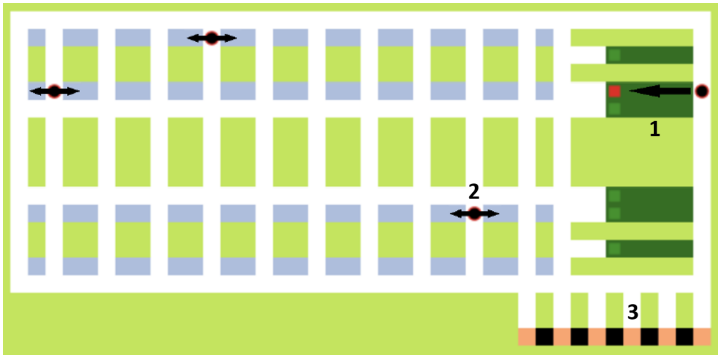


Figure 2.3: FLEET layout as presented in [3].

In the figure below, figure 2.4, three other concepts from Vanderlande are shown. In each of the figures the infeed is shown on the left and the chutes is displayed on the right. In between those two, there is a zone in which the AGVs can move with increasing complexity from the left figure to the right figure with arrows representing directed lanes that are to be respected by the AGVs. In the first figure, the system is rather simple and only has to coordinate the merges at the infeed and the chutes. The middle concept becomes more complex as the AGVs are allowed to take shortcuts and finally, the concept on the right is the most complex one with a free flow zone in the middle. This is the most flexible solution for the AGVs but requires the most coordination [4].

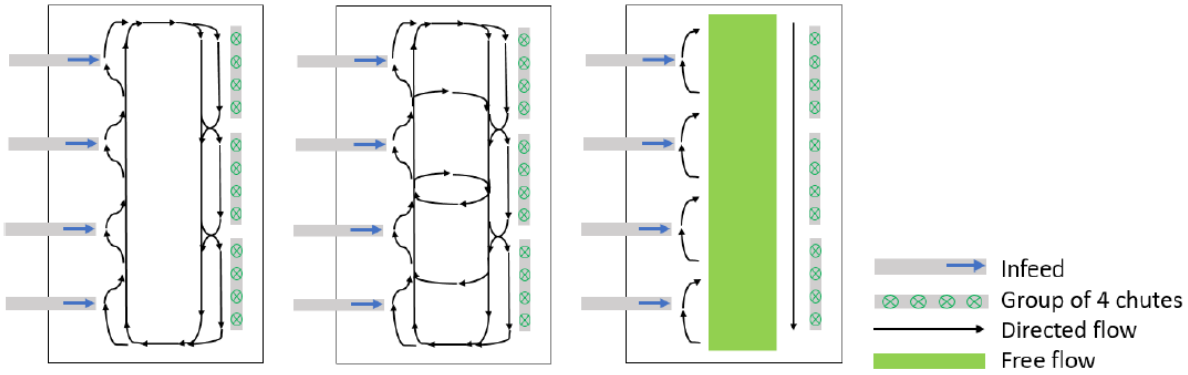


Figure 2.4: Concept layouts for FLEET with increasing complexity from the left to the right [4].

Requirements

From the working principle, several requirements for the system can be deducted that will have to be considered throughout this literature review. A short overview of those requirements is given below.

1. **Spatial arrangement**

   One of the main considerations in the research will be the physical space that is used for the automated luggage logistics system. One can use a warehouse-like layout as shown in figure 2.3 but another approach can be to have an empty physical space between the infeed of the baggage and the chutes where the baggage is delivered, this is shown in figure 2.4.

2. **Charging**

   A second point is that the robots have to charge once in a while. An AGV should thus take into account that it has a sufficiently high energy level to be able to finish its current task and make it to the charging station. This while considering the optimality of the problem.

3. **Priority bags**

   Bags will arrive with different priority levels. For example, a bag for which the aircraft is about to leave soon, think about transfer flights, should find its way to the gate as quickly as possible. In this case, these so-called *hot bags* get priority over other ones.

   The system should thus make sure that prioritized bags are picked up directly and brought to their drop off point without interference with other robots. Two bags with equal priority might still happen to, coincidentally, interfere with each other when on their way to their drop-off points. A way to provide prioritization in close conflicts will have to be provided in the next phase of this research.

4. **Task allocation**

   This item answers to the question 'which AGV takes which bag?'. Depending on the space layout, the AGV energy level, timespan,... it might be beneficial to assign certain bags to specific AGVs or that AGVs wait in line to pick up a piece of luggage. Task allocation will be part of the planning process of the AGV trajectories.

5. **Planning & coordination**

   The last requirement is that the robots have to find paths to their respective tasks. These paths must be so that te global operation is optimized with regard to waiting times for pieces of luggage to be dropped off.

   Coordination also implies that the AGVs have to maintain a certain separation and it has to make sure that no collisions or deadlocks occur between the AGVs.

2.3. Automated warehouse logistics: Kiva Solutions

In 2006, an automated material-handling system for warehouses from Kiva Systems (now Amazon Robotics) became operational. It focussed on the efficiency of *pick-pack-and-ship warehouses*. In these warehouses workers were continuously running around to fetch items from specific shelves to make ready for shipment. Instead of letting the workers move to the shelves, the system made the shelves move to the workers which (at least) doubled the productivity.

This section introduces the Kiva System that is used in Amazon warehouses by first explaining the working principle in section 2.3.1, followed by the benefits that are introduced by it in section 2.3.2. Hereafter, in section 2.3.3, the technique that the Kiva system uses is elaborated upon and finally, section 2.3.4 compares the system with FLEET from section 2.2.2.

2.3.1. Working principle

In the previous section, in figure 2.3, a warehouse-like environment could be seen. In this environment, robots pick up pieces of luggage and drop them off at the requested drop-off point. A comparable working principle and layout can be seen in the Kiva warehouse system in figure 2.5.

In this figure, the ovals (with arrow) represent the robots, the dark grey cells represent storage locations and the light gray cells (on the left) represent the inventory stations [5].

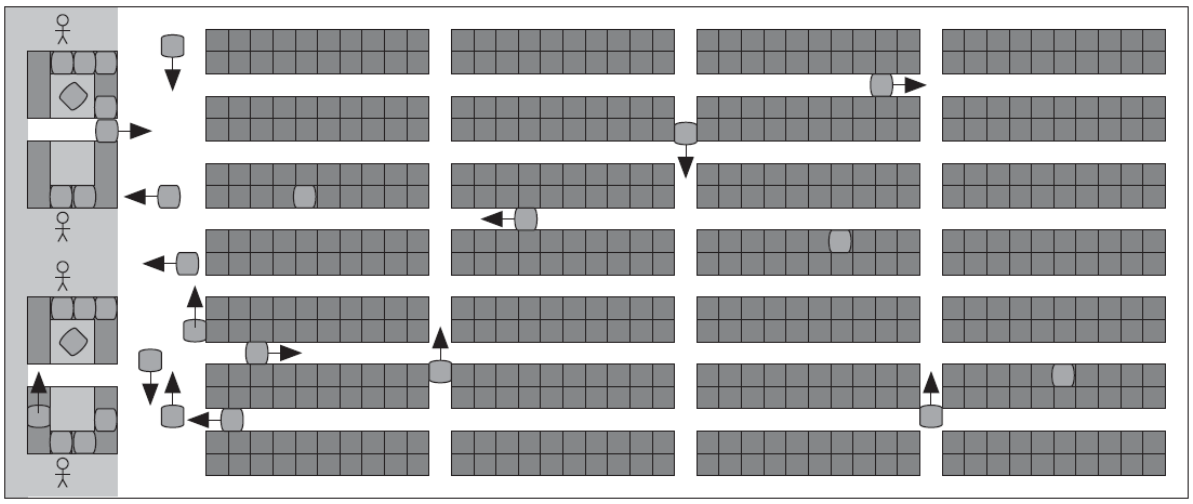


Figure 2.5: Kiva warehouse system layout [5].

The robots in the Kiva system, called drive units, are capable of lifting and carrying inventory pods and transport those pods by driving below them and picking them up. The inventory pods are shelves in which items, that are to be shipped, are stored. The drive units fetch those pods and bring them to inventory stations where workers pick certain desired items and put them into shipping cartons. After this, the robot brings the pod back to a nearby storage location [5].

2.3.2. Kiva solution

Kiva has been introduced to improve productivity and to reduce costs. Since workers do not have to run through the whole complex and 'just' have to wait for shelves to arrive, more work can be done by less people. Besides productivity, Kiva allows for more benefits [5]:

- **Accountability:** Less workers are involved so interdependencies disappear.
- **Batch processing elimination:** Everything can be done in real-time.
- **No single point of failure:** If one robot fails, the whole system is not interrupted unlike a system with conveyors.
- **Spatial flexibility:** The system can work across several rooms without the need of significant adaptions to the environment. Robots just need to be able to pass.
- **Expandability:** When more capacity is needed, one can simply add more robots, pods, inventory stations,...

2.3.3. Kiva techniques

The Kiva system makes use of a multi-agent control architecture that consists of three parties as shown in figure 2.6: the drive unit agents, the inventory station agents and the job manager. In the job manager the system's resource allocation is centralized and it communicates with the warehouse-management system: It receives orders and assigns these tasks to the drive units, the pods and the inventory stations. The inventory stations at their turn communicate with other agents whether a task is requested, received and accomplished [5].

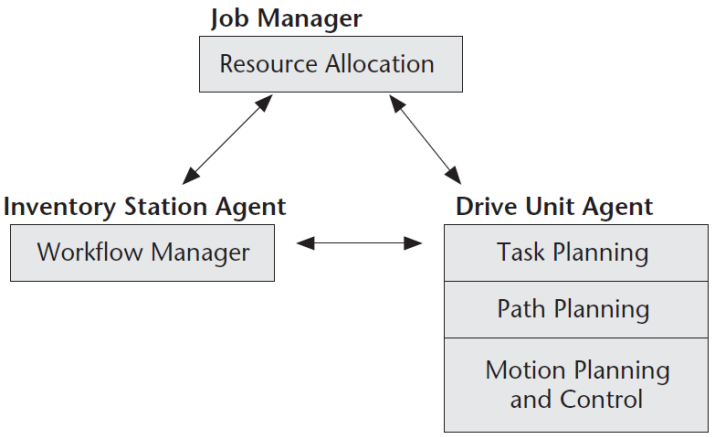


Figure 2.6: An overview of Kiva's multi-agent control system [5].

2.3.4. Kiva vs FLEET

The FLEET concept thus not varies that much from the earlier Kiva system. They both are continuously moving between pickup points and drop-off points, they both want to minimize the time a request (a pod or a bag) has to wait before pick-up and they both require more-or-less the same coordination techniques (collisions, deadlocks, shortest path, task assignment). The coordination technique that is used in Kiva is MAPF (Multi-Agent Path Finding) with A* as shortest path algorithm [20]. MAPF will be introduced and explained together with other coordination methods in respectively chapter 4 and chapter 5.

3

Multi-agent systems

The goal of the upcoming research is to create a multi-agent system (MAS) that provides a solution for a distributed baggage handling system as presented in chapter 2. This chapter begins with an introduction to multi-agent systems in section 3.1. Hereafter, in section 3.2, three different control architectures which a MAS can deal with are presented. Thirdly, the specifications of a multi-agent system are explained in section 3.3 and finally, section 3.4 introduces the planning problem.

3.1. Definition

A multi-agent system is, like the term indicates, a system that exists of multiple agents. These agents are autonomous entities that are capable of making their own decisions by perceiving their environment and acting upon it. In a multi-agent system, a set of agents interact with the environment and with each other to satisfy design objectives that are difficult to solve for single agents. To interact successfully, the agents need to cooperate, coordinate and negotiate with each other by communicating with each other [8, 38, 39].

Looking at FLEET, several agents that are identified at this stage of the research are:

- **AGVs:** The AGVs will need to communicate with other AGVs and with crossing points to obtain coordination. Furthermore they have to communicate with the agents mentioned below to be assigned pieces of luggage and to go charging when they are required to.
- **Infeed stations:** The infeed stations are in contact with AGVs and by means of negotiation, AGVs are linked with a piece of luggage (a task) and a corresponding chute. If a new piece of luggage is available, the station makes the AGVs aware of this.
- **Chutes:** If a chute has received its piece of luggage, it marks that bag as being received and communicates this to the system. The AGV that has delivered the bag becomes available again.
- **Charging stations:** Charging stations need to show whether they are free to use or occupied by a charging AGV.

3.2. Control vs MAS

Figure 3.1 shows three different control architectures. The first figure (A) shows a **centralized** control architecture: one central control unit controls all the agents and the whole operations the system carries out depends thus on this central unit. It is the simplest architecture of the three since the controller has an overall view of the situation and can impose certain coordination relationships. Since all communication runs to one central unit and it has to perform all computations, bottlenecks can arise which makes a centralized control architecture only useful for a small number of agents [39–41].

Since FLEET should be modular and scalable, using a centralized control architecture is not an option. One of the problems that was to be mitigated with FLEET was the dependence on a single controller to avoid the whole system to fail when one component should fail. Furthermore, since centralized systems have proven to rise communication bottlenecks, only a limited number of AGVs can be used resulting in limited scalability.

The second figure (B) represents a **decentralized** system. In this case there still is a centralized control unit but now it operates different groups of agents with each a central agent. The groups of agents are considered as specialists in certain tasks which each creates pieces of the global plan. Decentralized control has the advantage over centralized control that errors in one group do not affect the whole working of the system but only has an impact on other groups that interact with it. On the other hand, a centralized architecture returns optimal solutions while a decentralized one cannot always find one and instead often returns sub-optimal solutions [39, 40].

Finally, in the third figure (C), the most complex system architecture is shown: a **distributed** architecture. In this system, no control agents are present and the problem is divided between several agents that share their knowledge to come up with a solution. The agents cooperate or negotiate with each other in order to create individual plans while moving throughout the system, they become *'processing nodes'*. A disadvantage however is that when asking one agent about the state of the system, one will receive an unclear response. This is because an agent only perceives or knows that what it observes and has no knowledge about the overall state of the system [39–41].

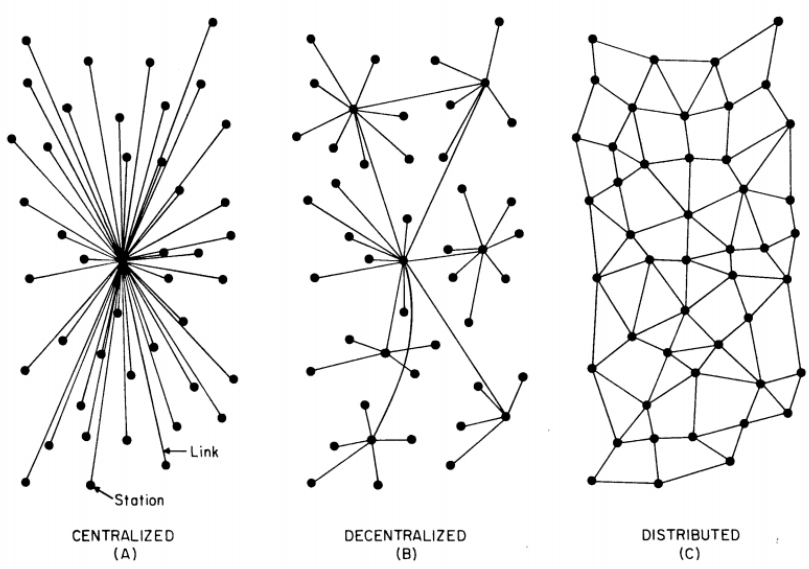


Figure 3.1: Centralized vs decentralized vs distributed control [6].

A decentralized or distributed system will thus be more beneficial for FLEET. Since these are highly flexible and modular, AGVs can easily be taken out or added to the system in case of peak periods of baggage transportation. Moreover, no single point of failure exists since none of the components will be fully interdependent. Also, since communication and computational bottlenecks (centralized controller) are removed, these systems have an improved scalability.

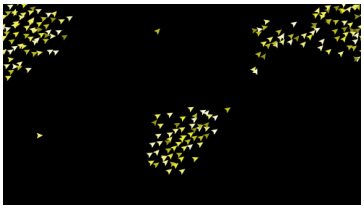


Figure 3.2: Example of emerging behaviour: flocking [7].

The previously mentioned disadvantage in distributed systems (i.e. each agent only perceives a part of the system) brings along a property for multi-agent systems: **emergent behaviour**. An example of emergent behaviour is flocking. In the flocking example in figure 3.2, each agent (bird) is programmed to (1) have a certain separation between each other (avoid collision), (2) to align with each other (move in the same direction) and (3) to cohere (move towards nearby agents). These three rules only have an effect on the agent's heading and no agent is selected as leader. The emerging behaviour, flocking, is thus not programmed in any of the agents but it is a "*global outcome of individual agent coordination*" [7, 42].

In the FLEET model that will be developed, each agent will be programmed to transport bags between the infeed and the chutes while taking into account separation, prioritization, path planning and task allocation. What can emerge from their individual behaviour are for example deadlocks, livelocks or, not necessarily negative, AGV streams.

3.3. MAS specification

Multi-agent systems can be split up in three parts: (1) an environment has to be specified in which the agents are located, (2) the agents have to be defined, how they perceive the environment and how they process gathered information, and (3) interactions between agents (and the environment), i.e. coordination, has to be integrated. Section 3.3.1 explains the properties of an environment, followed by section 3.3.2 in which the behaviour of an agent is presented. Finally, section 3.3.3 elaborates on the interactions between agents and between agents and the environment.

3.3.1. Environment

In a multi-agent system, the environment represent objects that are not agents. Agents observe these objects and act upon them. An environment can be [39, 40, 43]:

- Deterministic or non-deterministic: In a deterministic environment the outcome of a certain action is clear, there are no uncertainties.
- Static or dynamic: A static environment remains unchanged except if it is modified by an agent. When other processes (not the agents) act on it, it is a dynamic environment.
- Accessible or inaccessible: If an environment is accessible, agents can obtain complete, accurate and up-to-date information about its state.

In Delegate multi-agent systems [23] (DMAS, see section 5.2.2), agents can communicate through the environment by leaving traces (pheromones). This is called stigmergy [43].

The environments used (figure 2.3 and figure 2.4) are (1) **non-deterministic:** actions can lead to an undesired outcome, (2) **dynamic:** the model will occasionally let an AGV fail so that lanes are blocked, or dead- or livelocks can occur and (3) **accessible:** AGVs are aware of their environment, its layout and other agents that are present.
The use of pheromones in the FLEET environment might be useful to identify crowded areas. These can be avoided to find a more optimal route.

3.3.2. Agents and their local properties

There are two types of agents: reactive agents and proactive agents. Reactive behaviour is the simplest of the two, namely the agent perceives a change in the environment and acts to that change. Proactive agents possess cognitive abilities, they have a goal in mind and will solve problems, plan, make decisions,... according to that goal. They can process the information (changes in the environment) and react accordingly [43].

Proactive agent architecture: Belief-Desire-Intention (BDI)

Belief-Desire-Intention is an architecture for proactive agents (see algorithm 1). In this architecture an agent has a belief, a desire and an intention. A belief is the information an agent has about the state of the environment, a desire is a goal or a state an agent wants to achieve and an intention is an action that needs to be carried out to achieve that goal or state [43, 44].

Consider a student (the agent) at a university as an example [45]. A student has the **belief** that he has a class tomorrow at 14:00, however classes have actually been suspended. The student thus plans on going to school at 14:00 (**desire**) but he also wants to meet his friends (from another university) at 14:00 (**desire**). The student may **intend** to go to class and can thus not **intend** to meet his friends.

An AGV wants to go pick up a piece of luggage (**desire**). It therefore checks his energy level, calculates the distance to pick up that piece of luggage and drop it off at the corresponding chute, and finds that it still has sufficient energy to perform that action (**belief**). It therefore **intends** to go for that piece of luggage and does thus not **intend** to go charging first.

Algorithm 1: BDI [46]

```
1:  procedure BDI-INTERPRETER(event − queue)
2:      Initialize state
3:      repeat
4:          Retrieve options options from event − queue ▷ Read the event queu and return possible options
5:          Select set of options selected from options ▷ Select a subset of options and adopt them
6:          Update intentions from selected ▷ Add options to intentions
7:          Execute ▷ Execute intentions
8:          Get new events ▷ Add new internal and external events to event queu
9:          Drop successful attitudes ▷ Remove performed desires and intentions
10:         Drop impossible attitudes ▷ Remove impossible desires and intentions
11:     end repeat
12: end procedure
```

3.3.3. Interactions

Agents are required to work towards a common goal. By communicating with each other, directly or through the environment (stigmergy), by being able to understand and process other agents their behaviour or intentions, and by the ability to perform coordination, global goals can be reached in very efficient ways [8].

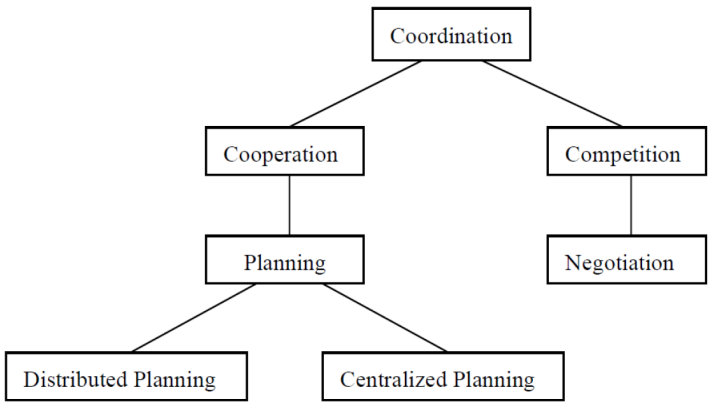


Figure 3.3: Coordination overview [8].

By coordination, wanted states or unwanted states can be achieved or avoided. This can be by cooperating, in which agents work together to achieve a common interest. Cooperation leads to the need for planning since multiple robots can perform multiple actions to contribute to the system. It can be found in for example search and rescue missions, transportation problems (Kiva) and exploration. On the other hand there are competitive environments in which each agent acts out of self-interest, think about games or auctions in which agents want to maximize their own benefit [8, 41]. A scheme of coordination in a MAS is shown in figure 3.3.

FLEET will use a cooperative coordination approach. Since bags should be transported as efficient as possible, the AGVs will need to plan their routes together for the most optimal solution.

- **Agent - Agent:** Agents, at the short term, will have to give way to each other when the plan execution leads to conflicts. Also, they can negotiate[a] with each other on which AGV gets assigned a certain task.
- **Agent - Infeed:** The infeed agents will have to inform the agents that new tasks have become available. Agents, at their turn, have to update the assigned infeed regularly that they are picking up the piece of luggage and have to inform the infeeds when they have become available again.
- **Agent - Chute:** Chutes are coupled to tasks thus the agent and chute communicate when the bag has been delivered.
- **Infeed - Chute:** An infeed should inform the corresponding chute that a piece of luggage is destined for it. The chute informs the system that a piece of luggage has been delivered.
- **Agent - Charging station:** Agents communicate with the charging stations when they note that their battery level is too low. Furthermore, charging stations have to declare their availability to interested agents.

[a] Figure 3.3 indicates that negotiation is a form of competitive coordination However, market-based approaches can eventually be programmed so that the outcome is optimal for the system.

3.4. Multi-agent planning problem

As can be seen in figure 3.3, planning is part of a cooperative coordination environment. The goal of multi-agent planning (MAP) is to find a sequence of actions (i.e. plans for each agent) that leads to a set of achieved goals. During planning, the system has to take into account the initial state of the environment, a set of global goals and a set of individual goals for the agents and their capabilities. The optimal solution would be finding a plan for each agent which makes sure that their individual goals as well as the global goals are met while making sure the individual plans are coordinated [47–49].

MAP consists of six phases [47]:

1. **Global goal/task refinement:** The global goals are refined so that only subtasks remain. These subtasks can be assigned to agents to achieve the global goals.
2. **Task allocation:** The subtasks are assigned to an agent.
3. **Coordination before planning:** Rules or constraints are defined for individual agents to prevent them from producing conflicting plans.
4. **Coordination during planning (individual planning):** A plan is made for each of the agents to achieve their individual goals.
5. **Coordination after planning:** The individual plans are coordinated.
6. **Plan execution:** The plan is executed.

During execution, conflicts might still arise and **replanning** might be required. Numbers 4 to 6 are repeated in this process.

As stated in subsection 2.3.4, the Kiva system uses MAPF to perform path planning. MAPF is a solution for the planning problem that has been presented above. The next chapter, chapter 4, provides a proper introduction to the MAPF problem.

4

Task allocation and path finding

As introduced in chapter 2, FLEET aims to optimally transport bags throughout the airport. 'Optimally' does not only consider the movement of AGVs, finding optimal paths and avoiding conflicts, but also task allocation, where each bag is optimally assigned to an AGV. This chapter introduces three concepts to deal with combined task allocation and multi-agent path finding. Both principles, multi-agent task allocation (TA) and multi-agent path finding (MAPF), are presented in section 4.1. Section 4.2 then introduces the three concepts: task allocation and path finding separately, task allocation and path finding simultaneously (TAPF) and the multi-agent pickup-and-delivery problem (MAPD). In the next chapter, several algorithms for the concepts will be presented. To eventually select the best algorithm out of those, a trade-off will be performed. Criteria for this trade-off are presented in section 4.3. Finally in section 4.4, the key performance indicators for the system are presented to ultimately assess the performance of the distributed BHS model.

4.1. Basic principles

This section explains the two basic principles on which the solutions in section 4.2 are based on. First, task allocation is addressed in which its two categories are explained. Afterwards, path finding in multi-agent systems is introduced generally.

4.1.1. Multi-agent task allocation

A short overview of what is discussed by Khamis, Elmogy and Hussein in [9] is presented. They provide an overview of methods to tackle the Multi-robot Task Allocation (MRTA) problem. The MRTA problem deals with the optimal assignment of robots to a set of tasks while keeping in mind a global goal such as, in this review's context, minimizing the makespan (the time in which that goal is completed) [50, 51]. There are two main directions in the solution approach: market-based and optimization-based approaches.

1. Market-based approaches

Market-based approaches are more known as 'auctions' in which tasks are assigned to a set of bidders (self-interested agents) according to some defined criteria and the bids of those agents. This approach has become popular because of its efficiency, robustness and scalability [9].

Figure 4.1 shows the working principle of the auction design 'Contract Net Protocol' (CNP), a task-sharing protocol in multi-agent systems consisting of four phases. The first phase is the **announcement stage**, in which an agent takes up the coordinating role of auctioneer. The auctioneer announces the tasks that are available for bidding. Secondly, in the **submission stage**, agents can bid on these tasks. Agents calculate a certain value based on the objective function, e.g. path length or time to target, which they communicate to the auctioneer as their bid. The auctioneer processes all the bids that it has received and evaluates them according to an optimization strategy in the **selection stage**. From the bidding agents, one winner is chosen. Finally, in the **contract stage**, the agent with the best bit gets assigned the tasks and the bidding process repeats for the next assignment until all tasks have been assigned [52, 53].

The auction approaches are attractive because they are (1) efficient (utilize local information and preferences of participants), (2) robust (can be decentralized to prevent single points of failure), (3) scalable (communication requirements are mostly manageable), (4) allow for online input (new tasks can be added easily) and (5) able to operate in unknown and dynamic environments.

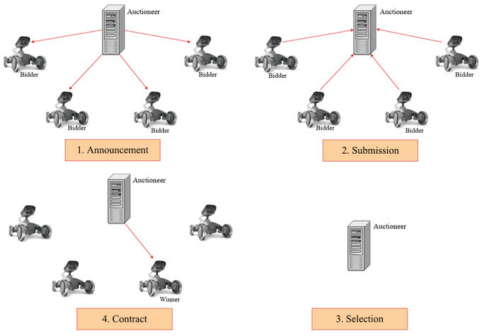


Figure 4.1: Visualisation of the auction process, Contract Net Protocol (CNP), a task-sharing protocol in multi-agent systems [9].

Besides the advantages, the disadvantages also have to be considered. A first one being that bids used in auctions are not always accurate since the cost and revenue functions can not always grasp the design requirement or due to the environment being unknown or continuously changing. Furthermore, as explained in the auction process above, the agents are self-interested. This might lead to the final solution being optimal for the agents involved but it may not be optimal for the whole operation.

2. Optimization-based approaches

As opposed to market-based approach, optimization-based approaches do keep the global goal directly in mind when solving a problem. The optimal solution follows from an objective function which quantifies the global goal of the system and which needs to satisfy a set of constraints.

There are two main optimization techniques, deterministic and stochastic techniques, figure 4.2. The key difference between those two is that stochastic techniques have randomness in their solution while a deterministic approach would always lead to the same solution.

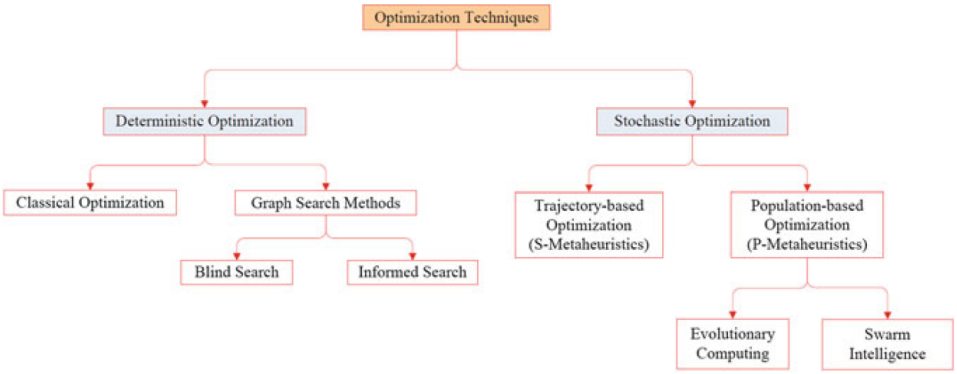


Figure 4.2: Optimization techniques [9].

4.1.2. Multi-agent path finding

In MAPF problems, the goal is to find coordinated paths for a set of agents so that each agent reaches its goal without collisions. It has been studied in several applications, think about warehousing [5] as mentioned in subsection 2.3.4, strategy games [54], industrial automation [55],...

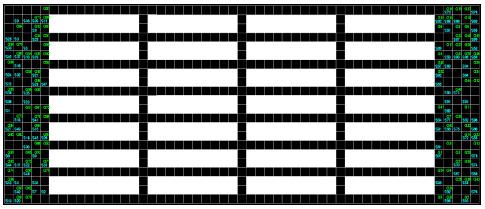


Figure 4.3: Simulated warehouse environment with blocked (white) and unblocked (black) cells [10].

Consider the following: In figure 4.3 a warehouse environment is shown (corresponding to the Kiva warehouse system in figure 2.5) with blocked and unblocked cells. The blocked cells, in white, represent the shelves, whereas the unblocked cells, black, represent corridors for robots to move through. Each robot gets assigned another unblocked cell as goal and every time step (discrete), the robots can move to a neighbouring, unblocked cell or can wait at their current cell. MAPF methods aim to optimize the trajectory of each robot so that no collisions, i.e. two robots in the same cell at the same time step, occur. Optimization can be for example with regard to minimal makespan or to minimize the sum of steps of each robot until

reaching their goal cell [10].

4.2. Solutions

Solving problems in which both task assignment and path finding have to be performed can be done in three manners. First of all it can happen separately, TA + MAPF, which first solves the task assignment problem and subsequently plans paths for the robots according to these tasks. The second proposed solution is TAPF, in which TA and MAPF are combined. Thirdly, MAPD addresses the full problem in which robots have to attend to a continuous stream of tasks.

4.2.1. TA + MAPF

The simplest way in which FLEET can tackle its transportation problem is by first assigning tasks to AGVs by using one of the task allocation approaches as presented in section 4.1.1. When each task has been assigned optimally, a MAPF algorithm is run to provide collision-free paths for each robot, first to its assigned tasks and then to the corresponding goal.

Typical approaches are thus to keep the two phases, task allocation and path finding, separate: First having to generate all possible assignments and then solve the path planning for each of those assignments. This leads to difficulties with (1) scalability, since the search space becomes too big when generating all possible assignments and planning paths for all of those, and (2) solution quality, due to the two phases not being related there might exist certain task assignments that result in fewer collisions when both are optimized considering each other [11, 12].

4.2.2. TAPF

TAPF tackles the aforementioned scalability and solution quality problem by combining task allocation and path finding. Agents are assigned to tasks and collision-free paths (to their tasks) are then found so that eventually the makespan is minimized [11]. It is a generalization of the anonymous and non-anonymous MAPF problem. In the anonymous MAPF problem, any agent can perform any task. In the non-anonymous version, each agent can only be assigned to one task.

4.2.3. MAPD

However TAPF seems a good solution for the concept this research deals with, there is one key characteristic that is missing: FLEET handles a continuous stream of arriving bags that have to be transported and TAPF only deals with instances where there is an equal amount of tasks and robots. When AGVs have reached their goal, they are supposed to go back to the infeed to collect and transport a new piece of luggage. This is handled by the multi-agent pickup and delivery problem, it is a "lifelong version of the MAPF problem" [56].

In an MAPD problem, a task set exists to which new tasks, including pickup and delivery locations, are continuously added. Every free agent can get assigned to a task for which it plans the path to the pickup point and the drop-off point. While moving to the pickup location of a task, agents can already be assigned to a new task that it can execute after the current task has been delivered. Furthermore, tasks can be reassigned so that the last agent that gets assigned a task has to execute it and any agent that was previously assigned to it, is no longer obliged to do so. Individual goals of the agents are to, as quickly as possible, finish executing a task. The global goal of the MAPD problem is to ultimately reduce the average service time of the tasks. This is the time from appearing in the system to being delivered [56].

There exist two types of MAPD problems: online and offline problems. The difference between them is that in online problems, tasks can only be assigned when they appear (they appear randomly), while in offline tasks, it is known beforehand when a task will appear so AGVs can plan a sequence of tasks to perform.

One can see that a MAPD would be the best option to use as a means for coordination in FLEET since AGVs will have to attend to a continuous stream of bags (online) that have to be transported throughout the airport. TAPF, however, will not be ruled out since it might deliver ideas to integrate both task allocation and path finding that also has to be performed in MAPD problems. In the next chapter, several TAPF and MAPD algorithms will be presented and assessed based on the criteria from section 4.3.

4.3. Trade-off criteria

To ultimately select a method that will be used in the research, criteria are established to create a trade-off matrix in which each of the methods from chapter 5 will be assessed. The method with the highest score will considered to be the best method for coordination and planning in the distributed BHS. Each of the criteria will be given a score from 1 to 3 in the next chapter.

First of all, the **performance** of the methods will be assessed. This includes the following criteria:

- **Scalability:**

  The scalability of an algorithm is how well it behaves when the amount of agents increases. Since during the peak hours more baggage has to be handled, it might be desired to increase the amount of AGVs in the system (high capacity and throughput). On the other hand, at off-peak hours, few AGVs are desired. FLEET should thus be able to use both small and large amounts of AGVs without a significant difference in runtimes and success rates.

- **Environmental flexibility:**

  This includes the algorithm's behaviour when the physical infrastructure is changed. When the space grows smaller or larger, or when less or more lanes are being used, the algorithm should still be able to deliver real-time usable results.

- **Real-world application:**

  Runtimes should be considerably low (< 1 second [56]) to allow for real-time applications. This means that the system will not lie still when recomputations need to be performed. FLEET should be able to run in real-time since constantly new pieces of luggage are arriving.

- **Success rate:**

  The success rate relates to how many times, i.e. percentage, an algorithm delivers a conflict-free result. Algorithms with higher success rates are able to find sets of conflict-free paths more often than those with low success rates. Not finding solutions only leads to system delays and thus affects the real-time application.

Secondly, the **quality** of the methods needs to be graded:

- **Solution quality:**

  The solution quality considers the amount of conflicts that has to be resolved by the algorithm.

- **Makespan:**

  Most path finding algorithms optimize with regard to the solution cost, i.e. the cost of the paths, while task assignment mechanisms mostly consider makespan as their optimization objective. The makespan is the time it takes for all tasks to be delivered, from the first one entering to the last one being delivered. Alternatively, service times can be optimized. The service time considers individual tasks, the time between a task entering the system and that task being delivered.

- **Optimality:**

  Some algorithms guarantee optimality whereas others cannot and subsequently result in suboptimal solutions. Suboptimal solutions can however at their turn be more beneficial in terms of for example runtimes.

Finally, the **implementation** of the algorithms in a programming environment must be considered:

- **Complexity:**

  The complexity is the ease of the implementation of the method which will be a very important, time-consuming and risky process. To rate the complexity, pseudocodes in literature and implementations, if available, will be used. The less complex, the higher the score will be.

- **Availability:**

  Availability addresses previous implementations that are publicly available on for example GitHub and again whether pseudocodes are presented in literature.

4.4. KPIs

When the system eventually will be implemented, measures are needed to analyse the performance of the system. The key performance indicators (KPIs) will be split up in KPIs for task allocation, KPIs for path planning and KPIs for the system performance.

4.4.1. Task allocation KPIs

The most important performance indicators for task allocation are service time and makespan which both consider the frequency at which bags are picked up and delivered. Furthermore, it will be checked whether the task allocation method is able to consider the battery life of the AGVs so that AGVs would not suddenly notice that they cannot make it anymore to their delivery point. The final measure is how long AGVs are not moving to or executing a task.

- **Service time *(low)*:** The time it takes to deliver a piece of luggage, from entering the system to being delivered.
- **Makespan *(low)*:** The time it takes for all luggage to be delivered, from the first one entering the system to the last one being delivered.
- **Battery life shortage *(low)*:** How often an AGV had to go charging while executing a task.
- **AGV idle time *(low)*:** How often or how long AGVs are not moving to or executing a task.

4.4.2. Path planning KPIs

The path planning KPIs consider the paths AGVs follow, both individually and cooperatively. The first one indicating the average speed at which they can move to and from tasks and the deviation from their shortest path. The latter addresses conflicts that occur, i.e. when minimum separation is violated, and how often AGVs get involved in deadlocks. At this point it is assumed that deadlocks, once they occur, can be resolved so that the system can continue operating.

- **Conflict frequency *(low)*:** Amount of conflicts that have occurred (i.e. minimum separation violation).
- **Deadlock frequency *(low)*:** How often AGVs experience a deadlock.
- **Average detour *(low)*:** The average difference between the shortest path and the actual path of an AGV.
- **Average speed *(high)*:** The AGV's average speed during pickup and delivery.

4.4.3. System KPIs

Finally, the performance of the system itself. This considers two measures, namely the amount of AGVs the system can handle and the runtime of the system. The runtime is important because it determines whether the system will be usable in real-time.

- **Capacity *(high)*:** The amount of AGVs the system can handle.
- **Runtime *(low)*:** How long AGVs are idle due to the planning/allocation process.

5

Coordination and planning

This chapter determines which method will eventually be used in the distributed baggage handling system. Section 5.1 lays out different TAPF and MAPD algorithms as determined in Chapter 4. All algorithms are assessed and a trade-off is performed to select one to continue with. The shortcomings of the chosen algorithm, i.e. what to improve, are then identified and solutions for these are proposed in section 5.2. Section 5.3 finally addresses requirements, from Vanderlande or self-imposed to benefit coordination, that are not yet covered by the selected solution.

5.1. Simultaneous task allocation and path finding

This section describes the research selection: it is a trade-off between algorithms that combine task allocation and path finding is performed. Section 5.1.1 introduces three TAPF algorithms, section 5.1.2 presents five MAPD algorithms in which robots are continuously moving between pickup and drop-off points and the selection of the best algorithm is discussed in section 5.1.3.

5.1.1. TAPF

When task assignment and path finding have to be combined, the two are mostly calculated separately without considering that adjusting them to one another might benefit the whole system. So by planning and assigning tasks simultaneously, the operation might lead to better solution qualities and/or require less runtime. Three solution algorithms are presented in this section: the Conflict-Based Min-cost-flow (CBM) algorithm , Conflict-Based Search (CBS) with task assignment and thirdly, the improved version of the latter, Enhanced CBS (ECBS) with task assignment.

1. CBM

[11] presents the Conflict-Based Min-Cost-Flow (CBM) algorithm, an optimal solution for the TAPF problem which combines ideas from both anonymous and non-anonymous MAPF algorithms. It is a hierarchical algorithm that on the high level solves collisions and on the low level assigns agents to targets and plans paths considering constraints that follow from the high level so that no collisions occur.

The **high level** algorithm uses CBS (see section 5.2.2), a form of best-first search on a binary tree where each node contains constraints and paths for all agents. These are obeyed and move all agents to their unique targets while avoiding any collisions. The **low level** algorithm on the other hand assigns each agent to a unique target using a max-flow algorithm and finds the paths, taking into consideration the constraints that have been established in the high level. Paths are found with the help of edge weights to bias the search since running times of CBS (high-level) can be exponential with the amount of collisions that have to be solved [11, 57].

To see how CBM (weighted) performs, four experiments have been carried out: (1) comparison with other solvers, (2) comparing different team sizes, (3) assessing scalability with number of agents and (4) the application to a warehouse system. These experiments have shown that CBM outperforms MAPF algorithms (CBS and ILP, with random task assignment) in both **scalability and solution quality**. Furthermore, it **can be generalized** to applications with multiple teams and hundreds of agents.

In the final experiment, CBM was applied to the Kiva warehouse system [5]. 50 TAPF problems had been generated (see figure 5.1) with each 420 AGVs (7 teams of 30 *delivering* AGVs and 7 teams of 30 *picking* agents) for which the CBM method should plan collision-free paths so that eventually the makespan was minimized. The results showed that CBM solved 40 (80%) of the 50 simulations within the 5 minutes time limit with a mean makespan and running time (over those 40) of 64 and 92 seconds respectively. Comparing this to [20] (section 5.2.2), in which MAPF methods are applied to Kiva and shown not to scale well, CBM shows that it is a promising TAPF algorithm that can be applied to **real-word applications**.

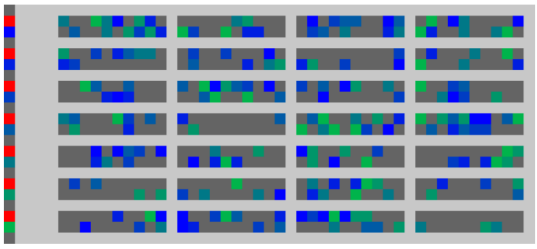


Figure 5.1: Randomly generated TAPF (Kiva) problem [11].

CBM is a first and promising algorithm that could be implemented in FLEET. It uses conflict-based search (CBS), which is considered to be one of the best path finding methods available. It therefore serves as a good baseline to compare the coming algorithms. CBM has the advantage that it aims to minimize the makespan while making use of a path finding solution that normally optimizes problems according to path cost.

2. CBS-TA

Because of the extendability and problem solving abilities of CBS, the authors of [12], have created an extension of the CBS framework. Just like CBM, CBS with Task Assignment (CBS-TA) aims to solve TAPF problems by *"jointly optimizing the task assignment and path planning for very dense cases"*. The two key ideas of CBS-TA are: (1) working in a search forest (instead of a search tree) and (2) creating that forest on demand, i.e. not expanding all possible task assignments. CBS-TA is shown to be complete and optimal and, furthermore, the authors have implemented ECBS-TA, its suboptimal version. In comparison with CBS (section 5.2.2), CBS-TA only needs to adapt the high level phase. It receives two additional node specifications: (1) a root node identification, whether the node is a root node or not, and (2) an assignment identification, which identifies the task assignment that is currently used in the low-level search.

As mentioned, CBS-TA works with a search forest. It starts of with a single root node that uses the best task assignment, while ignoring possible conflicting agents. Every time a root node is expanded, a new root node is created with the next best task assignment. The idea of the next best task calculation is based on [58] but instead of calculating a set of best assignments, the solution is calculated on demand. The idea of the algorithm is that some assignments are prevented to be included and other are forcefully being included. Consider the following example from [12]:

Figure 5.2 (a) shows the example environment where agents 1 and 2 are located at nodes a and b respectively with task locations c, d and e. The same figure shows (corresponding colours) that agent 1 can be assigned to d or e and agent 2 to c or e. The corresponding graph is shown in (b) with nodes a and b being start nodes and c, d and e being goal nodes. A cost matrix can then be created which displays the path lengths from each start node to each goal node as can be seen in figure 5.3. Given this cost matrix, any assignment algorithm (e.g. Hungarian method) can be used to obtain an optimal task assignment.

According to the example in [12], (1.) the first task assignment sends agent 1 to d and agent 2 to c (costs 3 and 1 respectively) as displayed in figure 5.2 (c). The latter corresponds to the first root node in figure 5.2 (d) but the path validation (2.) returns a conflict between the two agents at the second time step. Whenever a root node is expanded, a next best assignment and a new root node have to be added (3.). The first new assignment expansion prevents 1-d from happening and the second prevents 2-c from happening and enforces 1-d. (4.) The shortest path is again calculated

$$C = \begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} c & d & e \end{array} \\ \begin{pmatrix} \infty & 3 & 4 \\ 1 & \infty & 3 \end{pmatrix} \end{array}$$

Figure 5.3: Cost matrix according to figure 5.2 [12].

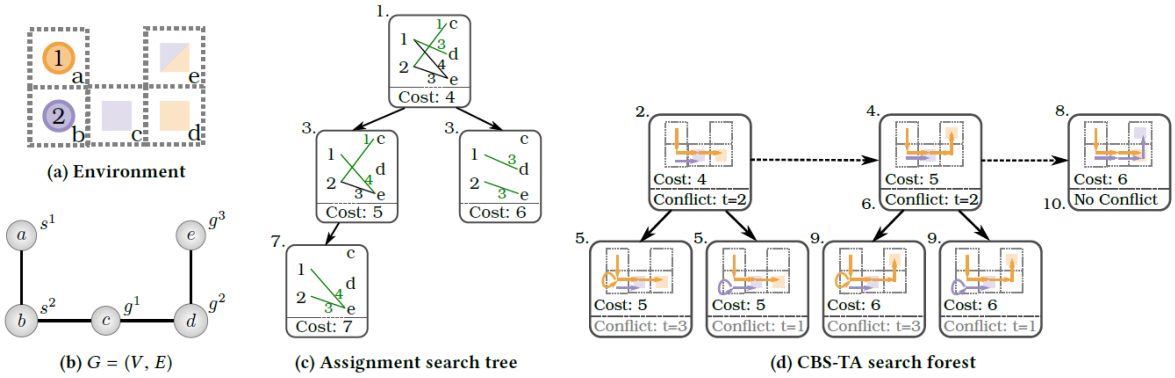


Figure 5.2: CBS-TA example. The environment and corresponding graph in, respectively, (a) and (b) with start and goal nodes.. [12].

following the new task assignment, again leading to a conflict in time step 2. At this point (5.) the first conflict is tried to be resolved by imposing constraints on when each agent is allowed to be at position c. A new root node has to be expanded (6.) and the process starts again by expanding the assignment tree (7.), adding a new root node (8.) and trying to resolve the previous conflict (9.) obtained in (4.). Since the third root node does not contain conflicts (10.), this solution is returned.

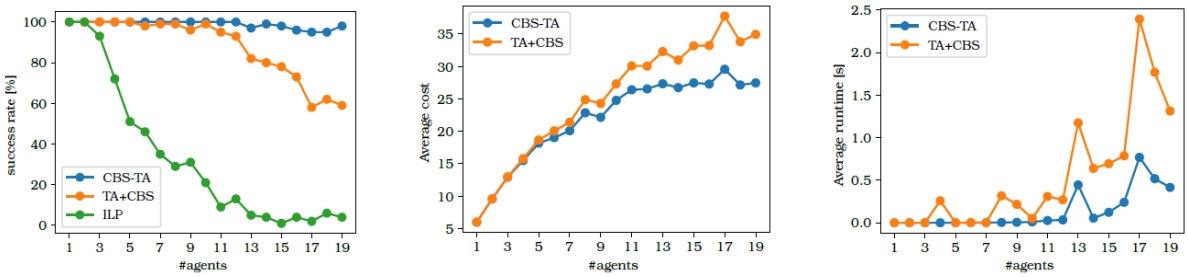


Figure 5.4: Performance of CBS-TA w.r.t. TA+CBS regarding succes rate (left), average solution cost (middle) and average runtime (right) against the number of agents [12].

The performance of CBS-TA is compared to TA+CBS, task assignment before CBS. Figure 5.4 speaks for itself. CBS-TA stands out in **success rate**, average **solution cost** and average **runtime**. One can see that until a certain point, the curves CBS-TA and TA+CBS stay together but when more and more agents are added, the performance of TA+CBS seems to diminish. This shows that CBS-TA behaves particularly well in **dense areas**, where task assignment and path finding ar tightly coupled [12].

CBS-TA has shown that combining both MAPF and task assignment performs better than having them separate: success rate, solution cost and runtime were better in the combined version. Since ECBS has already proven to outperform CBS in terms of path finding, as will be presented in section 5.2.2, CBS-TA has not been compared to CBM, this is left for ECBS-TA in the following part.

3. ECBS-TA

As stated in section 5.2.2, ECBS uses focal search in the high- and low-level search algorithms of CBS. The nodes in the FOCAL list are entries with a cost that is within a certain factor w of the lowest cost in the OPEN list which contains all nodes that are expanded. The focal search looks for the entry in FOCAL with the lowest secondary heuristic, the estimated amount of conflicts. Thus, the high-level FOCAL list only contains those entries which satisfy the w factor constraint, the low-level search uses the amount of conflicts for expansion.

Since the search becomes more flexible due to suboptimality bound w, more options arise on when to add an additional root node, [12] considers three. First there is the *CBS-TA style* option in which each time a node is expanded, a new root node is generated. Secondly, the *MaxRoot*, is an option that adds as many rood nodes

as might be useful for a given w. This means that all nodes with a cost lower than factor w times the lower bound ($w \cdot \min LB(n)$) are added, with the lower bound being the highest cost of the already expanded nodes. This method is however not very practical for large problems since it might produce too many potential assignments. Finally, opposing to *MaxRoot*, also *MinRoot* exists. This option wants to minimize the amount of extra generated root nodes by only generating a new one when a node has a cost within $w \cdot \min LB(n)$ where the lower bound is the lowest cost found in the high-level OPEN list.

ECBS-TA (CBS-TA style) is, just like CBS-TA, compared to task assignment followed by ECBS (TA+ECBS) and additionally to ECBS-TA(MinRoot) in both small (8x8 4-connected grid and up to 19 agents) and large environments (32x32 4-connected grid and up to 100 agents). Comparing both ECBS-TA variants to ECBS in the **small environment**, ECBS-TA achieves higher succes rates, lower solution costs and lower runtime compared to TA+ECBS when using the same suboptimality bound w. In the **large environment**, when comparing the two ECBS-TA variants, the CBS-TA style version results in lower costs for higher values of w. The optimal solution ($w = 1$) was tried to be calculated with all three algorithms resulting in a significantly lower success rate for both ECBS-TA variants. When the suboptimality bound increased, their success rates improved to a comparable level and the solution costs became almost identical to that of TA+ECBS. The runtimes however, did differ quite a bit: the runtime of ECBS-TA CBS-TA style version was significantly higher than the TA+ECBS and the MinRoot versions which at their turn resembled with a slight favor for MinRoot. This significant difference can be explained because of the amount of root nodes are expanded in the CBS-TA style version since the amount of possible task assignments grows factorial with the number of agents.

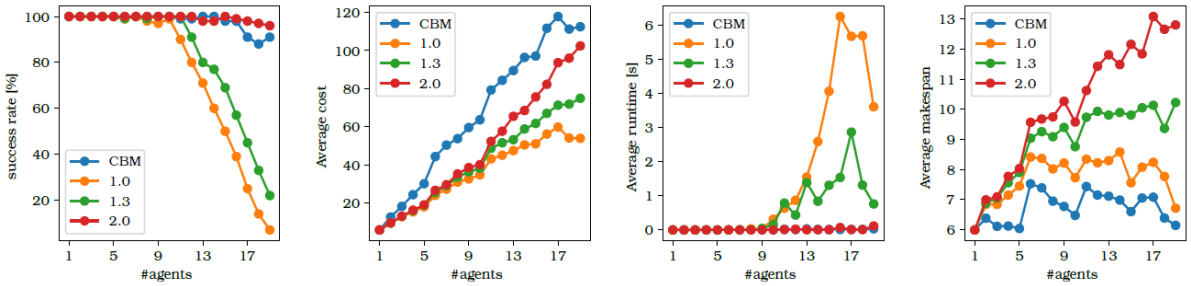


Figure 5.5: Results for the experiment in the small environment with increasing suboptimality bound w in function of the amount of agents [12].

Finally, ECBS-TA is compared with CBM in the small and large environment with team sizes of 5 and varying amounts of agents and thirdly in the large environment with 100 agents and varying team sizes. Figure 5.5 shows the results for the **small environment** experiment in which the success rate, average cost (sum of individual agents), average runtime and average makespan of CBM and ECBS-TA(MinRoot) are compared for different values of w and varying amounts of agents. Recalling that the CBM method optimizes instances with regard to minimum makespan, it shows that CBM does perform best in terms of this metric. Since solution cost and makespan cannot be optimized simultaneously [59] and ECBS-TA optimizes according to the sum of individual costs (which behaves better when minimizing idle times when human workers are in the loop), ECBS-TA is shown to be performing better in terms of **solution cost**. Lowering the suboptimality bound does reduce solution costs but also makes the success rate drop and increase the runtime significantly.

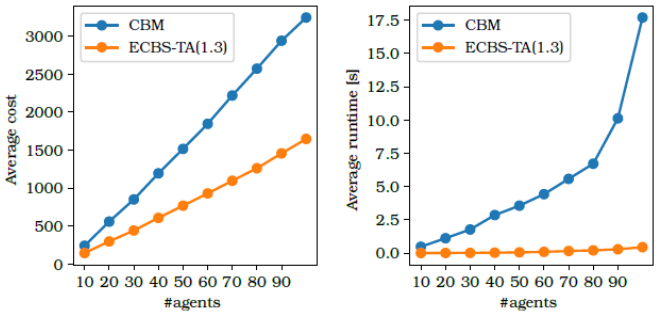


Figure 5.6: Results of the CBM and ECBS-TA algorithms in the large environment [12].

Considering the **large environment**, tests were performed for a suboptimality bound factor of 1.3 since only from this value on, ECBS-TA could **solve all instances**. The results of both algorithms in terms of average solution cost and average runtime are shown in figure 5.6. The figure on the left shows that the average cost achieved by CBM is double the one of ECBS-TA. This result is quite remarkable since the suboptimality is rather high but on the other hand, not shown in the figure, CBM delivers a lower makespan (averagely 33

against 47 for ECBS-TA). The right figure shows the significant benefit ECBS-TA has in terms of the **runtime** where that of ECBS-TA only increases very slightly while that of CBM does not seem to handle the amount of agents well.

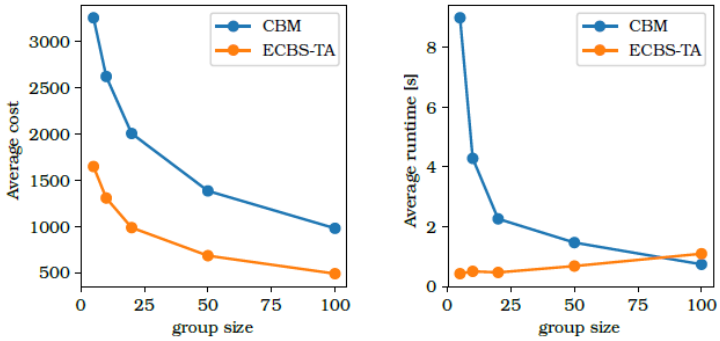


Figure 5.7: Results of the CBM and ECBS-TA algorithms in the large environment with varying team sizes [12].

The last experiment, varying the team sizes, returns the results in figure 5.7. The group size starts at 1 agent per team. This means that each agent has a specific task that is assigned to it. When a team has 100 agents, the team can assign the tasks themselves among each other. That is where CBM performs best with its max flow algorithm. On the other hand, ECBS-TA considers one possible assignment when there are 100 teams and 100! when there is only 1 team. This can be seen in the second figure where the runtime for ECBS-TA increases while that of CBM decreases drastically. The left figure shows again the average solution cost being in favour of ECBS-TA but makespan is smaller for CBM.

(E)CBS-TA has been shown to be a promising algorithm when task assignment and path planning have to jointly be optimized. Due to the flexibility of the CBS framework, it can be directly applied to other domains, however it optimizes with regard to the sum of costs, which might not always be favourable in certain domains. Especially in small environments with few agents, its runtime and solution quality perform very well but large environments seem not to benefit too much from a joint task and path optimization [12].

> ECBS is a suboptimal variant of CBS, it delivers less optimal solutions but at lower runtimes and higher success rates. Comparing ECBS-TA with CBM shows that ECBS-TA has the advantage of being more scalable. However, ECBS-TA optimizes with regard to solution cost instead of makespan and since FLEET does not have human workers in the loop, optimizing with regard to makespan is preferable. The latter allows thus for a better score for CBM regarding solution. A bonus is that implementations of ECBS-TA are available online (GitHub).

5.1.2. MAPD

TAPF problems do not capture an important characteristic of real-life path planning applications: the problem stops when each agents has delivered its tasks. However, agents often have to attend to a stream of different task which they each time have to pick up and deliver. This is what the Multi-Agent Pickup-and-Delivery (MAPD) problem aims to tackle, it is a lifelong version of the MAPF problem [56]. Three papers will be discussed, two which deal with MAPD in an online setting and one in an offline setting. In an online setting, agents only become aware of a task when it has entered the system, in an offline setting however, agents know beforehand when tasks will enter.

1. TP & TPTS (online)

The first paper [56] introduces two decoupled MAPD algorithms: (1) Token Passing (TP) and (2) Token Passing with Task Swapping (TPTS). Both methods are based on existing MAPF algorithms and it is shown that they are able to solve *well-formed* MAPD problems. In a *well-formed* infrastructure (see figure 5.8), endpoints are so distributed that a robot standing on an endpoint can never fully prevent other robots from reaching their endpoints [13].

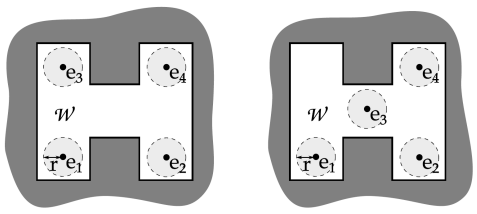


Figure 5.8: A well-formed (left) and a not well-formed (right) infrastructure where in the not well-formed infrastructure an endpoint blocks the way for other agents [13].

The approach in **Token Passing (TP)** is similar to that in Cooperative A*, in which each agent plans its collision-free route based on the knowledge on other agents, one after the other

[60]. The token is a shared block of memory in which all current paths of agents, the task set and agent assignments are stored. Token passing has been used previously in [13] where COBRA was introduced, a MAPF-like algorithm which deals with a system in which the user can at any time oblige robots to move from their current location to another one. However, COBRA does not take into account that pickup or delivery locations can be occupied by other robots (not executing a task), resulting in deadlocks [56].

Token passing works as follows: (1) The token is initialized with starting positions of each agent, then (2) at each time step new tasks (if any) are added. (3) When an agent has reached the destination of its path in the token, it requests the token once per timestep. (4) The system sends the token to each agent that requests it, one by one, and when an agent possesses the token (5), it chooses a task from the task set and (7) returns the token and moves along its designated path. The task is chosen so that the pickup or delivery point of the chosen task is not an endpoint of a path in the token. If the latter exists in the task set, the agent chooses that task which has the lowest path cost. If no such task exists, the agent makes sure it does not create deadlocks by avoiding paths of other agents and by moving away from other pickup or delivery points .

Token Passing with Task Swapping (TPTS) tries to make TP more effective by allowing agents to 'steal' a task that is assigned (i.e. not yet being executed) to another agent if beneficial for the makespan and/or service time (average number of timesteps needed to finish a task, measured from when the task has entered the system). The task set changes from containing unassigned tasks to containing unexecuted tasks. The main working principle of TPTS stays the same but additionally if an agent finds that its path cost to an assigned task is lower than the path cost of the assigned agent, it will unassign that agent and assign itself to the task.

The performance of both algorithms is compared to a centralized algorithm (**CENTRAL**), also developed in [56]. Since in a centralized algorithm more communication is possible, CENTRAL was expected to be more effective (better solution quality) but less efficient (increased runtimes). CENTRAL starts with assigning endpoints to all agents and then solves the MAPF problem, in which each agent has to move to their assigned endpoints, using CBS [57]. It should be noted that the MAPF part has become significantly more efficient by dividing it in two steps: First plan paths for agents that have been assigned a task in the current time step (while treating the other agents as moving objects) and then planning paths for free agents to their assigned endpoints (again while treating the other agents as moving objects).

In the experiment, a sequence of 500 tasks was generated with different task frequencies (0.2-10 per timestep) and different numbers of agents (10-50). The comparison of the three algorithms is based on the makespans, service times, runtimes and scalability. It showed that the CENTRAL algorithm performed best on **makespan and service times** (and TPTS better than TP). On the other hand, on **runtimes**, CENTRAL performed significantly worse (below 10 ms, 200 ms and 4000 ms for respectively TP, TPTS and CENTRAL). Finally, the **scalability** test showed that both TPTS and CENTRAL did not allow for real-time operations for large numbers of agents due to too long runtimes, TP with 200 agents had a runtime of 500ms (allowing for real-time operations). Note that the latter was tested in a large warehouse environment with 1000 delivery tasks.

> Since TPTS and CENTRAL did not allow for real-time operations, these two mechanisms are not considered in the trade-off table table 5.1. TP has shown to be scalable while keeping runtimes down and solves all well-formed MAPD problems.

2. TP-SIPPwRT (online)

In [61], Token Passing (TP) is made more efficient by applying another search algorithm, Safe Interval Path Planning with Reservation Table (SIPPwRT), for single agent path planning. This novel algorithm allows for fast updates and lookups of current agent paths, furthermore it can compute forward movements and point turns continuously with given velocities. **TI-SIPPwRT** is complete for well-formed MAPD problems.

The reservation table in SIPPwRT handles continuous agent movements with given velocities. It keeps a priority queue per cell in which reservations with the lowest bound (time-wise) have the highest priority. It allows to calculate safe time intervals, add reservations when a new path is calculated and remove past reservations to limit the size.

TP-SIPPwRT is compared to TP and CENTRAL from [56] and again tested in a simulated automated warehouse with 30 agents. It shows that it has smaller **service times** en **makespans** than TP and CENTRAL while

having a higher troughput, making TP-SIPPwRT thus more effective.

The paper compares TP-SIPPwRT to TP and CENTRAL in only 1 scenario (with different velocities for task-executing robots). However, no comparison on scalability is made and on performances with less or more robots. Looking at the results in [56] and [61], it seems that TP-SIPPwRT (with uniform speed) scales worse than the three methods discussed in the previous section with higher runtimes, makespans and service times. The difference with the results in this paper could be explained by the fact that in [61], TP and CENTRAL had to be post-processed to account for robot sizes and velocities and TP-SIPPwRT being centralized.

> However delivering smaller makespans and service times, TP-SIPPwRT did not seem to behave better than TP in terms of runtime and its scalability was not tested. It therefore scored relatively low and will thus not be considered as a plausible method.

3. TA-Prioritized & TA-Hybrid (offline)

The two offline MAPD methods in [14] are **Task Assignment and Prioritized planning** and **Task Assignment and Hybrid planning** which both build further on CENTRAL [56] by improving task assignment, path planning and avoiding deadlocks. Both algorithms first perform task assignment by for each agent computing a sequence of tasks by using a special Travelling Salesman Problem (TSP) that ignores collisions and makes use of estimated travel times with the goal to minimize the makespan (according to these estimated times). After this, collision-free paths are planned (different method for both algorithms) according to the pickup and delivery points as defined in the task sequence while minimizing the makespan. To prevent deadlocks, both algorithms use the method **reserving dummy paths**, this is "*a path with minimal travel time to the parking location of an agent*" [14]. Eventually, the difference in efficiency and effectiveness will lie in the path planning method since this is the only difference between the two algorithms.

The shared **task assignment** tackles the secondary objective of the MAPD problem (after minimum makespan), namely minimizing the sum of the execution times of all task sequences. This is different from the makespan since execution times are not necessarily equal to the travel distance since agents might have to wait for the release times of tasks. The MAPD algorithms of the authors construct a directed weighted graph based on similar TSP methods as in [62, 63]. The TSP solver that is used is LKH-3 [64] which plans a good *Hamiltonian cycle* (i.e. closed loop through a graph in which each vertex is visited only once [65], see figure 5.9) and appears to plan good task sequences.

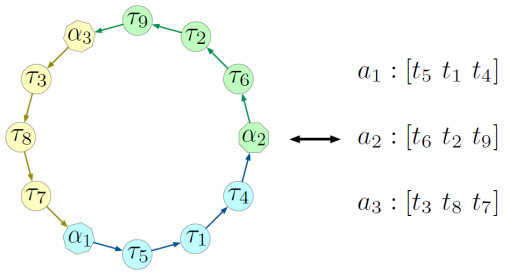


Figure 5.9: Example of a Hamiltonian cycle with three agent vertices and nine task vertices with the corresponding task sequence on the right-hand side [14].

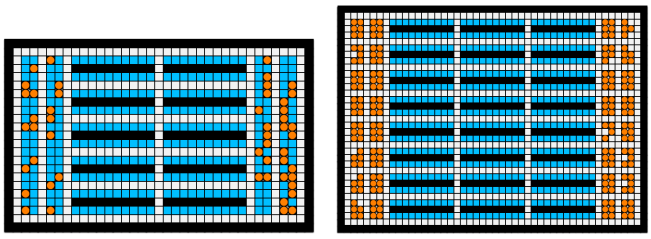
In **TA-Prioritized**, the path planning method is an improvement on the method presented in [66]. In that paper, by van den Berg and Overmars, priority is given to paths with longer expected execution times. When a path is planned for an agent (A*), the remaining agents plan their paths so that no collisions occur with the already present paths. With this approach, ultimately shorter makespans are hoped to achieve since the longest paths are not penalized extra. The difference between this method and the one in TA-Prioritized is that the method of van den Berg and Overmars already knows the next agent based on estimated travel times before the path is planned. In TA-Prioritized, the next agent is only chosen after planning the path since this might affect travel times, allowing for using actual instead of estimated travel times. A path in **TA-Prioritized** consists of multiple sub-paths in which a sub-path is the path from the current position of the agent to the goal (pickup or drop-off) location of that agent. The goal is then to minimize execution times of those sub-paths so that ultimately the makespan is minimal.

TA-Hybrid uses a similar method as TA-Prioritized but the path planning method is different: it uses a MAPF method for *new task agents* and an Anonymous MAPF (AMAPF) method for *free agents*. For the *new task agents* Improved CBS (ICBS) [67] is used to plan sub-paths from their current locations to the delivery point of the new task. A planned path remains unchanged until the agent has reached its drop-off point. For the *free agents*, sub-paths are planned from the drop-off point to the next pickup location. It uses a polynomial time min-cost max-flow algorithm, in which tasks can be swapped, at every time step where a new free agent has

joined or a free agent has left (i.e. has become a new task agent). The min-cost max-flow algorithm aims to minimize the makespan, while keeping in mind release times of tasks and estimated execution times.

Reserving dummy paths is, just like path planning, different in both methods. In TA-Prioritized, a new sub-path and corresponding dummy path are planned directly when a non-final dummy path has been planned for an agent. The new dummy path thus replaces that non-final one and does not have to avoid collisions with that one anymore. The Hybrid method stores all dummy paths since it does not directly replace the non-final dummy path that just has been planned and it thus still needs to take it into account to avoid collisions. A dummy path should contain (1) the parking spot of an agent, where the agent could stay forever, (2) no collisions with paths of other agents and (3) no parking spots of other agents. Furthermore, an agent should never move along its dummy path except when there is no other dummy path left since these are just to guarantee there is another option left for the agent to move.

The authors have implemented several **improvements to TA-Hybrid** to counteract large makespans due to possible conflicts. (1) Path planning is firstly performed without considering the dummy paths. If collisions would exist, the dummy paths that are involved are replanned. If then conflicts are still present, conflict-free plans with regard to the original dummy paths are computed. (2) If pickup points following from the min-cost max-flow computations deem infeasible, the min-cost max-flow algorithm tries to recompute pickup points for the agents. If no dummy paths can be found for those, the dummy paths are replanned while keeping the original pickup points. Finally, (3) when an agent is done performing its tasks (i.e. empty task sequence), the agent gets assigned the last task that has been assigned to an agent with the largest estimated execution time.



(a) 21×35 grid with 50 agents (b) 33×46 grid with 180 agents

Figure 5.10: A small and a large simulated warehouse environment where black cells are blocked, blue cells are pickup or delivery locations and orange cells are non-task endpoints [14].

Finally, the two developed methods have been compared to CENTRAL [56] and three other MAPD algorithms, GREEDY$_1$[1], GREEDY$_2$[2] and TA-ICBS[3], in a small and a large warehouse environment as shown in figure 5.10. In the small warehouse environment, 500 pickup and delivery tasks are generated with varying frequency (1-500 per timestep) and for different amounts of agents (10-50). In the large environment, 2000 tasks are generated that are all released at the beginning and have to be carried out by 60 to 180 robots.

The experiments showed that, regarding **task assignment**, the TA algorithms (task sequence) produced solutions with smaller makespans overall. Furthermore, the **path planning** methods of TA-Prioritized an TA-Hybrid showed to scale much better than TA-ICBS and CENTRAL. When comparing **deadlock avoidance**, between GREEDY$_1$ and GREEDY$_2$ (dummy paths vs holding task endpoints), reserving dummy paths resulted in a smaller makespan. Finally, comparing **makespans** and **runtimes**, the TA methods perform better than the other MAPD algorithms with TA-ICBS delivering the best makespans followed by TA-Hybrid. On the other hand, the GREEDY algorithms delivered better runtimes due to their faster path-planning methods.

TA-Prioritized and TA-Hybrid have shown that an improved task assignment and fast path planning, together with their deadlock avoidance method has lead to improved makespans in warehouse environments at the expense of runtimes. With respect to the CENTRAL algorithm, smaller makespan solutions are produced and better scalability is achieved in small and large simulated warehouses.

[1] Only gets assigned the next task; path planning happens in a similar way as TA-Hybrid (ICBS and min-cost max-flow); deadlock avoidance by holding task endpoints.
[2] Only gets assigned the next task; path planning happens in a similar way as TA-Hybrid (ICBS and min-cost max-flow); deadlock avoidance by reserving dummy paths.
[3] Task assignment happens in the same way as TA-Prioritized; path planning is done by ICBS and does not require deadlock avoidance.

Both algorithms deliver promising results and, as the results indicate, are among the best algorithms. There is however one big drawback which excludes both methods from being chosen: they consider offline MAPD problems. In the FLEET concept, that is to be researched, an online MAPD problem is considered. It is not known beforehand when a bag will enter the system while in TA-Hybrid and TA-Prioritized, bag arrival times are anticipated upon.

5.1.3. Trade-off

To determine which previously presented algorithm performs the best, a trade-off is performed based on the criteria drawn up in section 4.3. The trade-off has been performed disregarding the fact whether or not methods are lifelong or online, to not disregard five of the seven discussed algorithms while possibly showing favourable behaviour in certain criteria. The results of the trade-off are shown in table 5.1.

Table 5.1: Trade-off matrix MAPF with task allocation

	CBM	CBS-TA	ECBS-TA	TP	TP-SIPPwRT	TA-Prioritized	TA-Hybrid
Performance							
Scalability	2	2	3	2	1	2	2
Environmental flexibility	3	3	3	3	3	3	3
Real-world application	3	2	3	3	2	3	3
Success rate	2	3	3	3	3	3	3
Quality							
Solution quality	3	3	3	3	3	3	3
Makespan	3	2	2	1	1	2	3
Implementation							
Complexity	2	2	2	2	1	2	1
Availability	2	3	2	1	1	1	1
TOTAL	20	20	21	18	15	19	19

The results show that ECBS-TA is the best option to continue this research with, mainly due to its excellent path planning performance in terms of both runtimes, success rates and scalability. However, to be implemented successfully in FLEET, several improvements should be made based on the following remarks:

1. **MAPD:** ECBS-TA is not a lifelong version of the MAPF problem, the individual problem for a robot ends when that robot has reached its goal.
2. **Task allocation:** All tasks are available at the beginning and there is an equal amount of robots and available tasks. A task allocation mechanism, in this case the Hungarian method, should thus only be run once.
3. **Task allocation:** The Hungarian method is a centralized algorithm while the aim is to use distributed methods.
4. **Path finding:** Improvements on ECBS exist to enhance its performance even further when applied in warehouse-like environments.

In the next section, several improvements on the remarks above will be presented. It should be kept in mind that the goal will be to create a lifelong MAPF solution by combining ECBS and an optimal decentralized task assignment mechanism.

5.2. Improvements

The improvements that will be performed on ECBS-TA are now discussed. The first problem that is addressed is the task allocation in section 5.2.1. In section 5.2.2 the working principle of ECBS will be explained by presenting CBS and how ECBS compares to other improvements upon CBS. The same section also introduces three other path finding algorithms to ultimately assess the performance of ECBS and check whether it can be improved. Finally, based on previously discussed MAPD algorithms, methods to create such a lifelong version will be discussed in section 5.2.3.

5.2.1. Multi-agent task assignment

The task allocation process in ECBS-TA does not satisfy the needs of FLEET. First of all, all tasks are available at the beginning and since there is one task for each robot, the task assignment algorithm only needs to be run at the beginning of the problem. Secondly, a centralized method is used to perform the task assignment which does not satisfy the requirement for a decentralized system.

To elaborate upon the task allocation in FLEET, a description of the system is first provided based on the taxonomy of Gerkey and Matarić in [51]. They present three axes to describe multi-robot task allocation (MRTA) problems:

- **Single-task (ST) vs multi-task (MT) robots:** ST indicates that a robot is only capable of executing one task at a time. MT robots can execute multiple tasks simultaneously.
- **Single-robot (SR) vs multi-robot (MR) tasks:** SR tasks are tasks that only requires one robot while MR tasks can require more robots.
- **Instantaneous assignment (IA) vs time-extended assignment (TA):** IA means that tasks only become available when they enter the system (i.e. online) while TA indicates that it is known beforehand when tasks will enter the system (i.e. offline) and robots can anticipate upon it.

FLEET receives the taxonomy ST-SR-IA: (1) ST since each AGV can only transport one bag at a time, (2) SR because each bag can only be transported by one robot and (3) IA since it is assumed that bags only become available when they have entered the system (passengers deliver their luggage at random times).
ST-SR-IA is a form of the 'Optimal Assignment Problem' in which workers are assigned to jobs in such a way that overall performance is maximized. Furthermore, since online task assignment is considered in which tasks can be reassigned to different AGVs, if beneficial, the problem becomes an instance of the *iterated* ST-SR-IA problem [51].

According to Gerkey and Matarić, the Hungarian method and auctions are able to solve ST-SR-IA problems optimally. It should be noted that the latter has been mentioned in section 4.1.1 as market-based approach in which agents act out of self-interest. However, auctions can be used to eventually optimize a global goal since they are still programmed to deliver tasks with minimal service time [68]. Both approaches will be explained below.

1. Hungarian method

The Hungarian method works as follows [15]: Imagine you have 3 people working for you in 3 different cities and they have to travel to 3 other cities in the cheapest possible way (prices are shown in the cost matrix in figure 5.11).

	Delhi	Kerala	Mumbai
Jaipur	2500	4000	3500
Pune	4000	6000	3500
Bangalore	2000	4000	2500

Figure 5.11: 3 workers (1 in Jaipur, 1 in Pune and 1 in Bangalore) have to travel to 3 other cities (Delhi, Kerala and Mumbai). The cost matrix shows travel costs between each of the cities [15].

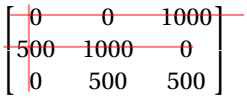
1. For each row in the cost matrix (figure 5.11), find the lowest value and subtract it from each element in the row. Note that the cost matrix is always a square matrix (nxn).

$$\begin{bmatrix} 0 & 1500 & 1000 \\ 500 & 2500 & 0 \\ 0 & 2000 & 500 \end{bmatrix}$$

2. Do the same for each column.

$$\begin{bmatrix} 0 & 0 & 1000 \\ 500 & 1000 & 0 \\ 0 & 500 & 500 \end{bmatrix}$$

3. Cover all zeros with horizontal and vertical lines. Note that a minimal amount of lines is to be used.



4. If there are at least n lines (in this case $n = 3$), optimality is possible and the algorithm is done. Else, a 5^{th} step is necessary.

$$\begin{bmatrix} 2500 & 4000 & 3500 \\ 4000 & 6000 & 3500 \\ 2000 & 4000 & 2500 \end{bmatrix}$$

5. If there are less than n lines, the smallest number from the uncovered entries is subtracted from all uncovered rows and then the same smallest number is added to the covered columns. After this, steps 3 and 4 are repeated.

The optimal solution in this case would thus be the combination Jaipur-Keraia, Pune-Mumbai and Bangalore-Delhi. According to [51], the Hungarian method has shown to be able to cope with real-time applications with runtimes of less than 1 second for 300 robots.

2. Auctions

Auction methods have proven to be flexible, scalable and operational in real-time. They can be made decentralized which allows for even better flexibility, robustness and less need for communication. On the other hand, decentralized auctions can lead to highly suboptimal solutions [9]. Furthermore, auctions (central and decentral) can be used iteratively [69, 70] in order to enhance the task allocation process.

In [71], the Improved Distributed Market-Based (IDMB) algorithm is presented. Instead of constantly iterating (i.e. repeating the auction) to improve the solution, the method lets robots swap tasks after the auction is complete. First each robot bids on a task provided by an auctioneer, the robot which is currently closest to the task wins the auction. In case a robot wins multiple auctions, it keeps the best assignment and sells the rest so that each task is assigned to a robot. When each robot has a task assigned, they communicate with each other to possibly swap tasks to minimize the total cost. The method has been tested with an amount of tasks and robots of up until 30. Results showed that the solution cost stayed within 2% of the optimal Hungarian algorithm, no runtimes have been provided however.

Another method, MURDOCH [68], is a mechanism for solving the online assignment problem (no reassigning) and the best one available according to [51]. Its working principle is to assign the task that has become available to the most fit, currently available robot using auctions. The auctioneer can monitor the progress by frequently sending renewal messages to the executor of that task and, if necessary, can reassign it.

> This research deals with the online assignment problem, therefore it is decided that an auction-based method will be used. Since each task is dealt with individually, using the Hungarian method has no benefit because the matrix it creates would only have one row with values in it (dummy rows to create a square matrix for the rest) from which the minimal value would be chosen.
> To find a global (near-)optimal solution, IDMB will be used so that robots themselves can swap tasks when it would benefit the makespan. IDMB is chosen over MURDOCH to limit communication with a centralized auctioneer and to make use of a distributed approach.

5.2.2. Multi-agent path finding

This section discusses several state-of-the-art path planning algorithms, starting with the basis of ECBS, Conflict-based search (CBS) followed by its variants, including ECBS itself. Hereafter, three other concepts to provide coordination are introduced: (1) DMAS, which uses ant behaviour as a means of coordination, (2) DiMPP, a fully distributed MAPF method, and (3) MA-RRT*, a sampling-based method that finds paths in a random manner.

1. Conflict-Based Search (CBS)

CBS is a solver for MAPF problems in which a set of agents have to find non-colliding paths to their destination. Like coupled MAPF approaches, CBS provides optimal solutions, on the other hand, the path-finding that is performed are single-agent searches, similar to decoupled approaches. The CBS algorithm exists of two levels: (1) a high level which uses a constraint tree (CT) with nodes including time and location and (2) at each of those nodes, a low-level search is performed to find new paths taking into account the constraints imposed by the high level. The main difference with A*-based searches is that CBS is exponential in the number of conflicts that the algorithm encounters during the solution process, whereas A* is exponential in the number of agents [16, 57].

The idea behind CBS is to generate constraints for conflicting paths so that by following these constraints, new generated paths do not conflict. The high level finds those conflicts and adds constraints for each of the agents. The low level updates the paths while taking into account the constraint sets of the agents so that conflicts are avoided.

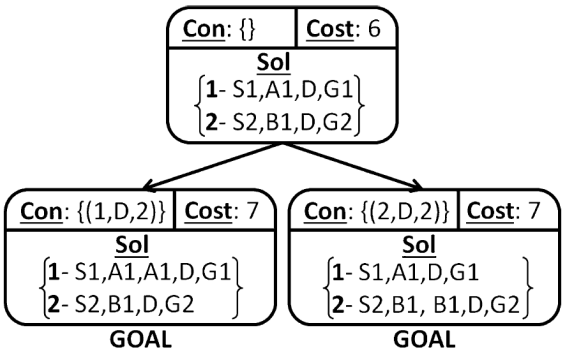


Figure 5.12: Example of a constraint tree where a conflict occurs at vertex D [16].

At the **high level**, CBS searches a **constraint tree (CT)** in which each nodes consist of (1) a set of constraints, (2) a solution (a set of paths) and (3) the total cost of that solution. Based on the set of constraints (per node), the low-level search finds a shortest path for each of the agents that is consistent with the constraints for that agent. When a consistent path is found for each of the agents, these have to be validated with respect to the other agents. If no conflicts are found, the node under consideration is declared as the goal node, if not, the node becomes a non-goal node. To resolve a conflict, two actions are possible since both agents can perform a 'maneuver'. Since optimality is required, both possibilities are considered and the node thus splits up in two child nodes, as shown in figure 5.12, which each adds a constraint for one of the involved agents (one agent per child node). The low-level searched is then only performed for the agent that is affected by the new constraint.

As mentioned before, the **low level** searches a path for an agent, according to the corresponding constraints set. It does this without taking the other agents into account (in a decoupled manner). The authors of [57] used A* but any search algorithm can be used. Ultimately, if two low-level states have the same solution cost, the one with the least conflicts is used since this results in a higher solution quality.

Figure 5.13: 'Dragon Age: Origins' maps (a) brc202d, (b) den520d and (c) ost003d [17, 18].

To assess the performance of CBS, experiments have been performed on 8x8 4-connected grid maps and on maps from the videogame 'Dragon Age: Origins' (DAO) [17]. On the grid map, CBS was compared with ICTS (+ pruning)[4] , EPEA*[5] and A* [73]. On the DAO maps, CBS was compared to EPEA* and ICTS only [16].

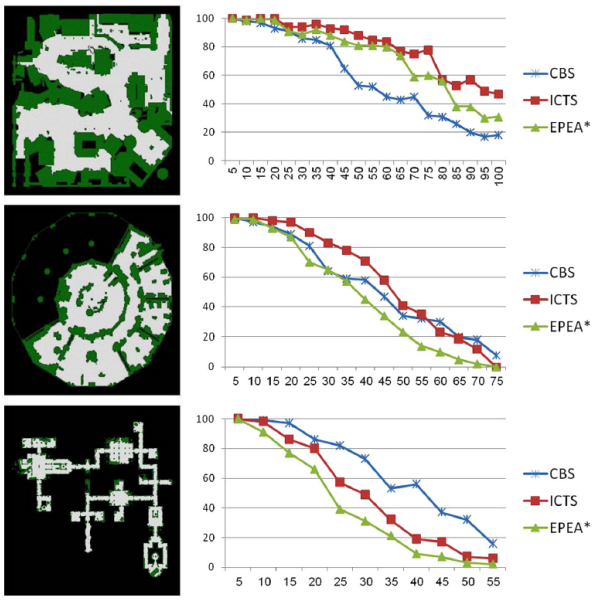


Figure 5.14: Comparison of the success rate (y-axis) between CBS, EPEA* and ICTS with varying amounts of agents (x-axis) on the different DAO maps [16].

The grid experiment compared the success rate (i.e. the amount of instances solved within a 5 minute time limit) of the four algorithms against a varying amount of agents (3 to 21). The A* algorithm clearly performed worse (not solving the problem from 7 agents on) than the other three, which at the first sight seemed to behave quite similarly with CBS having a small advantage over EPEA* and ICTS when the amount of agents increased further. This is because CBS, on the low level, needs less time then EPEA* to process a node while having expanded more nodes.

On the DAO maps, CBS was compared to ICTS and EPEA* (only the strongest algorithms were considered). The results in figure 5.14 show that there is no overall winner but each algorithm seems to behave different in different environments (open space, small corridors, obstacles). The first thing that can be concluded is that ICTS always performs better than EPEA*. In the first figure, with many open spaces, CBS delivered the worst performance. When implemented in the middle figure, with open spaces and small doorways (bottlenecks) combined, it behaved intermediate most of the time. Finally, in the third figure, that almost only included small corridors and bottlenecks, CBS outperformed the rest.

CBS is shown to be working well in environments with small corridors and bottlenecks and on grid environments. It scales well and has no need for extra infrastructure on cross points which makes it easily adaptable for different environments. Runtimes however become, from a certain amount of agents (around 10), too high to be usable in real-time. Furthermore, he solution delivered by CBS is both conflict-free and optimal.

2. CBS variants

2.1 Meta-agent CBS (MA-CBS) As concluded in the previous section, CBS is very efficient in environments with many bottlenecks, small corridors,... but inefficient for others (open spaces). In the latter, agents are strongly coupled, they experience a lot of internal conflicts. Since in that case CBS has to deal with a lot of conflicts, it behaves poorly since it has to process a lot of conflicts to obtain the optimal solution. MA-CBS aims to identify these strongly coupled agents and merge them into a meta-agent to counteract that CBS behaviour. When a meta-agent is created, the high level phase of CBS continues but the meta-agent is treated as a single agent [16, 19].

MA-CBS adds the merging action to the CBS algorithm, when a new conflict is found, the algorithm has two options: to (1) **branch** into two nodes (like the basic CBS algorithm) and to (2) **merge** the agents (M) in conflict to one meta-agent, consisting of M agents (a single agent is a meta-agent of size 1, $M = 1$). Merging works as follows. A constraint tree node N consists of k agents and the algorithm returns that agents a_1 and a_2 will be merged. These agents are merged into a meta-agent of size 2, labelled $a_{1,2}$, and the system now

[4]Two-level (high and low level) search algortithm. high-level searches the increasing cost tree (ICT) for a set of costs. The low-level searches for a path for each agent that has the same cost as in the high-level set. With pruning, small sets of agents are grouped and unsolvable combinations are identified [72].

[5]A* tends to generate nodes that are never being expanded. Therefore, Enhanced Partial Expansion A* (EPEA) only generates child nodes which have the desired solution cost [28].

consists of $k-1$ agents. Meta-agents will never be split below that node, but they might be merged with other agents or meta-agents. When a meta-agent is created, the low-level search is performed again only for that meta-agent since nothing has changed for the other agents. The meta-agent CBS algorithm has two components: the merging policy (branching or merging) and a mechanism to define the constraints on the new meta-agent.

The first one is to choose whether agents should be branched or merged, a bound parameter B is defined (notation: MA-CBS(B)). This parameter defines the amount of conflicts that can occur between agents and when that amount is exceeded, the meta-agent is created. There are two extremes in the merging policy, namely $B = \infty$ and $B = 0$. The first is equal to the basic CBS algorithm where merging never occurs and branching is performed always. In the latter, merging occurs whenever a conflict is found and is identical to ID (Independence Detection algorithm) [74]. This technique (ID) identifies groups of agents to solve them independently.

Secondly, constraints on a meta-agent originate from three groups: (1) Internal conflicts which are the conflicts between agents a_i and a_j, (2) external(i) conflicts and (3) external(j) conflicts that are conflicts with other agent a_k with respectively agents a_i and a_j. Internal conflicts are discarded since both agents will be merged and conflicts between them will thus be solved. This leaves only the external conflicts. The constraints in the new meta-agent should include conflicts that have occurred previously with other agents a_k. These constraints should not apply to the whole meta-agent but only to the agent that was originally involved in the corresponding conflicts since otherwise the optimality of the solution might be affected.

The same experiments as in [57], have been performed where MA-CBS(B) with several values of B has been compared with CBS, ICTS and EPEA* in both success rate and runtime. Figure 5.15 shows the success rate in terms of the amount of agents for each of the DAO maps.

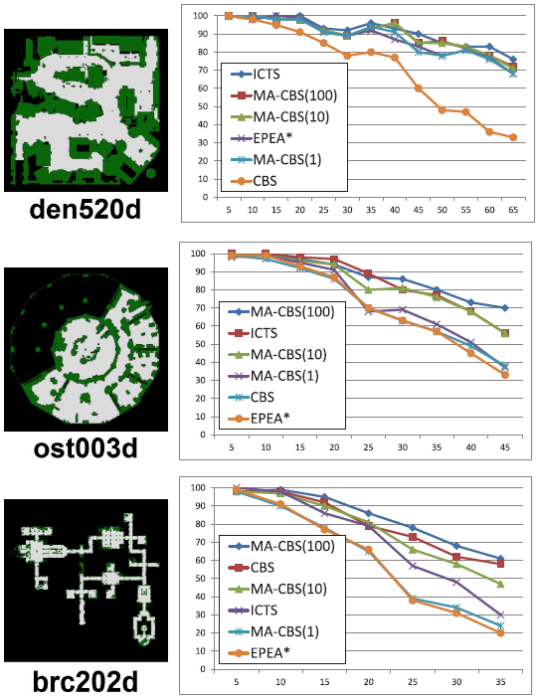


Figure 5.15: Success rate of the MA-CBS(B) algorithm on the DAO maps [19].

Comparing the success rates of MA-CBS and CBS, it can be seen that in the first map (den520d) CBS is underperforming significantly. As stated before, CBS cannot handle open spaces very well due to the amount of conflicts that arise. MA-CBS performs very well in this case since the benefit of merging is rather high because many conflicts are avoided. In the last map, the advantage of MA-CBS against CBS reduces since CBS constantly encounters new conflicts and merging agents thus only causes a small reduction of conflicts.

MA-CBS has several improvements in score compared to CBS. It has better success rates, however at the cost of complexity and availability. Interesting is that it behaves better in open spaces.

2.2 ICBS Two further enhance the working of CBS, the authors of [67] present two improvements that are implemented in an Improved optimal version of MA-CBS, ICBS and addresses a third, namely bypassing, from [75].

The first improvement is **Merge and Restart (MR)**. This method, whenever is decided to merge agents into a meta-agent, lets the search start from scratch in which the newly found meta-agent is treated as one throughout the whole process. Secondly, there is **Prioritizing Conflicts (PC)** which chooses which conflicts to split and which not according to three types of conflicts: cardinal, semi-cardinal and non-cardinal, in decreasing order of cost increase. Cardinal conflicts are thus split first since it these increase the cost the most. The third

improvement is **Bypass (BP)** which bypasses conflicts instead of splitting them by altering the path of one of the agents. BP is only accepted when it reduces the amount of conflicts in the considered node.

Experiments are carried out in the three DAO maps and show that ICBS (i.e. MA-CBS+BP+PC+MR) clearly outperforms the other variants, including ICTS and EPEA*, since it delivers the best runtimes and better or similar success rates. Comparing the CBS variants that are discussed in the paper on a 8x8 4-connected grid map, results were in favour of MA-CBS+BP+MR in success rate, runtime and amount of nodes explored. Adding PC and BP to CBS also increased the performance significantly.

> As stated above, ICBS (MA-CBS with bypassing and merge and restart) improves the success rates and runtimes of MA-CBS even further, allowing for better real-time performance.

2.3 ECBS CBS guarantees that optimal solutions will be found. However, in large problems, this leads to large runtimes. To tackle these large runtimes, algorithms have been proposed that offer minor losses in solution quality at the advantage of a significant decrease in runtimes. One of these algorithms is Enhanced CBS (ECBS), which is a bounded, suboptimal MAPF solver where the returned solution is within a defined factor from the optimal solution cost. It is shown that ECBS almost always outperforms other suboptimal MAPF algorithms [76].

ECBS(w_1) is a w_1 suboptimal variant of CBS. However, instead of best-first searches, the high- and low-level searches are focal searches. In a focal search, nodes are sorted in increasing order of solution cost in an OPEN list. An example of a focal search is A* but ECBS has an additional FOCAL list which includes all nodes in OPEN whose solution costs are not higher than factor w_1 times the current lowest value in the OPEN list. The nodes in the FOCAL list are then sorted in increasing order of secondary heuristic value, related to the amount of collisions. A focal search thus expands nodes in the FOCAL list which are close to a goal node, speeding up the search process [21].

ECBS is compared to other suboptimal CBS variants (GCBS, BCBS [76]) and to other MAPF solvers. From the first experiment followed that ECBS is a more appropriate solver than the other two regarding stability, reliability and guaranteeing the provided bound w_1. When compared to other MAPF solvers ECBS was overall the best performer .

> ECBS seems a bit less complex than MA-CBS and is available on GitHub, however literature does not provide pseudo-codes. Another very important difference is that ECBS does not provide optimal results but in return provides solutions faster, improving its real-time applicability.

2.4 (E)CBS+HWY In [20], the authors propose an extension of CBS and ECBS to be able to capture the MAPF problem in warehouse-like environments. In order to further reduce conflicts, to enhance the performance of (E)CBS, highways are added to the environment to provide guidance for the agents and avoid collisions with agents moving in the opposite direction, these are visualized in figure 5.16. An important note is that both algorithms, CBS+HWY and ECBS+HWY are bounded-suboptimal algorithms.

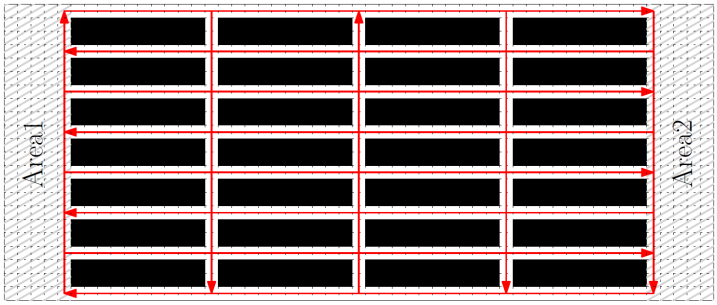


Figure 5.16: Warehouse-like environment with arrows representing highways [20].

The behaviour of the algorithms are influenced by a user-provided parameter w that "determines the level of encouragement for path finding to return paths that include edges of the highway", HWY(w) [20]. CBS+HWY(w) and ECBS(w_1)+HWY(w_2) are thus versions of CBS and ECBS in which the low level searches use the highway heuristic values (w and w_2 respectively) instead of the shortest path heuristic values.

ECBS(w) and ECBS(w_1)+HWY(w_2) are compared in the warehouse environment of figure 5.16 for which 10 random instances have been generated with 150 agents. The obtained results indicate that the HWY version results in lower runtimes, solution costs (better optimality) and that it has a higher success rate.

> Adding highways to ECBS even further enhances its performance by both decreasing runtimes and solution costs while achieving higher success rates.

2.5 iECBS As a reaction on ECBS+HWY which due to two suboptimality factors w_1 and w_2, caused by the focal search and the highway respectively, leads to having a suboptimality factor of $w_1 w_2$. The authors of [21] present a feasibility study in which a w_1 suboptimal variant of ECBS(w_1)+HWY(w_2) is developed that has suboptimality w_1 instead of $w_1 w_2$ while using both focal searches and highways: iECBS(w_1), Improved ECBS. Furthermore, the paper presents a way to automatically generate highways to reduce human interaction.

A performance comparison between iECBS(w_1) and ECBS(w_1)+HWY(w_2) is not performed since it is unclear how to compare both fairly due to the large amount of possible parameter combinations. Comparing ECBS(w_1) and iECBS(w_1) in the environment of figure 5.16, results show that iECBS runs faster and has better solution costs.

		1	2	3	4	5	6	7	8	9	10
ECBS(1.5)	Time	287	300	92	300	300	300	**71**	33	58	54
	Cost	9357	n/a	9234	n/a	n/a	n/a	9267	9349	9179	9676
iECBS(1.5)	Time	**28**	**15**	**7**	**7**	**41**	**8**	77	**8**	**7**	**17**
	Cost	**9141**	**9002**	**8930**	**8925**	**9143**	**8967**	**9070**	**9138**	**8785**	**9181**

Figure 5.17: Comparison of the runtimes and solution costs of ECBS and iECBS with $w = 1.5$ in the Kiva environment of figure 5.16 and a runtime limit of 5 minutes [21].

> As stated above, iECBS improves on ECBS on both runtime and solution costs in Kiva-like environments. Furthermore, comparing iECBS to ECBS+HWY, it is not guaranteed that iECBS will provide lower runtimes. Together with pseudocodes or implementations being available, iECBS is not considered to be a better choice than ECBS+HWY.

3. DMAS

Delegate Multi-Agent Systems (DMAS) have their foundation in the behaviour of ant colonies during food foraging. Ants, when looking for food, walk around randomly in the environment when there are no signs of food. When one of the ants would then 'by accident' discover a food source and walk back to the nest, it leaves a trail of a smelling substance, a pheromone. Other ants will pick up this trail and as such find the food source. When the food source has been gathered, ants will stop moving towards the source, stop dropping the pheromones and the trail eventually evaporates. Pheromones are thus meant to show possible routes to a food source which ants will instinctively trail [77]. The approach in [77] makes use of this property in a vehicle routing problem. By dropping information of its plan in the environment, an agent can provide other agents with information on its intentions. Other agents can then use this information to create their own plans.

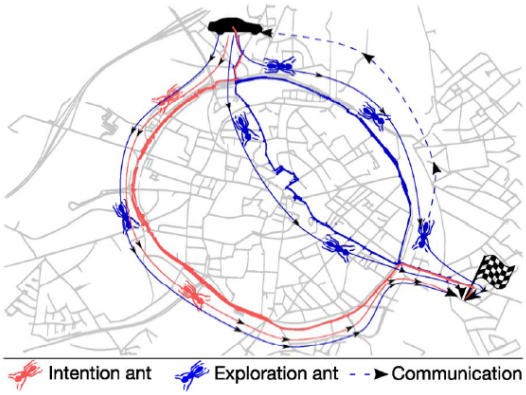


Figure 5.18: Representation of Leuven, Belgium, in which exploration ants are send out to explore feasible routes followed by the intention ant that follows the chosen route and notifies infrastructure agents [22].

Coordination is provided by two types of agents that share the environment: **vehicle agents and infrastructure agents**. **Vehicle agents** have two responsibilities which they perform by frequently sending out two types

of 'light-weight agents' as shown in figure 5.18 [77]. First they send out *exploration agents* that look for possible routes to its destination and return the quality of the route (i.e. the time it would take to follow the route). From these explored routes, the vehicle selects a route based on that quality. Secondly, an *intention agent* is send out to ultimately let the other agents know about the intentions of the vehicle under consideration. The *intention agents* do this by letting the infrastructure agents it passes know that the vehicle intends to pass by. Finally, the **infrastructure agents** can use the information (intentions) from the vehicle agents to determine future traffic loads and is aware of (unexpected) obstacles [22].

In [23], DMAS is used as a solution for the MAPF problem. Figure 5.19 shows a shared environment in which vehicles can move around. In this infrastructure, the vehicle agents are represented by the cars and the infrastructure agents (in this paper denoted as *resource agents*) are the dark yellow squares. The goal of the resource agents is not only to determine future traffic loads but also to manage a schedule off when an agent crosses the intersection.

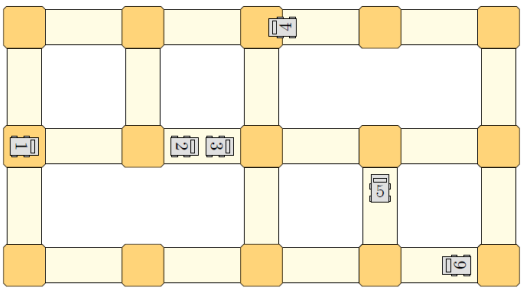


Figure 5.19: Grid-like environment where the crossings (dark yellow) are equipped with infrastructure agents [23].

The vehicle agents need to reserve free time slots that are provided by the resource agent to cross the corresponding intersection. The vehicles need to frequently repeat their reservation since, just like the pheromones, it otherwise 'dissolves' in time. Reservations happen based on the quality of routes (*intention agents*) that are being explored (*exploration agents*) repeatedly. If two vehicles would reserve an intersection simultaneously, the *resource agent* randomly gives priority [23].

The method that is presented in [23] is compared to context-aware routing (CA) [78] in which a set of agents have to plan conflict-free routes between two points without invalidating the plans of other agents. It is shown that due to its decentralized approach, DMAS offers better scalability in dynamic environments while providing a comparable solution quality. Furthermore, when vehicles are required to continuously calculate new routes, higher throughput can be achieved.

DMAS seems a good way to spread the traffic of AGVs over the network since dense areas are being avoided. It copes well with dynamic environments and provides good scalability with a decent solution quality, however due to constant rerouting of the AGVs the solution provided by DMAS will not be optimal. Considering the performance, since it requires an agent at each crossing, FLEET cannot be placed in a random hall and be expected to work. The success rate of DMAS on the other hand scores well since vehicles are constantly looking for other routes, being beneficial for the capacity of the system. Implementations of DMAS seem straightforward and are available both on GitHub and in pseudocodes in literature.

4. Distributed (improved) Multi-Agent Path Planning

Multi-agent path planning (MAPP) methods can be divided in two classes: coupled (centralized) and decoupled (decentralized or distributed) approaches. Coupled approaches have been shown to find optimal solutions for MAPF/MAPP problems (MAPP, CBS, OD) [57, 67, 79, 80], however these can become, when the problem increases, computationally very intensive. To tackle this problem, decoupled approaches have been introduced with the aim to increase efficiency but, currently, at the expense of optimality leading to suboptimal solutions (ECBS) [20, 21].

Decoupled algorithms consist of three phases: (1) First the paths of individual agents are planned. (2) Secondly, to provide coordination, the order in which plans will be replanned is defined by giving a certain priority to each of the agents. (3) Finally, considering the obtained prioritization, the plans of the individual agents are computed in decreasing priority to obtain conflict-free paths [79].

This section discusses two decoupled algorithms, DMAPP and DiMPP, which are fully distributed (i.e. all three phases are distributed) MAPP algorithms. First of all DMAPP will be explained after which its improved and complete version (DiMPP) is presented.

4.1 DMAPP Chouhan and Niyogi [25] present a fully distributed multi-agent path planning algorithm, DMAPP, that aims to tackle the need for a centralized controller. DMAPP consists of three phases: (1) individual path planning, (2) decision-making and (3) plan restructuring.

Phase 1, path planning, makes use of the FF planning system (Fast plan generation through heuristic search) [81]. In the latter article, a breadth first search is implemented that removes unnecessary successors of a search state to speed up the algorithm. It aims to find the shortest path (Euclidean distance, i.e. straight line between two consecutive points) by exploring a search tree, in which paths are excluded layer per layer (figure 5.20). At the end of the iteration, an individual path is found without considering the other agents.

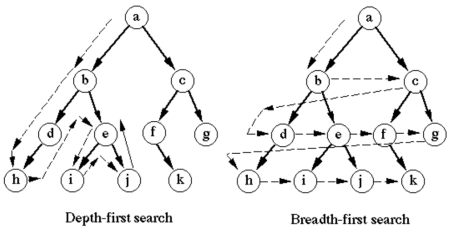


Figure 5.20: Depth-first search on the left and a breadth-first search on the right [24].

Since the individual plans from phase 1 might lead to conflicts between agents, agents will have to restructure their plans to assure collision-free paths. **Phase 2** will **decide the order** in which restructuring happens. The method used in DMAPP is based on the individual path lengths from phase 1 with the agent that has the longest path length to receive the highest priority. This method would allow for (1) a reduced restructuring time and (2) an increased number of solvable problem instances when compared to random prioritization.

The working principle of phase 2 is based on the LCR (Le Lann, Chang and Roberts) algorithm to solve the 'Leader Election in a Synchronous Ring' problem [82]. In this algorithm, each agent sends its identifier and path length to the next agent clockwise so that ultimately, when the algorithm finishes, each agent knows the path lengths of all other agents. By comparing their path lengths, each agent knows its priority. Figure 5.21 shows how it works. The network consists of 4 agents with each its identifier and path length. Initially (a) the set of each agents contains the agent itself. After each round (b)-(d), each agent receives a new tuple (<*identifier, path length*>) so that finally, after the third round (d), all agents have the same set. Each agent (1) knows it relative priority with respect to the other agents, (2) can thus define (in a distributed way) the order of restructuring and (3) know with which agents it should communicate. In figure 5.21 the resulting order is, from high to low priority, 4-1-3-2.

Finally in **phase 3**, the **plans are restructured** according to the priorities obtained in phase 2. The agent with the highest priority will not replan its path, the second agent will replan its path considering the path of the first agent, the third agent replans its path taking into account the previous two agents and so on till the agent with the lowest priority has restructured its path according to all other agents. Note that the same path planning method as in phase 1 is used.

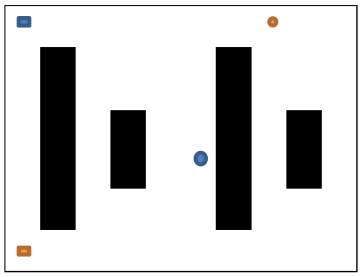


Figure 5.22: Simulated environment with blocked cells (black), two agents (rectangles) and their goals (circles)[25].

To evaluate DMAPP, the authors created a warehouse domain (figure 5.22) with the objective to find coordinated paths for all agents. The authors have used the randomized prioritized planning approach from [83], where priorities are already decided upon, and the priority scheme from [84], where priorities are also based on path length, to compare the performance of the priority decision making of DMAPP to. The results show that DMAPP behaves particularly well with respect to the other two methods since it solves large problem instances that for both the randomized and pre-decided methods deemed unsolvable. A drawback of DMAPP, however, is that it is incomplete [25]. It does not reconsider the priority order, so if it terminates and no solution is found, it terminates and reports that it could not find a solution. Even if it exists, DMAPP might thus not find the solution due to the restructuring phase.

No comparison with other MAPF methods is performed since in [25] this was still a future goal. In the improved DMAPP method, DiMPP, which will be discussed in the next section, these comparisons are indeed present.

4.2 DiMPP DiMPP [18] is an improvement on DMAPP which tackles the incompleteness of the latter. Since DMAPP only considers one priority order in phase 2, plans are restructured only according to this plan. If

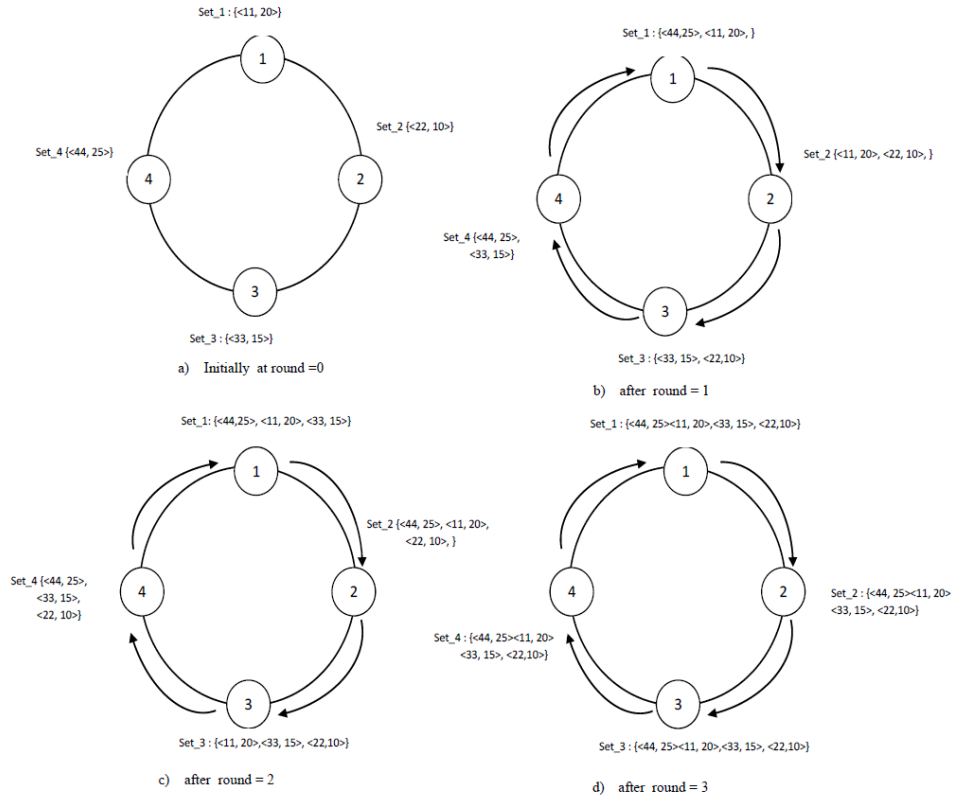


Figure 5.21: Representation of the distributed priority decision making process with four agents [25].

a priority order does not allow for a feasible solution, the algorithm terminates. However, it might be that a slight change in the priority order does provide a solution but this is not covered by DMAPP.

The **first two phases**, (phase 1) **individual path calculation** and (phase 2) **prioritization**, are the same as in DMAPP. Phase 3 on the other hand has changed, restructuring can start with any agent but a priority order is still defined by passing around the synchronous ring (figure 5.21).

Restructuring, phase 3, works as follows: the initiator (initially with highest priority) agent updates its plan and sends it to the next agent. If an agent receives the plan, the plan can be *empty, not empty* or can *contain the plan* of that agent already. In the first case, the agent makes itself the initiator, updates the plan and sends it along. In the latter case, the circle is round and a solution has been found. When a plan is not empty, the following cases can occur:

- *Agent's plan is not conflicting:* The agent updates the plan with its own path and sends it along.
- *Agent's plan is conflicting and can restructure:* The agent updates the plan with its restructured path and sends it along.
- *Agent's plan is conflicting but fails restructuring and has not been the initiator before:* The agent resets the plan (i.e. makes it empty), makes itself initiator, adds it path to the plan and sends it along.
- *Agent's plan is conflicting but fails restructuring and has been the initiator before:* The agent looks for an agent that has not been initiator before, resets the plan and sends it to that other agent which then receives an empty set and becomes initiator.
- *Agent's plan is conflicting but fails restructuring and all agents have been initiator already:* No solution exists.

The authors have provided proof that the plan restructuring always terminates by finding that a solution exists, or no solution exists. This solution, if it exists, is guaranteed to be found by the restructuring algorithm making DiMPP complete.

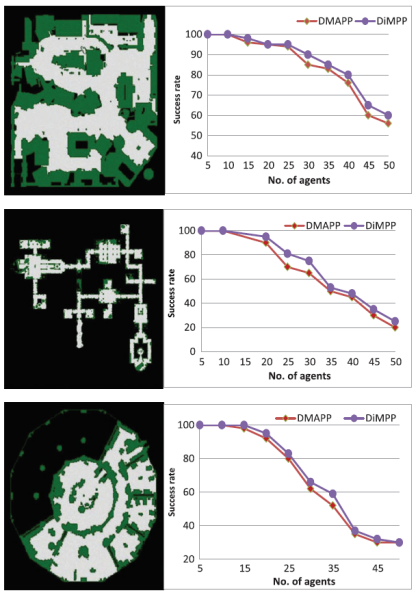


Figure 5.23: Success rate (y-axis) of DiMPP and DMAPP with varying amounts of agents (x-axis) on the different DAO maps [18].

The performance of DiMPP is evaluated in three maps of video game Dragon Age: Origins (DAO), since these are commonly used in MAPF evaluations consisting of small corridors and/or large open spaces as can be seen in figure 5.13, and on a 8x8 grid domain. Its performance is compared to centralized algorithms: to MA-A*, EPEA* and CBS on the 8x8 grid domain and to EPEA* and CBS (and DMAPP) on the DAO maps.

In the **small grid domain**, experiments were run with 3 to 15 agents with a time limit of 5 minutes. It showed that up to 6 agents, EPEA* outperforms DiMPP, MA-A* and CBS since it expands less nodes. However, since the amount of nodes in EPEA* increase exponentially, just like those MA-A* and CBS, their runtimes increased drastically from 7 agents on. DiMPP on the other hand, which increases polynomially, showed to scale really well and solve all of the instances.

The experiments on the **DAO maps** behave in the same way: At lower amounts of agents, DiMPP performs worse than EPEA* and CBS since it has to deal with the priority decision-making process which brings a standard overhead time with it. But again, when the problem grows larger and more agents are involved, DiMPP outperforms the other algorithms. Furthermore, in figure 5.23 the comparison between DiMPP and DMAPP in the DAO maps is shown. The graphs show that in each map the success rate of DiMPP slightly outperforms DMAPP.

Regarding its results, DiMPP seems very promising since it outperforms CBS in scalability, runtimes and solution cost. This leads to DiMPP behaving similarly as ECBS+HWY considering both performance and quality. However, when looking at the success rates in the middle figure of figure 5.23, it can be seen that from 20 agents on, the success rate quickly deteriorates. Comparing this to ECBS in section 5.2.2, the success rate of ECBS only starts worsening significantly from 130 agents on. This is important since in the literature review of Olijslager [4], which was in collaboration with Vanderlande, the aim was to create a system that can handle up to 96 AGV's.

5. MA-RRT*

As an alternative for forward search algorithms to solve cooperative pathfinding problems, multi-agent RRT* (rapidly exploring random trees) is proposed in [27]. MA-RRT* is a sampling-based algorithm which joins two approaches: planning of the actions of the agents still happens in their joint-state space but instead of using A*, the fast sampling-based algorithm RRT* [85] is used. Instead of searching for an optimal solution (minimal cost), RRT* converges to a continuously optimal solution.

The algorithm (RRT*), creates a tree that is rooted at the start state and randomly selects new follow-up states. If a goal region (the goal is not considered as being a single point) is reached, the edges are followed back to the root to find a feasible path from start point to goal region. If the goal region is found, the algorithm does not stop but instead continues generating random samples to discover new paths with lower costs. Figure 5.24 shows the working principle of RRT. The difference with RRT* is that RRT connects the random sample to the nearest state that already is present in the tree while RRT* starts with considering all states (in the tree) that lie within a certain radius and from these states connects to the random sample that, when following the path back to the initial state, leads to the path with the lowest cost.

To evaluate the performance of MA-RRT*, the authors compared their method with two A* search methods, the joint-state space (JA) and optimal anytime (OA) algorithms. Furthermore, one additional MA-RRT* improvement was added to the comparison, informed-sampling MA-RRT* (isMA-RRT*), which adapts the sampling method to favour promising regions of the state space and samples are thus more frequently generated in these regions. The four algorithms are tested in a grid environment shown in figure 5.25 with varying grid sizes (10x10 up to 90x90) and varying amounts of agents (up to 10). Figure 5.26 shows the performance curves

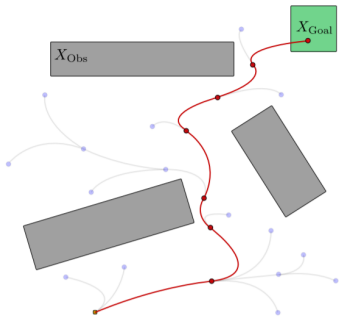


Figure 5.24: Example of the working principle of RRT where paths are randomly created until the goal region has been found [26].

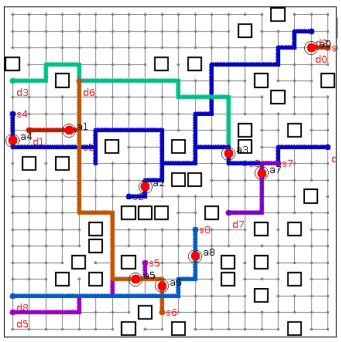


Figure 5.25: Example of problem instance with a 20x20 grid and 9 agents [26].

which shows that MA-RRT* and isMA-RRT* are more capable of finding solutions (56% and 77% respectively) than JA and OA within 5 seconds. However, better quality solutions are found by JA (always returns optimal solutions) and OA. In figure 5.27 the scalability of the algorithms is shown for varying grid sizes with 10 agents and for a large grid with varying amounts of agents. Overall, both MA-RRT* versions deliver the best results. Unlike the other two algorithms, the MA-RRT* versions do not show a decline in performance when the size of the environment grows. On the other hand, when the amount of agents increases, their performance declines. Thus, MA-RRT* can best be used in environments where agents have sufficient space to move.

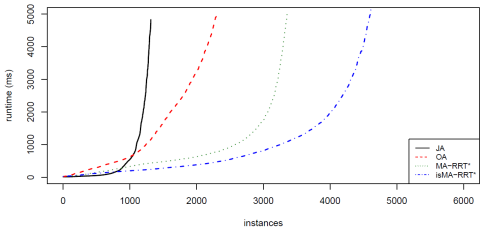


Figure 5.26: Performance curves to find the first valid solution [27].

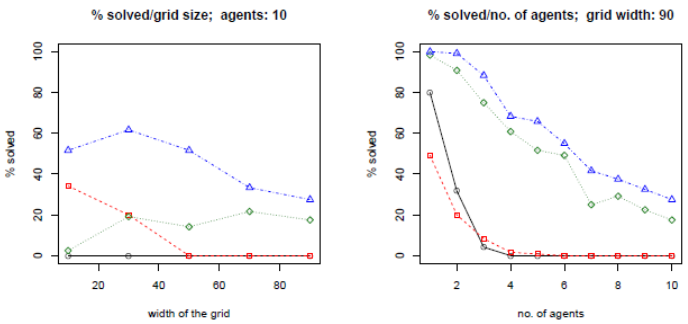


Figure 5.27: Results of the scalability in grid size (left) and number of agents (right). MA-RRT* is represented by the green line (diamonds), isMA-RRT* by the blue line (triangles), JA by the black line (circles) and OA by the red line (squares) [27].

This last MAPF solution is a bit of an outsider due to its randomized path finding mechanism. Regarding performance, MA-RRT* is shown to scale well but it is unclear how it behaves in comparison with for example CBS and its success rates are intermediate (between 56% and 77%).

6. Trade-off MAPF selection

To be sure that ECBS is the performing MAPF algorithm, a trade-off is again performed in table 5.2. The same criteria as in section 5.1.3 apply except for 'makespan' which is replaced with 'optimality'.

Results show that the improvement on ECBS, in which highways are added to the environment, is the best performing algorithm. This means that nothing changes in the algorithm itself, only the environment model needs to be modified.

A final small improvement that could be made on ECBS is on its lower level search, namely on the path finding mechanism A*. Since any path finding algorithm can be used and improvements on A* exist, Enhanced Partial Expansion A* (EPEA*) is shortly discussed below.

EPEA* The authors of [86] first improved upon the A* algorithm by adding Partial Expansion (PEA*). This lets the algorithm only store nodes that are necessary to find an optimal solution since A* unnecessarily stores

Table 5.2: Trade-off matrix MAPF selection.

	DMAS	CBS	MA-CBS	ICBS	ECBS	ECBS+HWY	iECBS	DiMPP	MA-RRT*
Performance									
Scalability	2	2	2	2	2	2	2	1	2
Environmental flexibility	1	3	3	3	3	3	3	3	3
Real-world application	1	1	1	1	2	3	3	3	3
Success rate	3	1	2	2	2	3	2	3	2
Quality									
Solution quality	2	3	3	3	3	3	3	3	2
Optimality	1	3	3	3	2	2	2	2	2
Implementation									
Complexity	3	2	1	1	2	2	1	2	1
Availability	3	3	1	1	2	2	1	1	3
TOTAL	16	18	16	16	18	20	17	18	18

all child nodes. PEA* is shown to be more memory efficient than A* and to be applicable to real-world applications. In [28], PEA* is even further enhanced (Enhanced PEA*, EPEA*) by only storing child nodes which have the same solution cost as the expanded node.

However, results show that EPEA* does not perform better than A* in single-agent path finding problems, it expands more nodes than A* leading to a larger runtime as can be seen in figure 5.28. This is due to EPEA* not being able to perform duplicate detection (i.e. already explored nodes are re-inserted in the system). Therefore, since EPEA* does not perform well on 4-connected grid maps, A* will be kept as single-agent path finding algorithm.

	Logarithmic		Constant	
Algorithm	Time (ms)	OPEN	Time (ms)	OPEN
A*	29.1	648	24.7	645
EPEA*	44.2	2371	31.5	2115
BPEA*	44.4	2349	34.0	2117
BPEA*$_{dd}$	47.7	1908	36.2	1724
BPEA*$_{dd-o}$	53.0	741	-	-

Figure 5.28: Comparison of PEA* (referred to as BPEA*, Basic PEA*) variants on a 4-connected grid map with obstacles [28].

5.2.3. MAPD

The MAPD algorithms discussed in section 4.2.3 all make use of planning paths one after another, except for TA-Hybrid. Since ECBS had shown to outperform other MAPF methods in which paths were planned according to priority, the MAPD methods quickly showed lack in path finding mechanism. However, TA-Hybrid became interesting when it seemed that they implemented another CBS variant, ICBS, in their path planning mechanism.

The approach divides the robots which require paths in two groups, robots that just picked up their task and robots that are free. Robots that just acquired their task are planned with ICBS whereas the free ones are planned using a min-cost max-flow algorithm. To not always have to run ICBS, paths between pickup and delivery points do not change. The free agents are planned each time a an agent has become free or an agent has become occupied (not free).

> The upcoming research will look for ways to create a lifelong MAPF version using ECBS+HWY. One of the research questions in section 6.1.2 will thus address how ECBS+HWY can be used in an MAPD problem.

5.3. Considerations

This last section presents five things that are not directly covered by the chosen MAPD method but that have to be considered during the actual project. First of all the environment in which the robots will be operating will be defined in section 5.3.1. Secondly, in section 5.3.2 the implementation of buffer zones will be introduced. Furthermore, priority will be addressed in section 5.3.3 followed by the battery life that has to be considered in section 5.3.4. Finally, since safety has to be guaranteed when plans are executed, section 5.3.5 presents three safety considerations.

5.3.1. Environmental layout

Looking at the infrastructure presented by Vanderlande in figure 5.29[6] and comparing this to the warehouse-like environment in figure 2.3, it is decided to disregard the latter as environment for future research because of the complexity of the required structural infrastructure. Like the master's thesis of Olijslager [30], the MAPD problem will be created in a similar environment as shown in figure 5.30. Note, however, that the research will start from scratch and thus the infrastructure will not be taken over entirely. New AGV streams (indicated by the blue arrows and white boxes) might be defined if beneficial.

Figure 5.29: The FLEET experience in an airport [29].

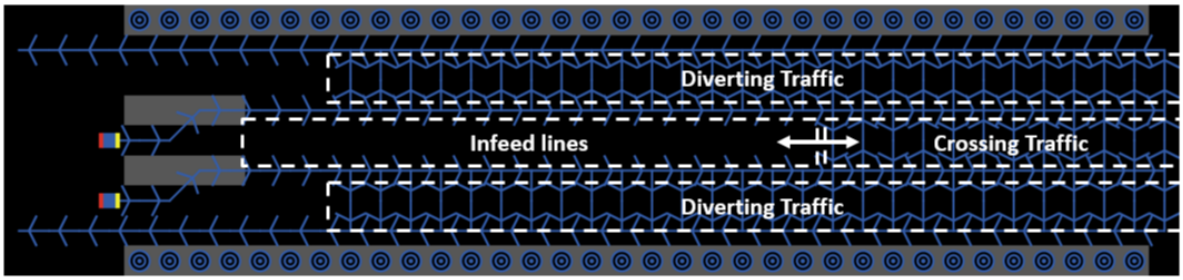


Figure 5.30: The layout used in the master's thesis of Olijslager [30].

5.3.2. Buffer zones

To minimize waiting times for bags, it is better
to already have an AGV waiting for one. Therefore in a previous assignment on this research [87], waiting lines have been used to have sort of a buffer zone. This buffer zone can also be found in [4] and has an additional advantage that by queueing up before pickup, not only bag waiting times are reduced but also charging periods for AGVs can be extended.

5.3.3. Priority

To not forget one of the main goals of FLEET, it should be made sure that the created MAPD algorithm will be able to capture the priority of bags. At this point it is not determined how it will be implemented in ECBS+HWY and the task allocation but there are ways to work around it. One way could be to whenever a bag with priority arrives, letting it plan its shortest path and re-evaluate all other paths, think about the working principle of TA-Hybrid.

5.3.4. Battery life

AGVs do not possess an unlimited amount of energy, their energy level will reduce when time passes. Because of this, they will have to charge in between executing tasks. Letting AGVs charge while carrying a piece of luggage will only increase the service time and should thus be avoided at all costs.

5.3.5. Plan execution (short-term planning)

During the execution of the BHS, ECBS will frequently be replanning paths to cope with the online task allocation problem. Therefore, a short-term planning mechanism will not be implemented. However, conflicting agents still might have to be dealt with when the plans are actually executed. Three items which the system should take into account are listed below.

[6] https://www.youtube.com/watch?v=CXJLpqscNYg

1. **Separation:**

   To guarantee safety at all times throughout the plan execution of the AGVs, a minimum separation between robots will be defined. When two robots violate the minimum separation, the system is informed about this conflict.

2. **Priority in close conflicts:**

   Priority in close conflicts refers to the moment when two AGVs are at a crossing point. In this case, one of the AGVs will have priority over the other one and the situation will be resolved by giving way to the agent with the highest priority. The order of priority can for example be defined by the total distance an agent has to travel or by the bag they are carrying.

3. **Dead- and livelocks:**

   During the plan execution, dead- and/or livelocks can emerge from interactions between the AGVs. At this moment it is still unclear whether ECBS+HWY will be able to solve these by replanning AGV paths. Figure 5.31 shows different conflicts that can emerge from AGV routing. The highways in ECBS will however provide some coordination already that reduces these conflicts.

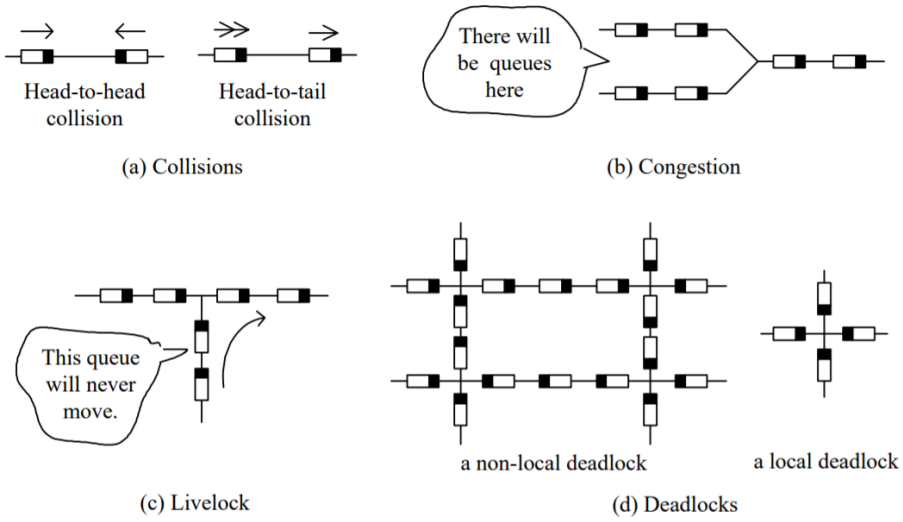


Figure 5.31: Conflicts during routing of AGVs [31].

6

Research plan

Considering the literature review and the method that has been chosen, this chapter addresses the goal of the MSc project. The research proposal is presented in section 6.1 and elaborates on both the research objective and its corresponding research questions. Section 6.2 then explains how the research questions will be dealt with in the following seven months, it subdivides the creation of the agent-based model in five work packages. Finally, an initial plan is made to estimate the working time on the first packages in section 6.3.

6.1. Research proposal

Considering the literature that has been reviewed in this literature study, this section addresses the project goal and the methods to achieve this in section 6.1.1. After this, research questions that will have to be answered are drawn up in section 6.1.2.

6.1.1. Research objective

The literature review has presented several methods to use in a distributed baggage handling system. Since task allocation and path planning often has been used separately, the aim was to combine these two in order to improve the efficiency of the BHS. Additionally to this combination, the aim is to create a lifelong MAPF version. This means that instead of all tasks being available directly, like the TAPF solutions presented in section 4.2.2, tasks would appear at different times so that the problem eventually became a Multi-Agent Pickup-and-Delivery problem as elaborated upon in section 4.2.3.

However, all state-of-the-art MAPD solvers use a combined task allocation and path planning method which plans paths for AGVs one after the other (except for TA-Hybrid, section 5.1.2, where some robots are coordinated with ICBS). These paths always included a task which had been chosen by the AGV so that the task allocation was fully incorporated in the path planning. But, since ECBS+HWY had proven its planning capabilities, it was chosen to go into another direction and use this algorithm in an iterative way (similar to TA-Hybrid) to provide coordination for all AGVs in all movement phases (picking up, dropping off, going to charge). Furthermore, it was decided to use an auction-based method, IDMB, for task allocation which allows for swapping tasks in a distributed manner. Finally, the new MAPD solution will have to take into account bag priorities and the battery life of AGVs.

The main research question is thus defined as follows:

> *"How can a flexible and scalable distributed baggage handling system be controlled by means of a multi-agent pickup-and-delivery algorithm which combines prioritized task assignment and lifelong path finding while considering AGVs' battery life?"*

6.1.2. Research questions

Several research questions have been developed to eventually come up with a solution for the research objective. The research objective has been split up in three sub-questions that address the path planning method,

task allocation method and charging of the AGVs.

The first question addresses the behaviour of ECBS in a lifelong MAPF instance. Since the discussed ECBS-TA method did not consider an online task assignment, all tasks were available at the start and so all plans were planned directly. However, in TA-Hybrid (section 5.1.2) ICBS was used to schedule the paths of agents that just received a new task. It should thus be possible to successfully implement a lifelong version of ECBS.

1. How can ECBS+HWY efficiently handle a lifelong version of the MAPF problem?
    (a) How can ECBS+HWY be designed to optimize with regard to makespan instead of solution cost?
        i. Can a path cost be defined such that it represents the travel time?
    (b) How will ECBS+HWY guarantee a defined separation between AGVs?
    (c) How will priority be handled by the system?
        i. How can priority be integrated in ECBS?
        ii. How will agents handle priority during conflict avoidance, i.e. short term?
        iii. Can prioritizing AGVs positively impact the performance of the system?
    (d) How will obstructions (e.g. failing AGVs) be tackled by ECBS+HWY?
        i. How will these obstructions be noticed by the system?
        ii. Will replanning suffice for avoiding the failed AGV?

Secondly, it should be considered how auctions will be combined with ECBS to create a MAPD problem. The MAPD solutions that have been discussed use a block of shared memory (token) from which they choose a task, plan their paths taking into account previously planned paths and pass the token to the next agent. Since ECBS does not plan paths one after the other, the auction should thus somehow consider paths that are provided by ECBS.

2. How can auctions be used in a lifelong MAPD problem?
    (a) How will the auction mechanism be connected to ECBS+HWY to act like a MAPD algorithm?
        i. Can the path lengts generated by ECBS be used as bids in the auction?
    (b) How will auctions handle the iterative process of reassigning tasks?
        i. Will IDMB positively influence the task assignment?
        ii. What will be the difference between iterating the auction process and the IDMB method?
        iii. How will auctions consider priority during task allocation?

Finally, charging of the AGVs should be addressed.

3. How will an AGV decide to go charging?
    (a) At what point should an AGV decide to go charge instead of executing a task?
    (b) How does the AGV know at which charging station to go charge?
        i. Can the task allocation method (IDMB) be used to decide upon the charging station?
        ii. Should an AGV always be charged fully?
        iii. Will an AGV that wants to charge be able to take the position of a charging one when the latter is still charging?

6.2. Methodology

Following the research objective, the goal is to implement ECBS+HWY combined with IDMB in a lifelong MAPD problem in a similar environment like figure 5.29. This section therefore elaborates on the way towards the final model. The objective is summarized in section 6.2.1 after which the construction of the model is divided in five work packages in section 6.2.2.

6.2.1. Objective recap

The literature review addressed a lifelong multi-agent pickup-and-delivery problem in a distributed baggage handling system. In a lifelong MAPD, agents have to pickup and deliver tasks that appear in an online or offline manner. Here, the online version is considered. This means that it is not known beforehand when tasks will appear, unlike the offline version. In this case, luggage will appear randomly in the system and only become available (and known) for the AGVs when they are ready for pickup.

The agent-based model that is to be made will have to take into account three major issues: (1) which task will be picked up by which agent, (2) planning collision-free paths for all AGVs and (3) the battery life of the AGVs. These will thus be the three main work packages. It should be noted however that these will be iterated upon. For example defining how to handle the battery life of AGVs and how task allocation will be incorporated in path planning. Furthermore, the research will take into account bag priority and buffer zones [4] will be implemented.

6.2.2. Implementation phases

The assignment of the agent-based modelling course in 2018 [3], in collaboration with Vanderlande, also covered developing a basic model for FLEET. With a given Kiva-like environment (figure 2.3), the following workflow was imposed:

1. Make the AGVs go to charging stations
2. Create individual plans for the AGVs to pick up and deliver pieces of luggage
3. Ensure separation between AGVs
4. Coordination

The software that was used in the mentioned assignment was NetLogo[1]. However, in the upcoming model, Python 3.7[2] will be used because of its flexibility and speed.

WP1 (4 weeks): Familiarize with ABM in Python and make the AGV go charge

The first work package will take approximately 4 weeks to complete. The main goal of this package is to become familiar with agent-based modelling in Python. This will be done by creating an environment with the first two main agents of the system: the AGVs and the charging points.

The environment that will be created will resemble the environment in figure 5.29. However, several adaptions will have to be performed in order to comply with the considerations in section 5.3. Charging stations will have to be added as well as a buffer area in which unassigned robots can wait for luggage.

When the environment has been created, the first AGV will be generated. This work package will only consider a single AGV which will move from a random point to one of the charging stations. Also kinematics will be added for the AGV according to data provided in [4]. At the same time, when an AGV is moving, its energy level will decrease. Therefore, the interaction between AGV and charging station will also be established. It is assumed that, under normal conditions, AGVs will charge fully.

WP2 (1.5 weeks): Task execution

In this second work package, tasks will be generated with random delivery chutes. Since the shortest path already has been established, the agent will move between the infeed and a defined chute. It should be noted that while executing tasks, the energy level of the AGV will decrease and the AGV will have to make sure it is charged before taking up a new task.

When the AGV is successfully moving between tasks, priority bag generation will be added to the infeed. Three priority levels will be defined, each one accompanied with a score as a means to inform the system about the importance: low priority (0), intermediate priority (1) and high priority (2).

Since most generated bags will have low priority, the AGV will get assigned the low priority tasks on a first-come first-serve basis.

[1] http://ccl.northwestern.edu/netlogo/
[2] https://www.python.org/

WP3 (1.5 weeks): Separation
Now, multiple AGVs will be generated. Since AGVs will overlap, separation will have to be introduced. Since it has been chosen to use ECBS+HWY in an iterative way, it was assumed that an extensive short-term planning method would not be required. However, to guarantee safety during operations, it was decided to implement basic separation techniques. Think about minimum separation between AGVs, AGV interaction at crossing points, head-on conflicts,... Separation requirements have been presented in [4] and will thus be taken from there.

WP4 (~9 weeks): Implement ECBS+HWY
After about 7 weeks, the first main part of the research can begin, the implementation of ECBS+HWY. The first step will be to become familiar with implementations of conflict-based search and the implementation of highways. Before starting the implementation of ECBS, highways will first be added to the system to predefine the streams of AGVs.

During the implementation of ECBS, it should be determined how to deal with different groups of AGVs. These different groups can be for example, according to [14], new task agents and free agents. Furthermore, since new tasks appear continuously, ECBS should be able to frequently update all plans to include priorities of new tasks, agents needing to charge and ultimately to adapt to the task allocation. When ECBS+HWY has been implemented in the first environment, the feasibility of adding kinematics will be assessed. If variable speeds would be too complex, the choice can be made to use different AGV speeds for different priorities.

An additional element to test the dynamic behaviour of the implementation of ECBS+HWY is to frequently let an AGV be 'broken'. This will lead to vertexes being inaccessible at some times. By frequently replanning it is expected that ECBS will make sure that the failed AGV is avoided at all times.

WP5 (~9 weeks): Task allocation
The final work package is again a big one. The auction mechanism will be implemented and will have to be combined with the path planning method in order to create a pickup-and-delivery solution. The idea is to let the bids of the AGVs on certain tasks be the time to that task that is calculated using ECBS. The battery level of the AGVs also influences the bidding process since it is not desired to let AGVs charge while executing a task.

The auction method will first be implemented without task swapping and without being incorporated with ECBS in order to eventually compare both separate task assignment and path planning and their combined version. Finally, IDMB will be implemented to allow for distributed task swapping. It should be noted that priority bags are still being generated and the auction mechanism will have to be able to deal with it accordingly.

6.3. Planning

This final section shows the planning of the previously presented work packages together with dates for the milestones and major deadlines of the upcoming research. The milestones proposal can be found in table 6.1 and the schedule of the work packages is shown in table 6.2 and in the Gantt chart in figure 6.1.

Table 6.1: Dates of milestones and deadlines (proposal).

Milestone	Date
Kick-off meeting	13-03
Mid-term review	05-06
Thesis draft	09-10
Green light review	16-10
Final submission	30-10
Thesis defence	13-11

[3] Note that the times shown do not add up to the estimated time per work package, this is because subpackages overlap as visualized in figure 6.1.

Table 6.2: Timing schedule of the work packages from section 6.2.2.

Work package	Content	Time[3]
WP1: AGV charging (single AGV)		**4 weeks**
WP1.1 - familiarize	• Familiarize with ABM • Visualize environment 1 (Kiva)	*3 weeks*
WP1.2 - create AGV	• Add shortest path algorithm • Add kinematics	*3 weeks*
WP1.3 - create charging stations	• Create charging agent • Add AGV interaction	*1 week*
WP2: Task execution (single AGV)		**1.5 weeks**
WP2.1	• Create infeed and chute agents • Generate tasks and execute them • Generate priority bags	
WP3: Separation		**1.5 weeks**
WP3.1	• Add basic separation techniques • Consider priority bags	
WP4: Path planning		**~9 weeks**
WP4.1	Familiarize with (E)CBS implementations	*TBD*
WP4.2	ECBS+HWY (iterative)	*TBD*
WP4.3	Analyze model	*TBD*
WP5: Task allocation		**~9 weeks**
WP5.1	Implement auction mechanism	*TBD*
WP5.2	Introduce task swapping	*TBD*

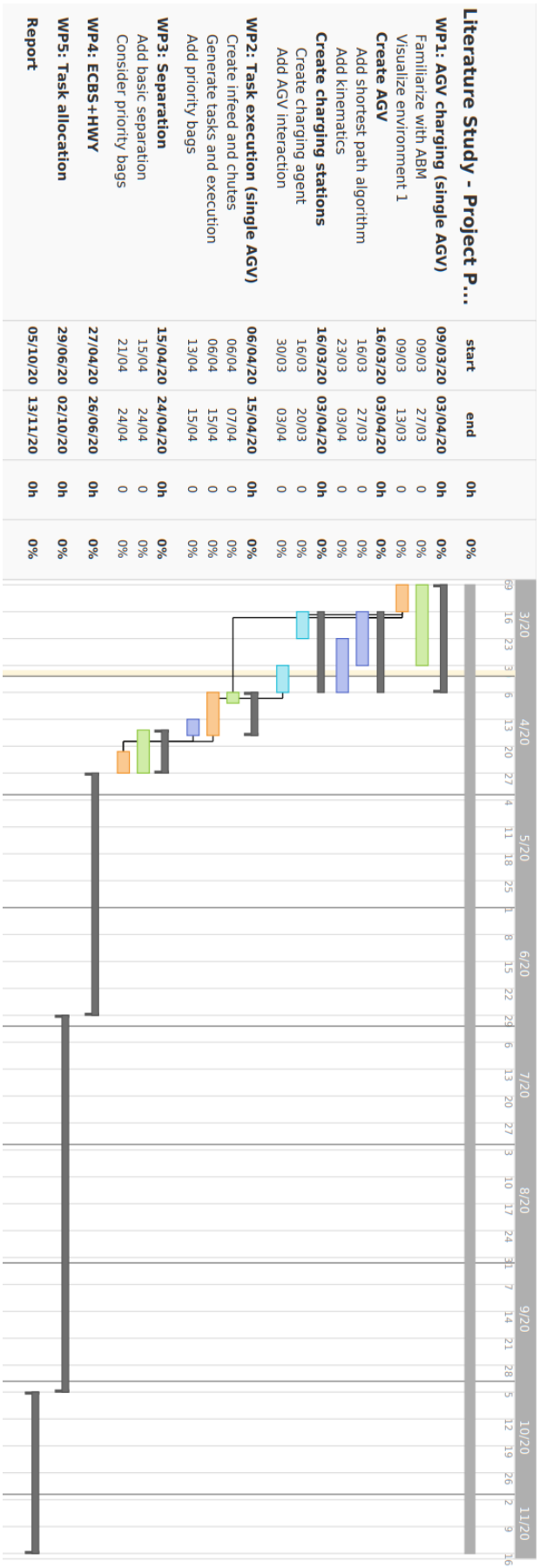


Figure 6.1: Gantt chart presenting the workflow of ABM development.

III

Supporting work

1

Agent properties

This appendix presents the properties of the agents and their interactions which are visualized in figure 1.1.

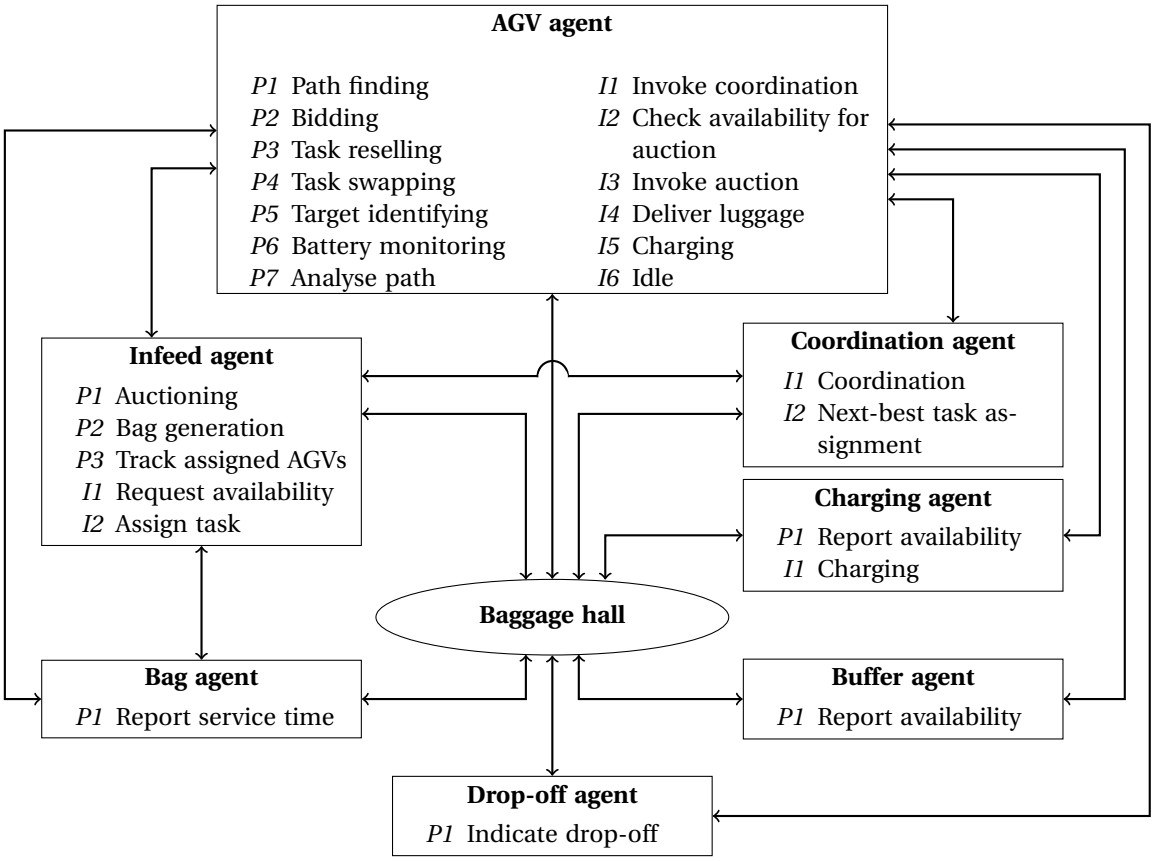


Figure 1.1: Agent properties and interactions

1.1. Charging agents

The AGVs in the system are only given a limited amount of energy. Because of this, they are required to go charge before their battery has fully drained. The charging agents therefore represent the charging infrastructure for the AGVs which can be reserved by the AGVs and charges them.

P1 - Report availability property: This property considers the availability of charging stations in which the charging agents report to the system when it has been reserved or has become available. This way, the AGVs

can more rapidly determine which stations are free to reserve.

I1 - Charging property: The *charging property* is an interaction between a charging agent and an AGV. When an AGV is located on the charging station, the charging station charges the AGV to a defined level. The AGV is aware of this limit and will re-enter the MAPD process when the charging limit is reached. At this point, the charging agent invokes its *report availability property* to update the system.

1.2. Buffer agents

The buffer area has the means to provide the AGVs with a place to wait for luggage or for a charging station to become available. Each cell in the buffer area is represented by a buffer agent that communicates with the environment to report its availability.

P1 - Report availability property: This property is equal to *P1* for charging agents. The buffer agents notify the system of their availability which allows AGVs to access that information when required.

1.3. Bag agents

Bags can already be tracked in current baggage systems. Because of this, each bag that is generated in the model is represented by a bag agent. Bag agents are aware of their arrival time, drop-off point and the AGV they are assigned to. This allows the system to track travel times of bags and identify the carrying AGV quickly. Bags are generated at each infeed station at a user-defined rate. This makes it possible to analyse the capacity of the system and assess how the performance alters between different amounts of AGVs, spawn rates and different task allocation methods.

P1 - Report time property: An important performance indicator of the created MAPD algorithm is the time 'tasks' spend unattended. Therefore, when a bag has been picked up for delivery, the waiting time is reported to the system, this is the time from entering the system till being picked up by an AGV. In the same way, the service time is reported when the bag has been delivered. This is the time from entering the system to being delivered.

1.4. Infeed agents

P1 - Auctioning property: The auction can be run at three different moments. The first is whenever an unassigned bag is present on the infeed station. This can be a bag that has just been generated (*bag generation property*) or a bag that had been generated in a previous time step but had not been assigned yet due to no AGVs being available at that time. The infeed agents initially start locating AGVs within a small search radius. This search radius gradually increases depending on whether identified AGVs have reported to be available or not (*request availability property*). If available AGVs are found within the search radius, the infeed agent asks those AGVs to report their bid: their distance to the infeed and their energy level. The infeed agent then selects the closest AGV that has a sufficiently high energy level to complete the delivery. It should be noted that an AGV can win multiple bids if multiple auctions are run during a single time step. The second and third moment are respectively during task reselling and during next-best task assignment in ECBS-TA. In both of these, bags are possibly re-auctioned while taking into account constraints imposed on the involved infeed agent in which specific assignments are excluded. During DMB, an AGV that has won multiple bids resells the tasks that take the longest to reach. For these tasks, the auction is then run again by the corresponding infeed agents while neglecting the originally assigned AGV. During ECBS-TA, the next-best task assignments are found by re-auctioning the involved bags. The auction is run for the bag (and infeed station) that could not be part of the solution in a child node, neglecting the AGVs that were disallowed in its parent node.

P2 - Bag generation property: The infeed agents generate bags a user-defined rate (system capacity). Each infeed agent keeps track of which bag has appeared first and which thus has to be auctioned first. Bags are characterized by arrival time, a drop-off point and eventually an AGV to which it is assigned. Bags can (1) be auctioned directly when the infeed station is empty or (2) wait in line to be auctioned.

P3 - Track assigned AGVs property: To ease the AGVs' *task swapping property* process, it is important that the infeed agent keeps track of the agents that are assigned to a bag on its station. In this way, AGVs can easily

identify specific AGVs to communicate with by first communicating with the infeed agents.

I1 - Request availability property: This property is an interaction between the infeed agent and AGVs. Whenever a bag is ready to be auctioned, the corresponding infeed agent communicates with all AGVs to invoke their *check availability for auction property*. This is based on their remaining energy, their position in the grid and whether or not they are already assigned to a bag or are transporting one. The identified available AGVs are then considered further throughout the auction.

I2 - Assign task property: The *Assign task property* is again an interaction between the infeed agent and an AGV. During this interaction, the required information for pick-up is communicated, this includes the pickup point (i.e. infeed agent) and the drop-off point of the bag.

1.5. Drop-off agents

The chutes in the environment are all accompanied by a drop-off agent. The task of the drop-off agents is to inform the system when a bag has been delivered. The AGV and the drop-off agent correspond (*drop off bag property*) when the AGV is at the correct chute at which point the AGV releases its bag and the bag removes itself from the system.

P1 - Indicate drop-off property: If an AGV has delivered a bag, the drop-off agent indicates that it has received a bag by lighting up for a short period of time. After this, the bag is removed from the system.

1.6. AGVs

The AGVs are the biggest part of the agent-based model. They are required to optimally transport bags throughout the baggage hall. There are four major processes an AGV can follow: (1) Moving to the assigned infeed, (2) transporting the bag to the right chute, (3) finding and moving to a charging station and (4) waiting for a task (buffering).

To be able to fulfil these processes, several other functions have to be completed. The most important are mentioned below.

P1 - Path finding property: As path finding mechanism, the AGVs use the A* algorithm[1] that finds shortest paths based on the edge weights (i.e. edge costs) in the graph. The algorithm has been modified to make it possible that AGVs wait at the same location before progressing, to allow for taking into account temporary obstructions (e.g. stationary AGVs) and to take into account constraints imposed by ECBS(-TA).

P2 - Bidding property: If an AGV has declared itself as available and is not excluded from the auction during DMB or ECBS-TA, the AGV participates in the auction invoked by an infeed its *Auctioning property*. It reports its distance and energy level to the infeed agent. By providing its energy level, the infeed agent can decide whether the AGV is able to reach the drop-off point while ensuring sufficient time for the AGV to reach a charging station when running out of energy[2].

P3 - Task reselling property: If multiple bags are auctioned by different infeed agents, it is possible that an AGV ends up with multiple tasks. When this is the case, the AGV selects the task that suits it best (i.e. the closest one) and resells the remaining ones by re-invoking auctions for those tasks. Since the AGVs act out of self-interest when reselling tasks and do not keep the global goal in mind, i.e. minimizing bag waiting times, IDMB allows for swapping tasks between AGVs. Pseudocode for task reselling can be found below in algorithm 2.

[1] Networkx in Python (`https://networkx.org/`).

[2] Optimizing the AGVs' battery level was not within the scope of this research. We therefore did not performed additional checks to guarantee the energy level to not run below zero.

Algorithm 2: Task reselling (DMB)

Input: A list of AGVs that have received a task in the current time step, DMB_list
Output: Assigned tasks based on self-interest
$bids\_won \leftarrow$ Tasks assigned to agv after CNP
while DMB_list not empty **do**
    for agv in DMB_list **do**
        if $agv_{bids\_won} > 1$ **then** ▷ AGV has more than 1 task assigned → resell
            Assign best bid to self ▷ Based on distance to infeed and energy level
            $sell\_bids \leftarrow$ List with tasks to resell
            for bid in $sell\_bids$ **do**
                Unassign bid from self
                $bag, inf \leftarrow$ Bag to resell & corresponding infeed agent
                $agvs\_available \leftarrow$ List with available AGVs identified by inf
                if $agvs\_available$ **then**
                    $inf.auction(bag, resell)$ ▷ Re-auction and add winner to DMB_list
                else
                    Bag will be re-auctioned next time step
                end if
            end for
        else
            Assign best bid to self
        end if
        remove agv from DMB_list
    end for
end while

P4 - Task swapping property: Task swapping can occur in two stages in a simulation step: (1) after a regular auction and potentially reselling tasks and (2) during next-best task assignment in ECBS-TA. Both differ in which AGVs are considered for swapping. In the first option, the AGVs that have been assigned to a bag in the current time step ask all AGVs that are assigned to the other infeed whether it would be beneficial for minimal waiting times of bags to swap tasks. In the second option, the newly added AGV in the next-best task assignment node considers the other AGV(s) in the same assignment node to swap tasks. The pseudo-code for task swapping is shown below in algorithm 3.

Algorithm 3: Communication between agents to check for task swapping (IDMB)

Input: Local information of AGV agent asking for task swapping, AGVs to consider for task swap
Output: List of possible task swaps, $swap\_tasks$, for considered AGV
$self \leftarrow$ AGV agent asking for task swapping
$infeed_{self} \rightarrow$ Infeed assigned to $self$
$agvs\_for\_swapping \leftarrow$ AGVs assigned to other infeed than $infeed_{self}$
$swap\_tasks \leftarrow$ empty list ▷ Store AGVs with which to swap
for agv **in** $agvs\_for\_swapping$ **do**
    $cost_{tot} \leftarrow$ Total original path cost of both agents
    $swap\_cost_{tot} \leftarrow$ Total path cost of both agents when swapping
    if $swap\_cost_{tot} < cost_{tot}$ **then**
        $swap\_tasks \leftarrow$ Add agv ▷ Consider AGV for swapping
    end if
end for
return $swap\_tasks$
if $swap\_tasks$ **then**
    $agv\_swap \leftarrow$ agv with lowest $swap\_cost_{tot}$
    $task\_swap(self, agv\_swap)$ ▷ Exchange bag & infeed properties
end if

P5 - Target identifying property: At each time point, AGVs check which function they are executing and whether they have a bag assigned (this can change at any time due to the *task swapping property*). Based

on this, it decides to update its target to a buffer area, a new bag or to continue moving to its current target.

P6 - Battery monitoring property: The energy of an AGV drains at every tick (one level when stationary, two otherwise). When its energy eventually reaches a user-defined minimum, the AGV decides it has to go charge after its current task is finished. After finishing its task, it determines the closest free charging station and reserves it. Having updated its target cell to one of the available charging spots, the AGV follows its conflict-free path to the station and starts charging.

P7 - Analyse path property: To be able to assess the performance of ECBS-TA, whenever an agent receives a new target, the AGV calculates its shortest path to the target and starts storing its actual path. After reaching the target, the AGV analyses how much longer it took to reach it goal and how much further it travelled compared to the initial shortest path. The time takes into account the difference in amount of time steps, the distances checks how many more cells have been covered during travelling.

I1 - Invoke coordination property: AGVs can update their characteristics in order to let the coordination agent invoke the ECBS or ECBS-TA algorithm. If one of the AGVs has received a new target cell or has become stationary, it informs the system that coordination should occur.

I2 - Check availability for auction property: When an auction starts, AGVs are asked by the infeed agent to report their availability. The availability is based on the following conditions: (1) the AGV is charged, (2) the AGV has no bag assigned and (3) the AGV is not delivering a bag. There is however an exception to the third condition. An AGV can, prior to having delivered its bag, be assigned to a new one when it is located in one of the outer corridors at the chutes.

I3 - Invoke auction property: This property is an interaction between an AGV and an infeed agent. This property lets an AGV indicate what kind of auction has to be run. This can either be during reselling a task or during next-best task assignment in ECBS-TA. Based on the information provided by the AGV, the infeed agent knows which AGVs to exclude and whether certain information should be assigned or updated to an AGV during the infeed agent's *assign task property*.

I4 - Deliver luggage property: Having reached its assigned infeed station, the infeed agent hands over the bag to the AGV which it continues to deliver. At this point, the AGV updates its target to the chute at which the bag has to be delivered. Furthermore, the bag updates its service time by its *tracking property*. When the AGV eventually reaches the bag's drop-off point, it drops its bag in the chute which notifies the system through its *baggage receival property*.

I5 - Charging property: The *charging property* is an interaction between an AGV and a charging agent. After deciding it has to go charge (*battery monitoring property*) and after having completed its task, the AGV reserves one of the available charging stations. The AGV selects the closest one available, reserves it and updates its target to the charging spot. If no charging station is available, it invokes its *idle property* to claim a buffer spot while waiting for a charging station to become available.

I5 - Idle property: If the *find charging station property* is unsuccessful or the AGV has no bag assigned, it chooses a random free spot in one of the buffer areas to wait for a charging station to become available or for being assigned to a bag.

1.7. Coordination agent

I1 - Coordination property: To provide conflict-free paths for the AGVs, a coordination agent is added to the system which contains and runs the ECBS(-TA) algorithm. The coordination agent is called upon depending on whether (1) a certain amount of time has passed, (2) AGVs have received a new goal or (3) AGVs have become stationary. Based on whether new goals were obtained due to the task allocation mechanism, the agent invokes ECBS or ECBS-TA. The coordination agent communicates with the AGVs to let them provide the required information: the task allocation, stationary AGVs, previously coordinated paths of AGVs

and AGVs that require a new path. After coordination, the agent communicates the computed conflict-free paths and, in case of ECBS-TA, possibly updated task assignments (*next-best task assignment property*) to the corresponding AGV agents.

I2 - Next-best task assignment property: During ECBS-TA, it can be required to find an alternative task assignment to optimize the TAPF instance. The next-best task assignment mechanism makes use of a constraint tree to find the next-best assignment. In this constraint tree, the root node consist of the task allocation provided by IDMB. A next-best task allocation is found by creating child nodes which contain (1) assignments that must be part of the solution and (2) assignments that cannot be part of the solution. In each child node, one of the assignments of the parent node is disallowed and re-auctioned. To provide ECBS-TA with optimal next-best assignments, the AGV that won the repeated auction can ask the other AGVs in the node to swap tasks. A more detailed explanation is provided in section 3.3.

2

Baggage handling system capacity

The required calculations to obtain the operational capacity of baggage handling systems [30]:

The line design capacity indicates the amount of bags that are generated by a single infeed station in a certain time window.

$$\text{Line design capacity} = \frac{\text{Time window}[ticks]}{\text{Minimum time separation}[ticks]} \tag{2.1a}$$

The time window equals the simulation time in ticks, while the minimum time separation is the rate at which bags are introduced on the infeed.

The system capacity is the amount of bags that the system is designed to handle, i.e. the amount of bags that all infeeds in the system should be able to generate.

$$\text{System capacity} = \text{Line design capacity} \cdot \#\text{ infeeds} \tag{2.1b}$$

The operational capacity is determined by evaluating how fast bags are being picked up without having bags piling up.

$$\text{Operational capacity} = \frac{\text{Time window}[ticks]}{\text{Time separation in stable flow}[ticks]} \tag{2.1c}$$

3

Combining task assignment and path finding

This section explains the components of the hybrid MAPD algorithm in more detail.

3.1. MAPF: Enhanced Conflict-Based Search

CBS [16] is a hierarchical algorithm (i.e. it has a high-and low-level search) that solves the MAPF problem by creating a constraint tree in which each node contains a set of paths and a set of constraints for each agent. The high-level search identifies conflicts between agents and imposes a constraint for each involved agent in a newly created child node. In each new child node, the low-level search finds a new path for the affected agent, using any shortest path algorithm, taking into account the corresponding constraint(s). If a new path has successfully been found, the child node is added to the OPEN list, containing all nodes that have not been expanded yet. After this, the high-level search is again applied for the lowest cost node in OPEN after which the process repeats until a node without conflicts is found.

These sets of paths and constraints are created by the high-level (node expansion) and low-level (path finding) search. On the high level, paths are verified and constraints are added if a conflict between two agents is found. These exist of a place and time of conflict. For each involved agent, a child node is created in which the new constraint is added to its assigned constraints that were already present in the parent node. Then, the low-level search is invoked in which the A* algorithm finds a new path for the affected agents in their corresponding child nodes and adds them to an OPEN list. After this, the high-level search is again applied for the lowest cost node in OPEN and the process repeats until a node without conflicts is found.

ECBS differs from CBS in the sense that it does not only run the high-level search for the lowest cost node. It allows the user to define a sub-optimality factor w_{ecbs} which ECBS uses to create a FOCAL list containing all nodes from OPEN for which the cost lies within that user-defined factor of the lowest cost node. The high level of ECBS then expands the node with the least amounts of conflicts in FOCAL [76]. This way, solutions can be found more rapidly, at the cost of optimality.

3.2. TA: IDMB

IDMB starts off by (1) letting available AGVs bid on the tasks that appear randomly (*online*) to the system. These bids consist of their distance to the pickup point and their current battery level. The infeeds, that act as auctioneers, then select the AGV that best suits the task. IDMB allows AGVs to win multiple auctions during a time step, an AGV can thus have more than one task assigned after an auction round. The second step in IDMB therefore lets AGVs with multiple tasks (2) pick the task that suits itself best and resell the remaining tasks to other AGVs based on their initial bids. During reselling, AGVs can again end up with more than one task. Reselling is therefore repeated until none of the AGVs has more than one task assigned. Finally (3) the AGVs that are assigned to a task communicate with each other to determine whether or not swapping tasks would benefit the cost of the solution. This is done by comparing equations (3.1) and (3.2) [71].

$$C(t_i, r_i) + C(t_j, r_j) \tag{3.1}$$

$$C(t_i, r_j) + C(t_j, r_i) \tag{3.2}$$

In these equations, C represents the path cost in function of which AGV takes on which task. t_i and t_j respectively indicate the tasks initially assigned to AGVs r_i and r_j. In case $equation$ (3.2) < $equation$ (3.1), the AGVs decide to swap tasks.

The proposed method works when when optimizing for minimum service times (the time a task spends in an infeed) and the system only has two infeeds. However, when using multiple infeeds, bags rapidly start piling up in the middle ones since these are generally further away for AGVs that are returning from delivery. This can be explained by the fact that equations (3.1) and (3.2) optimize with regard to path cost. Therefore, AGVs only move to the outer infeeds since these are less costly to reach. To tackle this problem, the service times of the considered tasks have to be taken into account in the decision-making. This leads to two conditions of which at least one has to be satisfied to let AGVs swap tasks. Both conditions check whether one of the AGVs can reach the other AGV's bag more quickly and whether the other AGV's bag has been waiting on the infeed for a longer period of time (equation (3.3)):

If AGV i can reach the longer waiting bag of AGV j sooner:

$$Condition\,1: \quad C(t_j, r_i) < C(t_j, r_j) \quad \textbf{and} \quad t_{serv,t_i} < t_{serv,t_j}; \tag{3.3a}$$

Or, vice versa, if AGV j can reach the longer waiting bag of AGV i sooner:

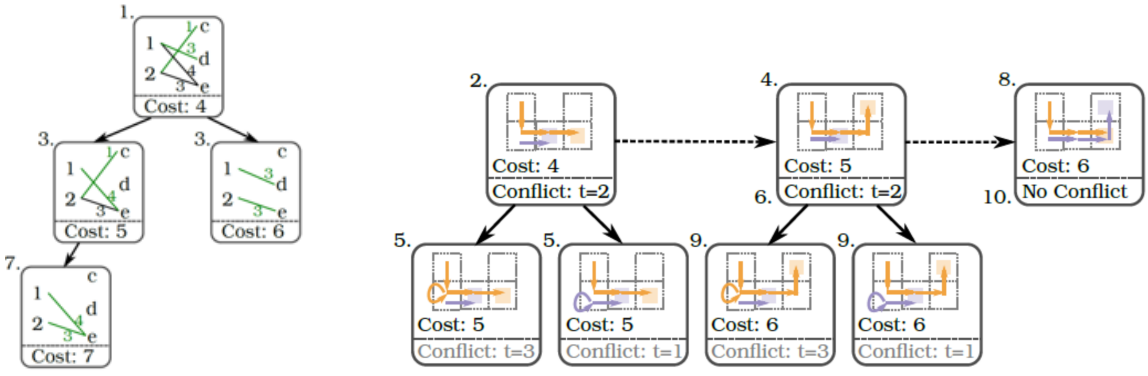
$$Condition\,2: \quad C(t_i, r_j) < C(t_i, r_i) \quad \textbf{and} \quad t_{serv,t_j} < t_{serv,t_i}; \tag{3.3b}$$

AGVs i and j swap tasks.

This way, IDMB optimizes the task allocation w.r.t minimum service times. Note that the approach in the scientific paper uses the method with equations (3.1) and (3.2).

3.3. TAPF: ECBS-TA

Figure 3.1 shows an example of the steps in a CBS-TA problem. The problem starts of with finding the optimal task assignment (node 1 in figure 3.1(a)) and creating the root node for the first constraint tree (node 2 in figure 3.1(b)). How new trees are created on demand is explained in section 3.3.1. Hereafter, section 3.3.2 explains the working principle of the next-best task assignment shown on the left in figure 3.1.



(a) Next-best task assignment

(b) Search forest

Figure 3.1: The working principle of CBS-TA for 2 tasks and 3 agents [12].

3.3.1. Root node creation

ECBS-TA still works with the FOCAL list, as presented in section 3.1, which contains the unexpanded nodes from the constraint tree(s). The nodes, however, do not only contain a set of constraints and a set of paths for each agent, they also contain the task assignment and information on whether the node is a root node or not. The latter is important for adding new constraint trees to the forest.

If the expanded node by ECBS is a root node, ECBS-TA can decide whether to expand the node and create a new root node or whether to expand the node and continue with the high-level search of ECBS. Three options exist:

- ***CBS-TA style***: Whenever a root node is expanded, a new root node is created.
- ***MaxRoot***: All root nodes with a cost lower than $w \cdot LB(n)$* are added, i.e. as many root nodes as possible.
- ***MinRoot***: When creating the first root node, a variable $r = w \cdot LB(n)$* is defined. A new root node is only added when the lowest cost node in OPEN exceeds r. A new root node is created and r is updated. As few root nodes as possible are added.

*In *MaxRoot* and *MinRoot*, w is the user-defined suboptimality factor of ECBS and *LB(n)* is the lowest cost node in OPEN.

When a root node has been expanded, based on one of the above methods, the next-best task assignment is invoked for (each of) the new root node(s).

3.3.2. Next-best task assignment

The next-best task assignment algorithm makes, just like ECBS, use of a constraint tree. An example is shown in figure 3.1(a). The optimal task assignment, found by IDMB, makes up the 'assignment root node' and is added to the ASG_OPEN list which contains all assignment nodes that are available for expansion. When it is decided to create a new root node, the next-best task assignment is invoked for the lowest cost assignment in ASG_OPEN.

The next-best task assignment mechanism works with two sets of assignments that are stored in the assignment nodes: assignments that must be part of the solution and assignments that cannot be part of the solution. The principle is similar to CBS. A child node is created for each of the assignments in the parent node in which that assignment is disallowed. The corresponding tasks are re-auctioned for each child node, considering the AGVs that are already part of the solution and those that are enforced to not be included in the solution.

As it is desired to provide ECBS-TA with optimal task assignments, IDMB is partly invoked when finding the next-best task assignment. For each newly created child node, an auction is run for the task in the disallowed assignment considering the present constraints in the parent node. Since in each node only one assignment is disallowed, only one task is auctioned at the same time and no reselling occurs. However, the newly assigned AGV can ask for task swapping with the AGVs in the same assignment node to optimize the new task allocation by again comparing equations (3.1) and (3.2).

3.3.3. Workflow diagram

A schematic view of the ECBS-TA algorithm is provided in figure 3.1. Note again that ECBS-TA follows the principle of ECBS (red dashed frame) with an additional task assignment loop (blue dotted frame). In the 'input' box it can be seen that it takes the task assignment that has occurred in the current time step (*Initial TA*), which can be empty if it has not taken place. Furthermore, it uses all stationary AGVs acting as obstacles during path finding, it uses previously coordinated paths for AGVs and the *new goal agents* as mentioned in section 1.7.

The problem starts with (1.) the root node containing the shortest path of each AGV, disregarding any conflicts. Now, the difference between CBS and ECBS, as mentioned in section 3.1, can be reduced to an extra list that is used while solving a MAPF problem. Where CBS uses the OPEN list that contains every unresolved node of the constraint tree, (2.) ECBS uses an additional FOCAL list. This FOCAL list contains every node from OPEN of which the cost lies within the user-defined suboptimality factor from the lowest cost node in OPEN. Instead of selecting the lowest cost node from OPEN, (3.) the high level of ECBS selects the node with the lowest amount of conflicts from FOCAL to validate. (4.) When conflicts are identified, the node is (5.) expanded

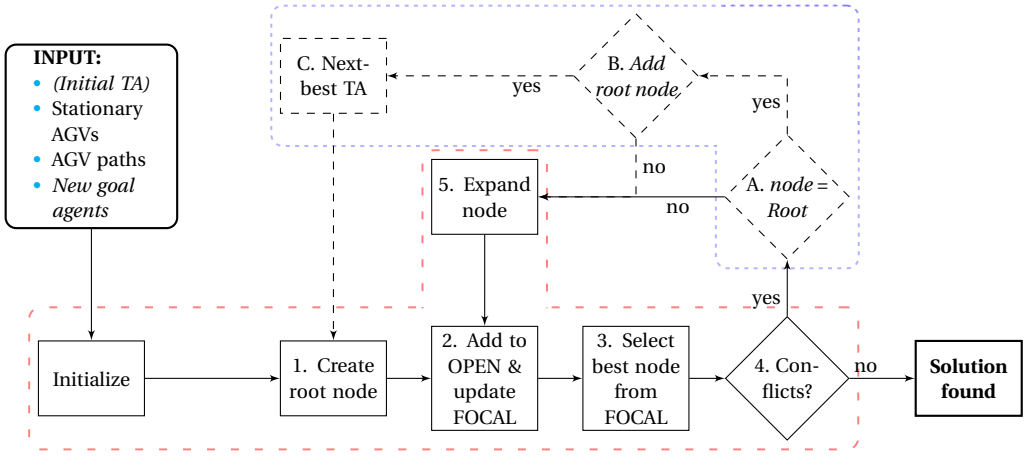


Figure 3.2: Workflow diagram of ECBS(-TA). The red dashed frame shows the working principle of ECBS. The blue dotted frame adds the steps for ECBS-TA.

(i.e. new shortest paths are found) and (2.) added to the OPEN list and, possibly, the FOCAL list.

When task assignment is included, the ECBS algorithm adds an additional loop: root expansion. When (4.) a conflict is identified by ECBS in the selected node, (A.) it is checked whether this node is a root node. If this is the case (B.) the algorithm checks whether a new root node should be created based on the selected method (MinRoot, CBS-Style, MaxRoot). In our case, MinRoot, the node cost is compared to the previously set threshold and if the cost exceeds threshold, (C.) the next-best task assignment is calculated.

In figure 3.1, the optimal solution assigns agent 1 to task D and agent 2 to task C. This is stored in the 'assignment root node'. This assignment root node is eventually expanded when looking for the next-best task allocation, leading to the creation of the assignment constraint tree. Nodes in the assignment constraint tree contain two sets of assignments: Assignments that must be part of the solution and assignments that cannot be part of the solution (constraints).

The principle of expanding the assignment nodes is similar to expanding nodes in the CBS constraint tree: There is an ASG_OPEN list containing all nodes in the assignment constraint tree that have not been expanded yet and the next-best task assignment algorithm first identifies the lowest cost assignment node from ASG_OPEN. The selected node is split into child nodes, disallowing certain assignments and enforcing others. In the example, this leads to the following solution: (1) enforce agent 1-D and disallow 2-C and (2) vice versa. Constraints are thus imposed on which agents can attend to which tasks. As the search tree grows, nodes will contain more and more constraints (i.e. disallowed assignments).

4

Results

This appendix shows extended results of the performed experiments.

4.1. Experiment 1

The first experiment included the selection of ECBS suboptimality factor and highway factor. The set-up of this experiment was a simulation time of 300 steps using 50 AGVs and w_{ecbs} and w_{hwy} factors of respectively $\{1.1, 1.3, 1.5\}$ and $\{1.5, 1.9\}$. No charging of AGVs was included in the process.

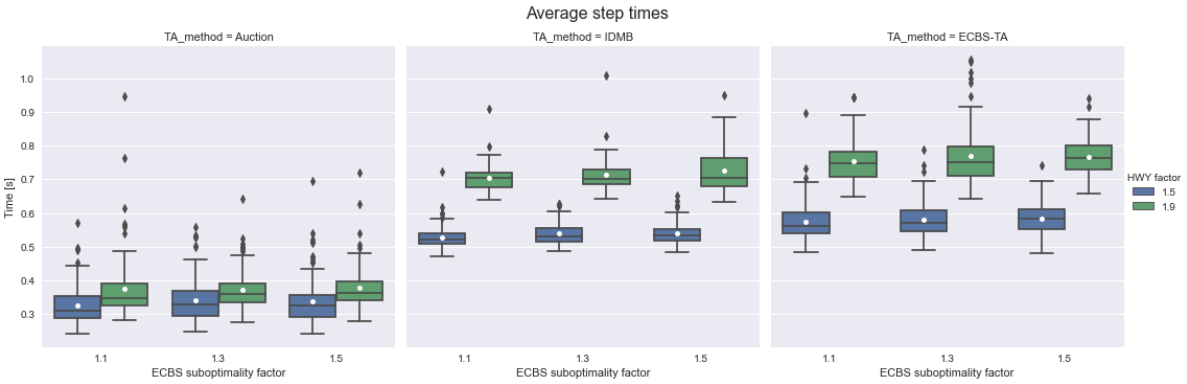


Figure 4.1: Average runtimes of the simulations in experiment 1.

Table 4.1: Runtime results from figure 4.1.

	Auction						IDMB						Hybrid MAPD					
w_{ecbs}	1.1		1.3		1.5		1.1		1.3		1.5		1.1		1.3		1.5	
w_{hwy}	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9
μ	0.326	0.374	0.342	0.373	0.338	0.377	0.526	0.705	0.538	0.713	0.539	0.725	0.575	0.753	0.579	0.769	0.585	0.766
σ	0.060	0.095	0.067	0.060	0.071	0.064	0.034	0.040	0.032	0.046	0.032	0.063	0.060	0.062	0.055	0.085	0.050	0.056
Med.	0.310	0.347	0.328	0.361	0.326	0.361	0.520	0.703	0.531	0.701	0.532	0.706	0.561	0.748	0.571	0.750	0.583	0.764
Min.	0.243	0.280	0.247	0.274	0.242	0.277	0.47	0.638	0.486	0.643	0.483	0.634	0.485	0.649	0.490	0.641	0.480	0.656
Max.	0.443	0.487	0.463	0.476	0.435	0.480	0.584	0.773	0.606	0.789	0.603	0.885	0.693	0.890	0.696	0.967	0.696	0.879

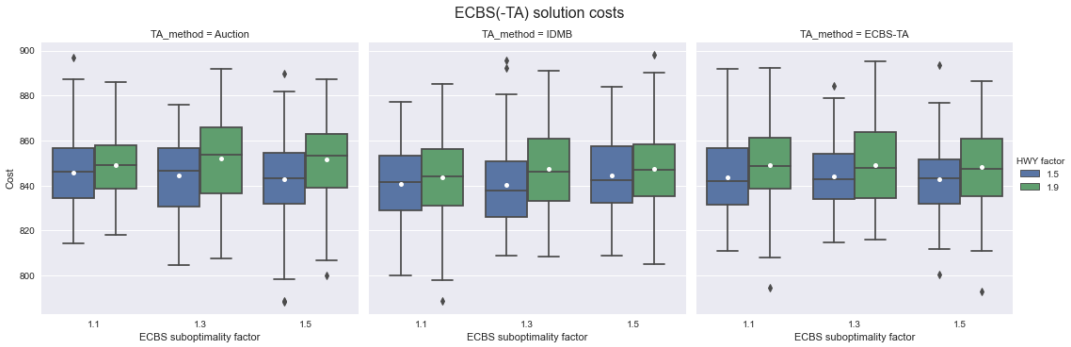


Figure 4.2: Average ECBS(-TA) solution costs of the simulations in experiment 1.

Table 4.2: Solution cost results from figure 4.2.

	Auction						IDMB						Hybrid MAPD					
w_{ecbs}	1.1		1.3		1.5		1.1		1.3		1.5		1.1		1.3		1.5	
w_{hwy}	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9
μ	845.7	849.2	844.4	852.0	842.8	851.6	840.8	843.8	840.2	847.5	844.5	847.5	843.5	849.1	844.0	849.2	842.6	848.4
σ	17.5	14.8	16.4	19.1	18.0	17.6	16.8	18.7	17.2	18.7	17.0	17.1	16.4	17.9	15.0	18.9	15.7	17.3
Med.	846.4	848.9	846.5	853.9	843.1	853.3	841.6	844.0	837.9	845.9	842.5	847.2	842.1	848.8	842.6	848.0	843.2	847.5
Min.	814.2	818.1	804.7	807.4	798.3	806.9	800.2	798.1	808.8	808.4	808.9	805.2	810.8	808.1	814.5	816.1	811.7	810.8
Max.	887.2	886.2	875.9	891.7	881.6	887.1	877.0	885.3	880.5	891.0	883.9	890.0	892.0	892.3	878.8	895.2	876.9	886.3

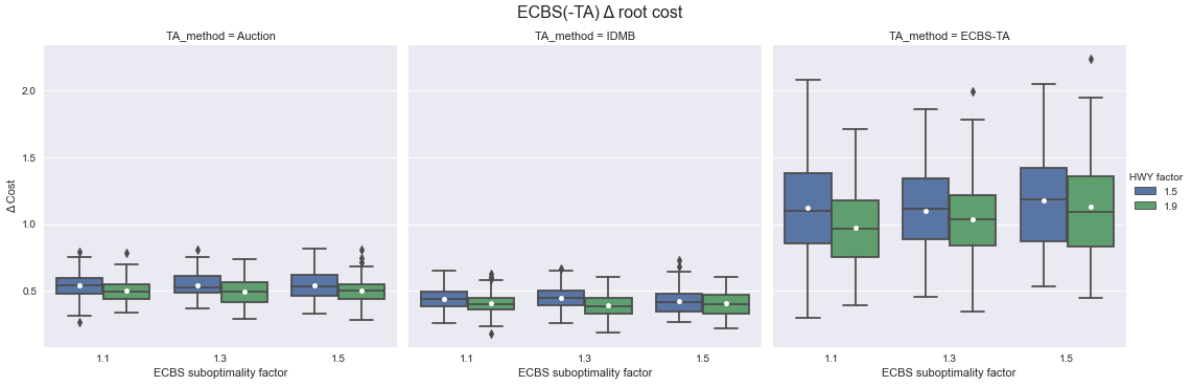


Figure 4.3: Average difference between ECBS(-TA) solution costs and initial root costs of the simulations in experiment 1.

Table 4.3: ΔCost results from figure 4.3.

	Auction						IDMB						Hybrid MAPD					
w_{ecbs}	1.1		1.3		1.5		1.1		1.3		1.5		1.1		1.3		1.5	
w_{hwy}	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9
μ	0.540	0.501	0.543	0.495	0.542	0.503	0.443	0.411	0.449	0.395	0.423	0.409	1.121	0.976	1.101	1.036	1.177	1.126
σ	0.098	0.087	0.086	0.092	0.104	0.092	0.078	0.083	0.092	0.090	0.093	0.087	0.377	0.291	0.327	0.307	0.371	0.368
Med.	0.539	0.491	0.530	0.494	0.531	0.500	0.442	0.397	0.450	0.384	0.413	0.404	1.1	0.968	1.112	1.036	1.185	1.093
Min.	0.311	0.338	0.372	0.289	0.331	0.283	0.239	0.239	0.259	0.188	0.264	0.219	0.298	0.393	0.457	0.347	0.538	0.447
Max.	0.752	0.697	0.750	0.735	0.817	0.687	0.58	0.58	0.651	0.608	0.647	0.602	2.08	1.707	1.859	1.781	2.044	1.949

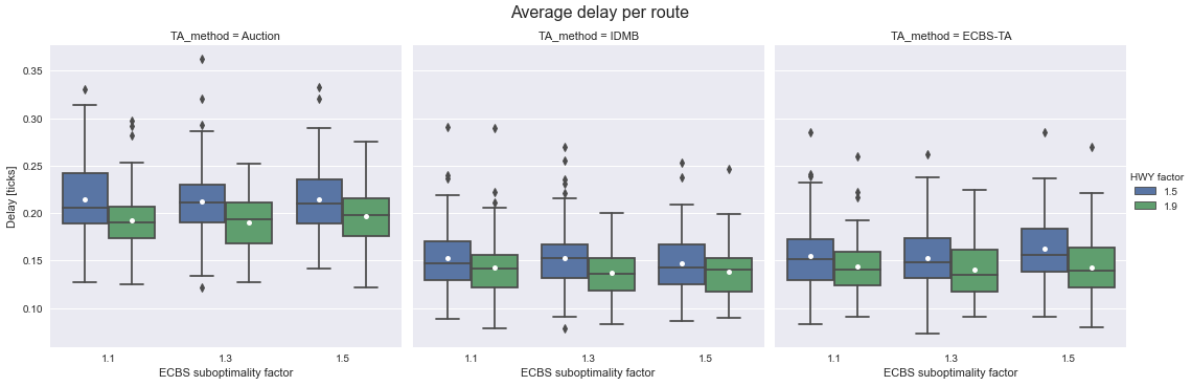


Figure 4.4: Average experienced delay by AGVs in experiment 1.

Table 4.4: Delay results from figure 4.4.

	Auction						IDMB						Hybrid MAPD					
w_{ecbs}	1.1		1.3		1.5		1.1		1.3		1.5		1.1		1.3		1.5	
w_{hwy}	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9
μ	0.215	0.192	0.213	0.190	0.215	0.197	0.152	0.143	0.153	0.137	0.148	0.138	0.155	0.144	0.153	0.141	0.163	0.143
σ	0.039	0.031	0.038	0.028	0.036	0.028	0.034	0.032	0.033	0.026	0.030	0.027	0.036	0.027	0.033	0.030	0.037	0.032
Med.	0.206	0.19	0.211	0.193	0.211	0.198	0.147	0.141	0.153	0.136	0.143	0.140	0.152	0.141	0.148	0.136	0.156	0.140
Min.	0.128	0.125	0.134	0.128	0.141	0.121	0.089	0.079	0.091	0.083	0.087	0.090	0.083	0.091	0.074	0.091	0.091	0.080
Max.	0.313	0.253	0.287	0.252	0.289	0.275	0.219	0.206	0.215	0.200	0.209	0.199	0.232	0.192	0.237	0.224	0.237	0.221

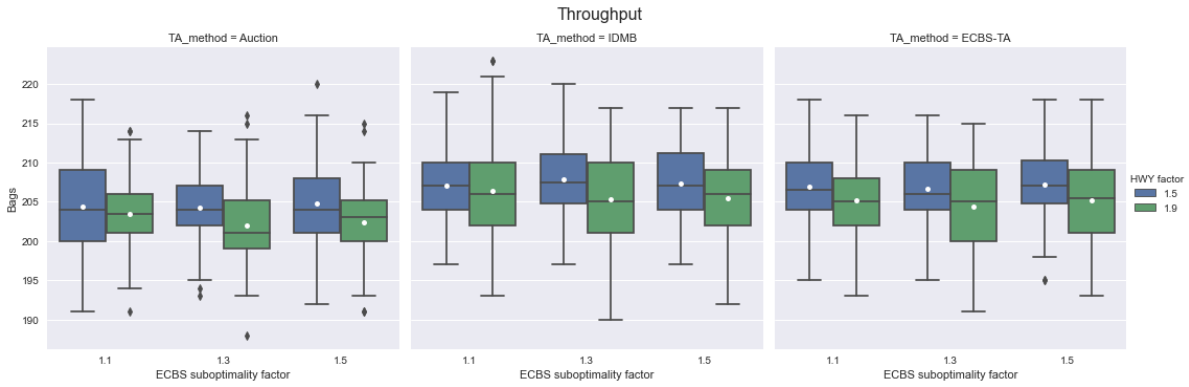


Figure 4.5: Average throughput of the simulations in experiment 1.

Table 4.5: Throughput results from figure 4.5.

	Auction						IDMB						Hybrid MAPD					
w_{ecbs}	1.1		1.3		1.5		1.1		1.3		1.5		1.1		1.3		1.5	
w_{hwy}	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9	1.5	1.9
μ	204.4	203.4	204.2	202.0	204.7	202.4	207.0	206.4	207.8	205.3	207.3	205.4	206.9	205.2	206.7	204.4	207.2	205.2
σ	5.4	4.3	1.2	5.1	5.4	4.6	4.7	5.8	4.5	6.0	4.9	5.2	5.0	4.7	4.2	5.3	4.4	5.2
Med.	204.0	203.5	204.0	201.0	204.0	203.0	207.0	206.0	207.5	205.0	207.0	206.0	206.5	205.0	206.0	205.0	207.0	205.5
Min.	191.0	194.0	195.0	193.0	192.0	193.0	197.0	193.0	197.0	190.0	197.0	192.0	195.0	193.0	195.0	191.0	198.0	193.0
Max.	218.0	213.0	214.0	213.0	216.0	210.0	219.0	221.0	220.0	217.0	217.0	217.0	218.0	216.0	216.0	215.0	218.0	218.0

4.2. Experiment 2

This experiment tested the scalability of the algorithms. The amount of AGVs was increased to test for real-time applicability and operational capacity. A simulation time of 300 steps was used for testing 60 and 70 AGVs with increased suboptimality factors up to 1.9 to increase flexibility. No charging of AGVs was included in the process.

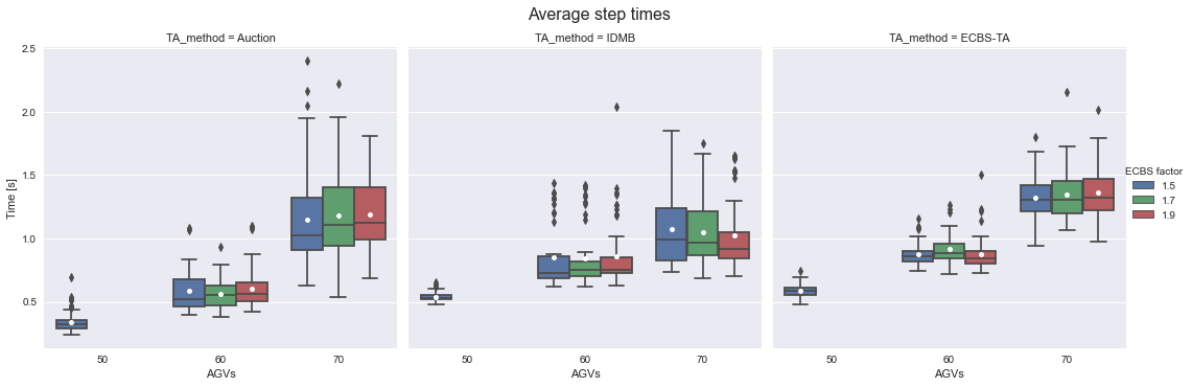


Figure 4.6: Average runtimes of the simulations in experiment 2.

Table 4.6: Runtime results from figure 4.6.

	Auction						IDMB						Hybrid MAPD					
AGVs	60			70			60			70			60			70		
w_{ecbs}	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9
μ	0.583	0.561	0.606	1.145	1.183	1.183	0.849	0.839	0.863	1.072	1.047	1.024	0.875	0.914	0.88	1.323	1.346	1.367
σ	0.155	0.122	0.159	0.379	0.365	0.283	0.248	0.234	0.268	0.288	0.252	0.275	0.090	0.118	0.139	0.176	0.199	0.215
Med.	0.522	0.551	0.561	1.023	1.108	1.127	0.731	0.749	0.756	0.992	0.967	0.914	0.860	0.888	0.842	1.308	1.307	1.322
Min.	0.397	0.381	0.419	0.628	0.535	0.684	0.622	0.619	0.625	0.732	0.684	0.699	0.74	0.719	0.731	0.946	1.066	0.977
Max.	0.838	0.797	0.879	1.951	1.960	1.81	0.874	0.896	1.018	1.848	1.669	1.297	1.013	1.1	1.018	1.687	1.727	1.792

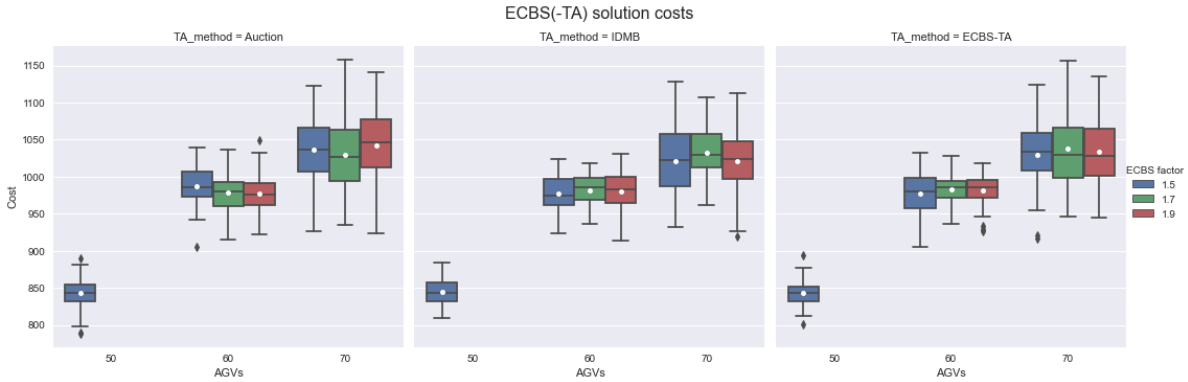


Figure 4.7: Average ECBS(-TA) solution costs of the simulations in experiment 2.

Table 4.7: Solution cost results from figure 4.7.

	Auction						IDMB						Hybrid MAPD					
AGVs	60			70			60			70			60			70		
w_{ecbs}	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9
μ	987.0	978.1	977.7	1036.3	1029.9	1041.5	977.4	982.0	980.5	1021.0	1031.8	1021.6	976.8	982.5	981.5	1029.5	1037.2	1034.0
σ	26.2	24.4	25.7	41.7	54.3	49.2	23.7	18.9	23.6	46.2	36.1	42.3	25.1	19.1	21.2	43.0	48.7	47.3
Med.	985.9	980.6	975.1	1035.9	1026.1	1046.8	974.3	985.1	982.6	1022.6	1028.9	1023.0	980.5	985.5	985.7	1033.0	1029.2	1027.8
Min.	942.1	915.7	922.1	926.0	934.7	923.2	923.5	936.7	914.2	932.0	961.5	926.5	904.8	936.5	945.8	954.7	945.6	944.2
Max.	1039.9	1036.2	1032.4	1122.0	1158.2	1140.4	1023.4	1017.9	1030.5	1128.1	1106.9	1112.0	1031.7	1028.6	1017.4	1123.3	1156.3	1135.0

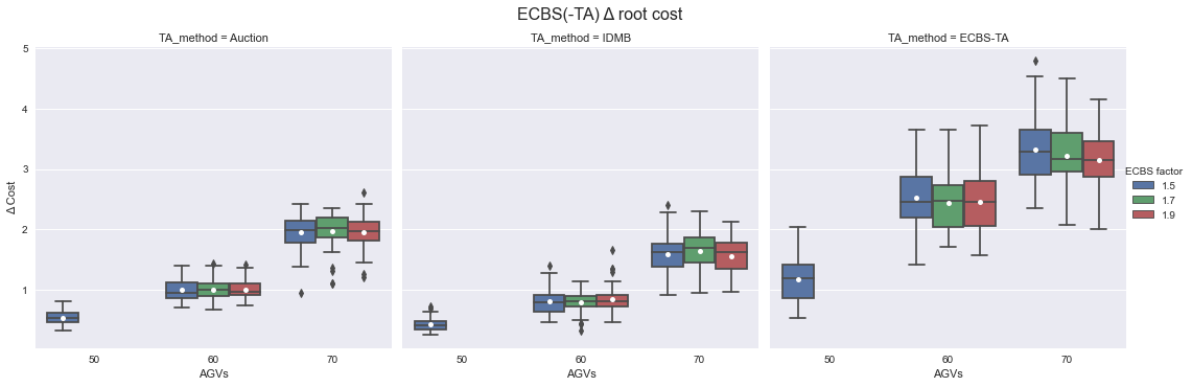


Figure 4.8: Average difference between ECBS(-TA) solution costs and initial root costs of the simulations in experiment 2.

Table 4.8: ΔCost results from figure 4.8.

	Auction						IDMB						Hybrid MAPD					
AGVs	60			70			60			70			60			70		
w_{ecbs}	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9
μ	0.995	0.997	1.009	1.952	1.978	1.955	0.804	0.799	0.846	1.583	1.647	1.550	2.525	2.444	2.459	3.319	3.219	3.156
σ	0.186	0.172	0.148	0.281	0.287	0.256	0.216	0.170	0.228	0.317	0.310	0.296	0.485	0.468	0.527	0.538	0.532	0.486
Med.	0.956	0.998	0.973	1.991	2.020	1.975	0.796	0.820	0.805	1.624	1.697	1.626	2.450	2.472	2.457	3.291	3.165	3.151
Min.	0.704	0.682	0.743	1.384	1.623	1.456	0.464	0.493	0.473	0.910	0.942	0.964	1.412	1.704	1.575	2.360	2.071	2.007
Max.	1.396	1.396	1.365	2.429	2.345	2.423	1.287	1.141	1.148	2.288	2.309	2.129	3.648	3.655	3.713	4.530	4.495	4.154

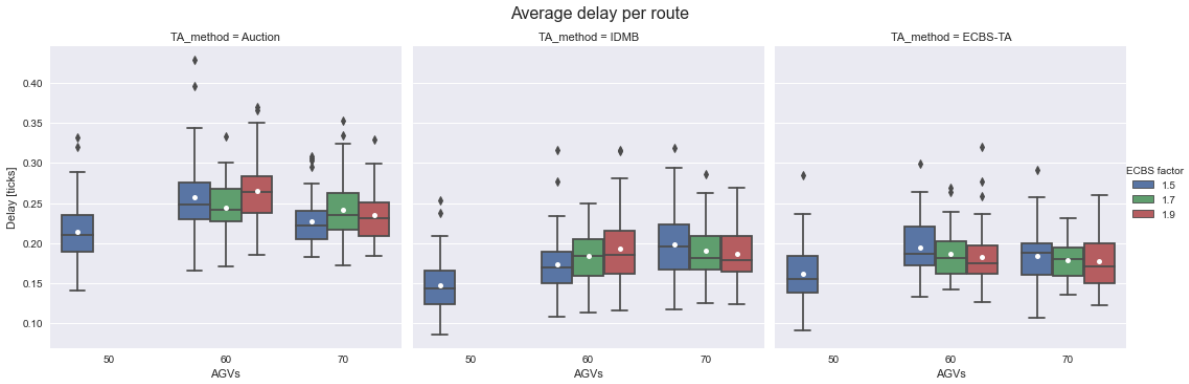


Figure 4.9: Average delay experienced by AGVs in experiment 2.

Table 4.9: Delay results from figure 4.9.

	Auction						IDMB						Hybrid MAPD					
AGVs	60			70			60			70			60			70		
w_{ecbs}	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9
μ	0.257	0.244	0.265	0.227	0.242	0.235	0.173	0.184	0.193	0.199	0.191	0.187	0.195	0.187	0.184	0.185	0.179	0.177
σ	0.050	0.033	0.041	0.032	0.041	0.033	0.038	0.035	0.043	0.043	0.034	0.034	0.036	0.031	0.037	0.037	0.023	0.037
Med.	0.248	0.242	0.264	0.223	0.235	0.232	0.170	0.184	0.185	0.195	0.182	0.179	0.186	0.181	0.175	0.188	0.181	0.171
Min.	0.166	0.172	0.186	0.183	0.173	0.185	0.108	0.114	0.116	0.118	0.126	0.127	0.133	0.142	0.126	0.108	0.135	0.122
Max.	0.343	0.301	0.350	0.274	0.324	0.300	0.234	0.250	0.282	0.294	0.262	0.270	0.264	0.240	0.237	0.257	0.232	0.260

Table 4.10: Throughput results from figure 4.10.

	Auction						IDMB						Hybrid MAPD					
AGVs	60			70			60			70			60			70		
w_{ecbs}	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9	1.5	1.7	1.9
μ	232.5	235.1	235.1	245.7	245.9	245.2	237.2	237.1	236.8	246.8	247.1	246.7	237.6	237.3	238.0	246.8	248.9	247.6
σ	4.7	5.8	5.8	4.0	5.0	3.4	3.7	4.7	5.1	3.6	4.2	3.8	5.4	4.5	5.2	4.0	4.4	3.9
Med.	233.0	235.0	235.0	245.0	245.0	244.5	237.0	236.5	237.0	245.5	247.0	247.0	238.0	237.0	237.0	246.0	249.5	248.0
Min.	219.0	226.0	226.0	239.0	237.0	240.0	228.0	228.0	225.0	241.0	238.0	240.0	227.0	225.0	226.0	238.0	240.0	240.0
Max.	240.0	247.0	248.0	254.0	257.0	251.0	244.0	244.0	245.0	256.0	257.0	254.0	251.0	248.0	250.0	258.0	256.0	254.0

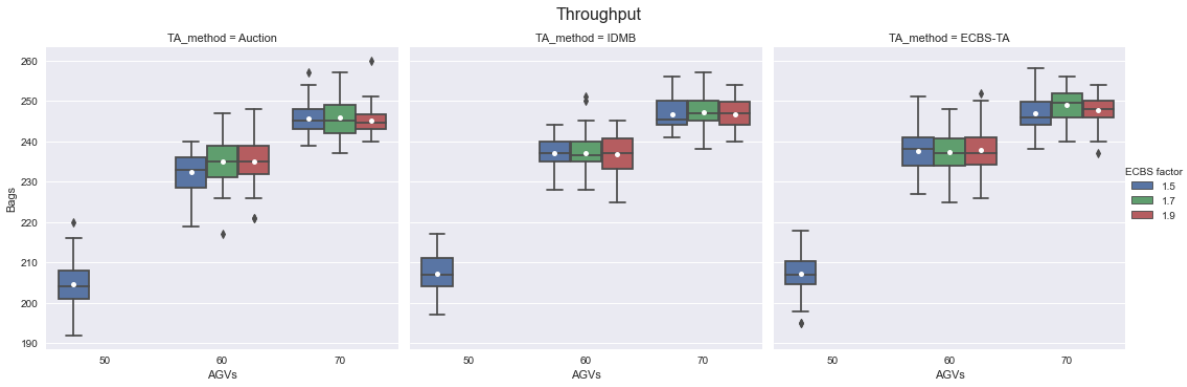


Figure 4.10: Average throughput of the simulations in experiment 2.

4.3. Experiment 3

The final experiment assessed the global performance of the system for the three algorithms using its best performing configuration from experiment 2.

- Auction: ECBS(1.7)+HWY(1.5)
- IDMB: ECBS(1.5)+HWY(1.5)
- Hybrid MAPD: ECBS(1.5)+HWY(1.5)

The heat maps mainly showed the concentration of conflicts at the infeeds for Auction+ECBS (figure 4.11), while the other algorithms (figures 4.12 and 4.13) clearly show the conflicts being spread out throughout the baggage hall. The colours do not fully represent the amount of conflicts that were identified (and solved) between the three algorithms as they are relative to the maximum amount of conflicts that were solved.

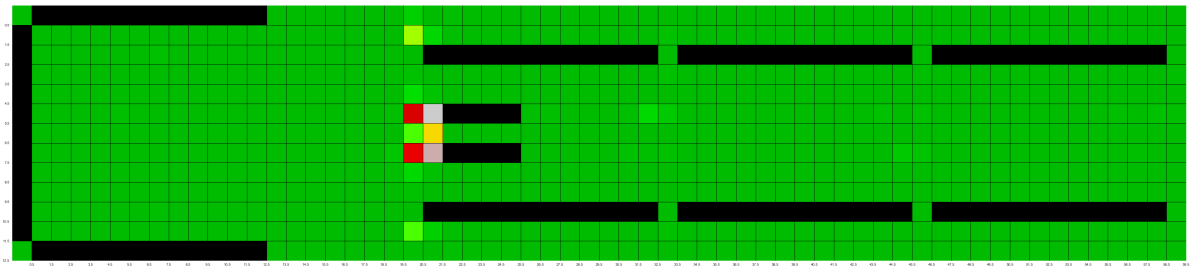


Figure 4.11: Heatmap Auction+ECBS

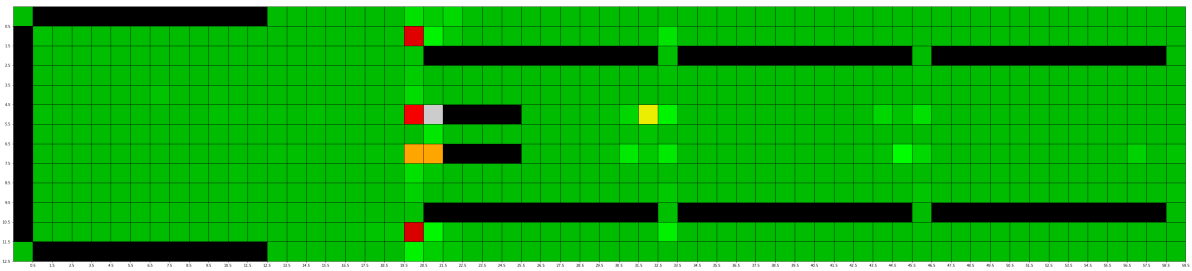


Figure 4.12: Heatmap IDMB+ECBS

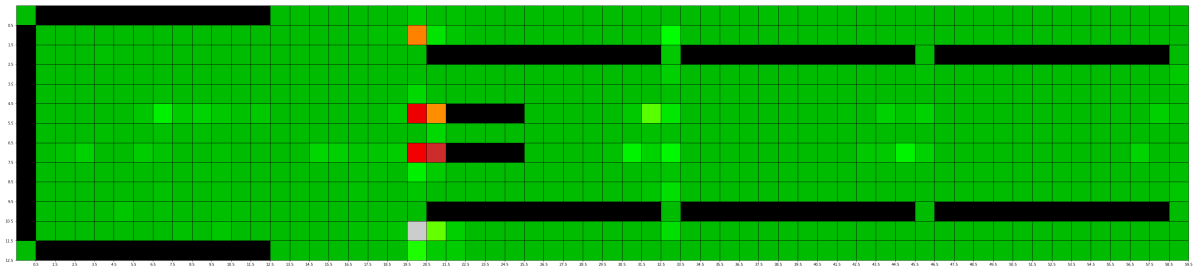


Figure 4.13: Heatmap hybrid MAPD

5

Model

5.1. Architecture

model.py This file contains the whole operational process. The model makes sure the different agent classes are activated in the correct order. Highways can be manually implemented here as well.

agents_agv.py This file contains the behaviour of the **AGV agents**. The AGVs have functions that can be activated individually like those needed for task assignment or for advancing a step. A full process for the AGVs however starts with checking whether they have a bag assigned and checking their battery level. These might update their current characteristics or leave them as is. Depending on the identified characteristics, the AGVs decide whether to *go_charge()*, *go_buffer()*, *go_pickup()* or *go_dropoff()*. To be able to perform these functions, several others exist in which it, among others, updates its goal, reserves a charging or buffer spot and moves forward.

agents_infr.py This file contains the behaviour of three classes of agents. First, the **bag agents** are used to calculate waiting and service time. During task assignment, the bags are used to identify AGV agents and be able to access their information. Then, the **infeed agents** represent the infeed stations. These invoke auctions when required, communicate with AGVs to report their availability and assign bags to best bidder. Specific infeed agents can be asked to re-run an auction during task reselling or during ECBS-TA. Finally, the file contains the **drop-off agents**. These are used to display at which chute a bag has been delivered.

agents_env.py This file initiates the lay-out of the baggage hall. The user can define the positions at which the infrastructural elements (infeeds, chutes, ...) are located. Furthermore, the **charging agents** and **buffer agents** are defined here which are aware of their occupancy. The chutes and infeeds that are defined in this file are used for display. Also **path agents** are defined to provide a heat map.

coord_ecbs_ta.py This file contains the ECBS(-TA) algorithm which returns a set of coordinated paths (and task assignments) for the AGVs.

coord_astar.py The A* method has been retrieved from the NetworkX[1] package in Python. This file contains a modification of that algorithm so that waiting at a position can be considered during path finding.

nodes_dir.py This file contains the graph along which AGVs can move. It uses the *agents_env.py* file but requires manual modifications to adapt the graph.

[1] https://networkx.org/

Schedule.py Activates agents in random order. Modified to allow for agent activation per group (i.e. AGVs, infeed agents, ... separately), as retrieved from MESA Boxworld on github[2] and to invoke the AGV *advance()* function which lets AGVs move one step forward along their path.

Run.py This is used to run the agent-based model (defined in *model.py*) and visualize it in your browser.

Server.py This file defines the visuals from *Run.py*. The grid in which the simulations take place is defined and the shape and colour of agents can be chosen. Furthermore, plots can be generated and real-time results can be shown.

Batchrunner.py The batchrunner allows for running multiple simulations successively and the user can decide which results to show. It significantly speeds up simulations as it does not need to constantly update the visuals of the grid. At the end, a large dataframe is provided which contains all requested results.

5.2. Remarks/recommendations

Multiple targets A* Newly assigned AGVs during next-best task assignment should have their path coordinated w.r.t. the new task. This happens when the newly AGV is not transporting a bag. However, when an AGV is still delivering its bag and already gets a new one assigned, the new root node does not use the path to this new target but keeps the original target of the AGV since a way to add multiple targets to a path has not been implemented. This problem was found when AGVs were not delivering their bags anymore since their new path did not include their current target. A next iteration of the model should therefore adapt the A* algorithm so that AGVs can plan a path to another destination while making sure their current goal lies on that path.

Modifying IDMB + ECBS-TA implementation Our model allows for relatively easy modification of the IDMB & ECBS-TA combination as several elements can simply be excluded while not affecting others. Examples are task swapping in IDMB or task swapping in assignment nodes ECBS-TA. Although the task assignment from IDMB provides less complex scenarios for ECBS to solve, it requires a significant amount of time to let AGVs communicate with each other. Testing whether not invoking different aspects of IDMB and ECBS-TA could lower runtimes, is something to look into in the future. On the other hand, other distributed task allocation methods could be tested.

Alternative layouts The layouts in which ECBS-TA had been tested made use of Kiva-like structures which limited the freedom of agents but which spread the origins and destinations throughout the environment. In our case, our origin was located centrally which lead to significant traffic density at this position and leading to difficulties for ECBS. Further research should therefore test our distributed MAPD algorithm in warehousing environments to see whether the layout can be something our algorithm (or ECBS-TA) could benefit from.

Battery usage optimization Optimizing the charging process of the AGVs was not within the scope of this project. This implies that we did not guarantee that battery levels would not drop below zero. It was however into account to a certain extent when during the task assignment. This did check whether AGVs had enough energy to, after delivering a bag, make it back in time for charging. But, this check was not performed during task swapping which therefore could cause longer travel times than battery levels allowed.

Furthermore, a next iteration could consider to, at times, not let AGVs charge fully when it would benefit bag waiting times.

Buffer spot allocation In our model, buffer spots were assigned randomly. The buffer spot allocation could however be improved to assign a spot close to the infeeds, when the AGV is available for a task, or close to a charging spot that will become available.

[2] https://github.com/projectmesa/mesa, https://github.com/seifely/mesa_boxworld

Bibliography

[1] C. Malandri, M. Briccoli, L. Mantecchini, and F. Paganelli. A Discrete Event Simulation Model for Inbound Baggage Handling. *Transportation Research Procedia*, 35:295–304, 2018.

[2] Vanderlande. CASE STUDY: Rotterdam The Hague Airport and Vanderlande are ideal innovation partners. Retrieved in December 2019, 2019.

[3] A. Sharpanskykh. *ASSIGNMENT 2: Multiagent planning, scheduling and coordination in a distributed airport baggage logistics system.* Delft University of Technology, 2018.

[4] V.J. Olijslager. Multi-Agent System application to a high-capacity Decentralized Bagagge Handling System, A Literature Review. Master's thesis, Delft University of Technology, April 2018. Accessed in December 2019.

[5] P.R. Wurman, R. D'Andrea, and M. Mountz. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1):9–20, 2008. ISSN 0738-4602.

[6] P. Baran. *On Distributed Communications: 1. Introduction to distributed communication networks.* RAND Corporation, August 1964. Memorandum RM-3420-PR.

[7] U. Wilensky. NetLogo Flocking model, 1998. URL http://ccl.northwestern.edu/netlogo/models/Flocking. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

[8] A. Sharpanskykh. Lecture: Agent-based Modelling and Simulation in Air Transport. Presentation, September 2019. Delft University of Technology.

[9] A. Khamis, A. Hussein, and A. Elmogy. *Cooperative Robots and Sensor Networks 2015*, chapter Multi-robot Task Allocation: A Review of the State-of-the-Art, pages 31–51. Springer International Publishing, 2015.

[10] H. Ma and S. Koenig. AI Buzzwords Explained: Multi-Agent Path Finding (MAPF). *AI Matters*, 3(3):15–19, 2017.

[11] H. Ma and S. Koenig. Optimal Target Assignment and Path Finding for Teams of Agents. In J. Thangarajah, K. Tuyls, C. Jonker, and S. Marsella, editors, *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, 2016.

[12] W. Hönig, S. Kiesel, A. Tinka, J.W. Durham, and N. Ayanian. Conflict-Based Search with Optimal Task Assignment. In M. Dastani, G. Sukthankar, E. André, and S. Koenig, editors, *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018)*, 2018.

[13] M. Cáp, J. Vokrínek, and A. Kleiner. Complete Decentralized Method for On-Line Multi-Robot Trajectory Planning in Well-formed Infrastructures. *Association for the Advancement of Artificial Intelligence*, 2015.

[14] M. Liu, H. Ma, J. Li, and S. Koenig. Task and Path Planning for Multi-Agent Pickup and Delivery. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019)*, 2019.

[15] Y. Varyani. Hungarian Algorithm for Assignment Problem, Set 1 (Introduction), January 2016. URL https://www.geeksforgeeks.org/hungarian-algorithm-assignment-problem-set-1-introduction/. Accessed in February 2020.

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

[17] N.R. Sturtevant. Benchmarks for Grid-Based Pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, June 2012.

[18] S. Singh and R. Niyogi. DiMPP: a complete distributed algorithm for multi-agent path planning. *Journal of Experimentsl & Theoretical Artificial Intelligence*, 29(6):1129–1148, April 2017. DOI: 10.1080/0952813X.2017.1310142.

[19] G. Sharon, R. Stern, A. Felner, and N. Sturtevant. Meta-Agent Conflict-Based Search For Optimal Multi-Agent Path Finding. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search*, pages 97–104, 2012.

[20] L. Cohen, T. Uras, and S. Koenig. Feasibility Study: Using Highways for Bounded-Suboptimal Multi-Agent Path Finding. In *Conference: Eigth Annual Symposium on Combinatorial Search*. Association for the Advancement of Artificial Intelligence, January 2015.

[21] L. Cohen, T. Uras, T.K.S. Kumar, H. Xu, N. Ayanian, and S. Koenig. Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding. In *IJCAI*, 2016.

[22] R. Claes, T. Holvoet, and D. Weyns. A Decentralized Approach for Anticipatory Vehicle Routing Using Delegat Multiagent Systems. *IEEE Transactions on intelligent transportation systems*, 12(2):364–373, June 2011. ISSN 1524-9050.

[23] H.T. Dinh, R. van Lon, and T. Holvoet. Multi-agent route planning using delegate MAS. In *Workshop on Distributed and Multi-Agent Planning*, pages 24–32, 2016.

[24] Viva Differences. Difference Between DFS (Depth First Search) And BFS (Breadth First Search) In Artificial Intelligence, October 2019. URL https://vivadifferences.com/difference-between-dfs-and-bfs-in-artificial-intelligence/. Accessed in January 2020.

[25] S. Singh and R. Niyogi. DMAPP: A Distributed Multi-Agent Path Planning Algorithm. In *28th Australasian Joint Conference on Artificial Intelligence 2015*, volume 9457, 2015.

[26] E. Frazzoli. Principles of Autonomy and Decision Making. Lecture 15: Sampling-Based Algorithms for Motion Planning. Presentation, November 2010. URL https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-410-principles-of-autonomy-and-decision-making-fall-2010/lecture-notes/MIT16_410F10_lec15.pdf. Massachusetts Institute of Technology.

[27] M. Cáp, P. Novák, J. Vokrínek, and M. Pechoucek. Multi-agent RRT*: Sampling-based Cooperative Pathfinding. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, 2013.

[28] A. Felner, M. Goldernberg, G. Sharon, R. Stern, T. Beja, N. Sturtevant, J. Schaeffer, and R. Holte. Partial-Expansion A* with Selective Node Generation. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence*, pages 471–477, 2012.

[29] Vanderlande. FLEET, 2019. URL https://www.vanderlande.com/airports/evolutions/fleet/. Accessed in December 2019.

[30] V. Olijslager. Operationalizing a High-capacity Decentralized Baggage Handling system. Master's thesis, Delft University of Technology, November 2018.

[31] L. Qiu, W. Hsu, S. Huang, and H. Wang. Scheduling and routing algorithms for AGVs: A survey. *International Journal of Production Research*, 40(3):745–760, 2002.

[32] A. Gittens. ACIs World Airport Traffic Forecasts reveal the drivers of air transport demand growth on the path to 2040, October 2019. URL https://www.internationalairportreview.com/article/106229/world-airport-traffic-forecasts-aci-world/. Accessed in March 2020.

[33] E. Kersten. Interview by Circulaire Maakindustrie: Vanderlande - FLEET. https://circulairemaakindustrie.nl/case/vanderlande-fleet/, November 2019. URL https://circulairemaakindustrie.nl/case/vanderlande-fleet/.

[34] J.P. Cavada, C.E. Cortés, and P.A. Rey. A simulation approach to modelling baggage handling systems at an international airport. *Simulation Moddeling Practice and Theory*, 75:146–164, June 2017.

[35] Glidepath. Design of Baggage Handling Systems. Presentation, 2017. Accessed in 2019.

[36] Vanderlande. Vanderlande, 2019. URL https://www.vanderlande.com/. Accessed in December 2019.

[37] Vanderlande. BAGFLOW, 2019. URL https://www.vanderlande.com/airports/evolutions/bagflow/. Accessed on 12/2019.

[38] F. Klügl and A. Bazzan. Agent-Based Modeling and Simulation. *AI Magazine*, Fall:29–40, 2012. ISSN 0738-4602.

[39] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, ltd, 2002.

[40] G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Massachusetts Institute of Technology, 1999. ISBN 0-262-23203-0.

[41] Z. Yan, N. Jouandeau, and A.A. Cherif. A Survey and Analysis of Multi-Robot Coordination. *International Journal of Advanced Robotic Systems*, 10(399), 2013.

[42] Z. Li, C.H. Sim, and M.Y.H. Low. A Survey of Emergent Behavior and Its Impacts in Agent-based Systems. In *International Conference on Industrial Informatics*, 2006.

[43] A. Sharpanskykh. Lecture: Specification of models of multiagent systems. Presentation, September 2019. Delft University of Technology.

[44] W.A. Luna-Ramirez and M. Fasli. Bridging the Gap between ABM and MAS: A Disaster-Rescue Simulation Using Jason and NetLogo. *Computers*, 7(24), 2018.

[45] Drexel University. Beliefs, Desires, Intentions (BDI), September 2006. URL https://www.cs.drexel.edu/~greenie/cs510/bdilogic.pdf. Accessed in January 2020.

[46] A.S. Rao and M.P. Georgeff. BDI Agents: From Theory to Practice. In *Proceedings of the First International Conference on Multiagent Systems*, 1995.

[47] A. Sharpanskykh. Lecture: Multiagent planning and scheduling. Presentation, October 2019. Delft University of Technology.

[48] M. de Weerdt and B. Clement. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, December 2009.

[49] D. Borrajo and S. Fernández. Efficient approaches for multi-agent planning. Paper, Universidad Carlos III de Madrid, Av. Universidad 30, Leganés, Madrid, Spain, May 2018.

[50] F. Zitouni, R. Maamri, and S. Harous. FA-QABC-MRTA: a solution for solving the multi-robot task allocation problem. *Intelligent Service Robotics*, 12:407–418, 2019.

[51] B.P. Gerkey and M.J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *Int. J. of Robotics Research*, 23(9):939–954, September 2004.

[52] C. Tovey, M.G. Lagoudakis, S. Jain, and S. Koenig. The Generation of Bidding Rules for Auction-Based Robot Coordination. In L.E. Parker, F.E. Schneider, and A.C. Schultz, editors, *Multi-Robot Systems. From Swarms to Intelligent Automate*, volume 3, pages 3–14. Springer, 2005.

[53] W. Yijuan, P. Weijun, and L. Kaiyuan. Multi-Agent Aviation Search Task Allocation Method. In *IOP Conf. Series: Materials Science and Engineering 646*, 2019.

[54] A. Geramifard, P. Chubak, and V. Bulitko. Biased Cost Pathfinding. Technical report, Department of Computing Science, University of Alberta, 2006.

[55] M. Cirillo, F. Pecora, H. Andreasson, T. Uras, and S. Koenig. Integrated Motion Planning and Coordination for Industrial Vehicles. In *ICAPS'14: Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, page 4, June 2014.

[56] H. Ma, J. Li, T.K.S. Kumar, and S. Koenig. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In S. Das, E. Durfee, K. Larson, and M. Winikoff, editors, *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)*, 2017.

[57] G. Sharon, R. Stern, A. Felner, and N. Sturtevant. Conflict-Based Search For Optimal Multi-Agent Path Finding. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 563–569, 2012.

[58] C. Chegireddy and H. Hamacher. Algorithms for finding K-best perfect matchings. *Discrete Applied Mathematics*, 18:155–165, 1987.

[59] J. Yu and S. LaValle. Structure and Intractabiltiy of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, pages 1443–1449, 2013.

[60] D. Silver. Cooperative Pathfinding. *American Association for Artificial Intelligence*, 2005.

[61] H. Ma, W. Hönig, T.K.S. Kumar, N. Ayanian, and S. Koenig. Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery. *Association for the Advancement of Artificial Intelligence*, 2019.

[62] E. Osaba, P.L. Garcia, E. Onieva, R. Carballedo, F. Diaz, and A. Perallos. A Parallel Meta-Heuristic for Solving a Multiple Asymmetric Traveling Salesman Problem with Simultaneous Pickup and Delivery modeling Demand Responsive Transport Problems. In *Hybrid Artificial Intelligent Systems*, 2015.

[63] S. Yoon and J. Kim. Efficient multi-agent task allocation for collaborative route planning with multiple unmanned vehicles. *IFAC PapersOnLine*, 50(1):3580–3585, July 2017. ISSN 2405-8963.

[64] K. Helsgaun. An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems. Technical report, Roskilde University, 2017.

[65] E. Weisstein. Hamiltonian Cycle, 2003. URL http://mathworld.wolfram.com/HamiltonianCycle.html. From MathWorld–A Wolfram Web Resource. Accessed in January 2020.

[66] J.P. van den Berg and M.H. Overmars. Prioritized Motion Planning for Multiple Robots. In *Intelligent Robots and Systems*, 2005.

[67] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and E. Shimony. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 740–746, 2015.

[68] B. Gerkey and M. Mataric. Sold!: auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, October 2002.

[69] W. yao, Y. Liu, and N. Wan. An Iterative Strategy for Task Assignment and Path Planning of Distributed Multiple Unmanned Aerial Vehicles. *Aerospace Science and Technology*, February 2019.

[70] H. Choi, L. Brunet, and J. How. Consensus-Based Decentralized Auctions for Robust Task Allocation. *IEEE Transactions on Robotics*, 25(4):912–926, August 2009.

[71] S. Trigui, A. Koubaa, O. Cheikhrouhou, H. Youssef, H. Bennaceur, M. Sriti, and Y. Javed. A Distributed Market-based Algorithm for the Multi-robot Assignment Problem. *Procedia Computer Science*, 32:1108–1114, December 2014.

[72] G. Sharon, R. Stern, M. Goldenberg, and Felner A. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195:470–495, February 2013.

[73] P. Hart, N. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems and Cybernetics*, 4(2):100–107, July 1968.

[74] T. Standley. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 173–178, 2010.

[75] E. Boyarski, A. Felner, G. Sharon, and R. Stern. Don't Split, Try To Work It Out: Bypassing Conflicts in Multi-Agent Pathfinding. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling*, pages 47–51, 2015.

[76] M. Barer, G. Sharon, R. Stern, and A. Felner. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *Proceedings of the Seveth Annual Symposium on Combinatorial Search (SoCS 2014)*, pages 19–27, 2014.

[77] D. Weyns, T. Holvoet, and A. Helleboogh. Anticipatory Vehicle Routing using Delegate Multi-Agent Systems. In *2007 IEEE Intelligent Transportation Systems Conference*, 2007.

[78] A. ter Mors, J. van Belle, and C. Witteveen. Context-Aware Multi-Stage Routing. In *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, volume 1, pages 49–56. Delft University of Technology, May 2009.

[79] C. Wang and A. Botea. MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *Journal of Artificial Intelligence Research 42*, pages 55–90, November 2011.

[80] T. Standley. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 173–178, 2010.

[81] J. Hoffmann and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research 14*, January 2001.

[82] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996.

[83] M. Erdmann and T. Lozano-Pérez. On Multiple Moving Objects. *Algorithmica*, 2:477–521, 1987.

[84] W. Yu, J. Peng, and X. Zhang. A Prioritized Path Planning Algorithm for MMRS. In *Proceedings of the 33rd Chinese Control Conference*, pages 966–971, July 2014.

[85] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[86] T. Yoshizumi, T. Miura, and T. Ishida. A* with Partial Expansion for Large Branching Factor Problems. In *Proceedings of the Seventeenth national conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 923–929, July 2000.

[87] W. Duchateau and S. Verhellen. Assignment 2: Multiagent Planning, Scheduling and Coordination in a Distributed Airport Baggage Logistics System. Agent-Based Modelling and Simulation in Air Transport, 2018, May 2018.