Designing a grid-based routing strategy for an AGV system

James, Sem J.

Award date:
2021

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct[i].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

06-05-2021

Name

S.J. (Sem) James

ID-number

0893273

Signature

[illegible]

Submit the signed declaration to the student administration of your department.

[i] See: https://www.tue.nl/en/our-university/about-the-university/organization/integrity/scientific-integrity/
The Netherlands Code of Conduct for Scientific Integrity, endorsed by 6 umbrella organizations, including the VSNU, can be found here also. More information about scientific integrity is published on the websites of TU/e and VSNU

Graduation project report

Designing a grid-based routing strategy for an AGV system

Graduation Project - Vanderlande Industries

S.J. (Sem) James - 0893273 - CST2021.018

Manufacturing Systems Engineering
Control Systems Technology

Department of Mechanical Engineering

Supervisors:

Supervisor Vanderlande:
ir. K.J.C. Fransen

Supervisor TU/e:
dr. ir. J.A.W.M. van Eekelen

Mentor TU/e:
dr. ir. M.A. Reniers

Eindhoven, Wednesday 5th May, 2021

Preface

With this graduation report I am at the point at which my study period at the Eindhoven University of Technology is almost over. Seven years ago I started my Bachelor in Mechanical Engineering. Three years later and with only the bachelor final project to go I decided to do a full-time board year at W.S.V. Simon Stevin. Half a year after this board year I finished my Bachelor degree. I decided to start the Master program Manufacturing Systems Engineering within the Control Systems Technology group. Two and a half years later the time is here: my graduation is just around the corner.

In September last year my internship in Australia would take place. Unfortunately due to the corona virus measures I was not able to go there and do my internship. Luckily, Vanderlande Industries gave me the opportunity to conduct a research within their company, which I liked and for which I am very grateful. A big thanks to my Vanderlande supervisor Karlijn Fransen, to my other Vanderlande (and TU/e) supervisor Joost van Eekelen and to my TU/e mentor Michel Reniers for their supervision throughout the project and help throughout the study. I would also like to thank my fellow interns at Vanderlande and all other colleagues I worked with for the great time I had here, despite working from home during almost the whole duration of the graduation internship.

Last but not least I want to thank my family and friends for their support (and sometimes distraction) during my studies. Members of Rhetoricadispuut Tau, board members of the *'Vigorous'* 61st Board, my former roommates at the *'Hamampaleis'*; thanks a lot.

Sem James
May 2021

Abstract

The subject of this graduation project is the design of a grid-based routing strategy for an AGV system. In this strategy path planning (planning routes for AGVs) and traffic control (avoiding collision and deadlocks) are combined into one approach and the routes (and where an AGV must be at what point in time) is determined before driving. Grid-based in this context means that a layout (for example a factory floor) is divided in grid tiles. *AgvSorter*, a Vanderlande in-house Matlab simulation model is used to test performance indicators like mean throughput, the mean number of jobs handled by the system per unit time. The main goals of this research are to design a routing strategy, implement it in the *AgvSorter* model and perform simulations to compare this method with the existing separate path planning and traffic control strategy.

In literature there are several options to combine path planning and traffic control into one approach. After evaluating some of these options the context-aware routing strategy as described by Ter Mors, Zutt and Witteveen [1] seems to be the most suited for the given system. In this approach routes for AGVs are planned through the reachable free time windows of tiles in a grid layout, in the ideal world resulting in deadlock-free routes. By doing so the routes are planned given all the other planned routes so far, meaning that the controller is aware of the context or state of the system. For this project an extension of this context-aware routing algorithm was used.

For the context-aware routing approach costs are introduced to estimate how long time window reservations should be per tile in a route. For example there are costs for moving, turning, (un)loading, accelerating and decelerating. A heuristic cost function is present to make sure the route planning is performed faster. Since vehicles that have no job might block other AGVs from driving a certain route, parking spots were introduced. A route for an AGV is always planned from start location to source to destination to parking spot.

The discussed routing approach works for determining routes for the AGVs. However, in execution there are disturbances in the system and therefore the planned routes do not exactly match reality. To make sure the planning remains feasible and executable, a push-pull mechanism is used. In case a reservation is delayed, all later reservations which interact with this reservation are pushed forward and in case a reservation is finished earlier, all later reservations which interact with this reservation are pulled forward. In the push-pull model, max-plus algebra is used to define the relations between the reservations.

To test the context-aware routing algorithm in combination with the push-pull model, simulations were performed using the *AgvSorter* simulation model. Three different layouts have been tested, in which one layout was tested for both unidirectional lanes as bidirectional lanes. In these simulations three key performance indicators were measured. The first one is the mean throughput, the mean number of jobs handled by the system per hour. The second one is the mean occupancy: the percentage of time an AGV is loaded. The third and last performance indicator is the mean item lead time which is the mean time that AGVs needed from pick-up to drop-off.

The simulation results were compared to the situation in which the existing separate path planning and traffic control (classic grid-based control or classic GBC) was used. In all cases (except the bidirectional situation) the throughput was lower and the mean item lead time was higher meaning that the context-aware routing strategy at this point performs worse than the original controls. There are some reasons for this which are discussed in the report. It is recommended to perform some optimizations to upgrade the performance of the context-aware control model. Also it is recommended to look more at bidirectional routing since the first results for this are promising.

List of abbreviations

Abbreviation	Definition
AGV	Automated Guided Vehicle
BHS	Baggage Handling System
CA*	Context-Aware A*
CARP	Context-Aware Route Planning
DAG	Directed Acyclic Graph
FCFS	First Come First Serve
FMS	Fleet Management System
GBC	Grid-based control
HCA*	Heuristic Cooperative A*
KPI	Key Performance Indicator
OTC	Online Traffic Controller
RTG	Routing Table Generator
SBC	Segment-based control
ULD	Unit Load Device
WHCA*	Windowed Heuristic Cooperative A*

Contents

1. Introduction

Vanderlande, part of the Toyota Industries Corporation, is a company that delivers innovative value-added automation solutions in the world of airport and parcel logistics. Next to this it focuses on process automation solutions in warehousing. The systems of Vanderlande are used at 600 airports (of which 12 among the 20 biggest in the world), moving over 11.5 million pieces of luggage a day. Using Vanderlande systems, over 48 million parcels are sorted every day [2].

Today, most of the solutions involve large systems using conveyor belts. These are big, fixed structures that can deliver high throughputs. One of the main disadvantages is the fact that these systems are not flexible. Maintenance or expansion causes long downtimes. If there is a failure in the main loop in the system system, the rest of the system does not work anymore. For this reason most traditional conveyor-systems have redundancy lines which cause extra costs. For these reasons Vanderlande is investigating and developing more flexible methods involving Automated Guided Vehicle (AGV) systems.

Currently there is FLEET, part of Vanderlande's end-to-end baggage logistics solution [3]. Using this system, with a number of AGVs the baggage handling system that used to work with roller- and belt- conveyors can be replaced. This results in a higher level of robustness regarding infrastructural and environmental changes, causing a more flexible and scalable way of handling baggage. Other types of AGVs that are developed include autonomous shuttles [4] and even forklifts.

Creating AGV solutions for the mentioned problems pose some interesting problems in terms of control. The main problem here is path planning and traffic control of the vehicles. The current FLEET system uses a segment-based control approach, in which different route parts (segments) are defined [3]. In recent years another method is proposed: grid-based control [5]. This report focuses on this grid-based control.

A system in which grid-based control can be used easily is an AGV Sorter system. As the name indicates, with this system it is possible to sort for example parcels or baggage. Within Vanderlande an in-house analysis model, called *AgvSorter*, is present in which all kind of simulations can be run. In the context of this project, this *AgvSorter* model will be used.

1.1 Thesis objectives

In recent years a lot of research is performed in the field of separate grid-based path planning and traffic control [6] and deadlock-avoidance strategies needed in this approach [7]. In literature there are also options to combine path planning and traffic control while avoiding deadlocks. The main objective of this research project is to come up with a new routing strategy in which this grid-based path planning and traffic control (and thus deadlock-avoidance) is combined using a time-window based approach. The different approaches can then be compared to see which method performs better. The main research question of this project is:

How can time-window based routing be used to combine path planning and traffic control in the AgvSorter system?

To answer the main research question, some objectives were defined:

1. Investigate different methods from literature that combine path planning and traffic control and make a selection of those.
2. Design an algorithm that routes AGVs based on available time-windows.
3. Find a way to deal with idle vehicles and disturbances efficiently.
4. Implement the proposed routing approach in the *AgvSorter* simulation model in Matlab.
5. Compare the time-window based routing approach to the currently implemented separate path planning and traffic control method.

1.2 Thesis outline

To answer the main research question and cover the mentioned objectives, this report consists of some chapters. The contents of these chapters are discussed briefly below.

Chapter 2: AGV systems - A detailed introduction to AGV systems within Vanderlande, the control of AGVs and the segment-based and grid-based control strategies.

Chapter 3: Combined path planning and traffic control - An overview of three different methods from literature to combine path planning and traffic control. Also a comparison between these methods and a conclusion on which method is the best in the *AgvSorter* case is presented.

Chapter 4: Context-aware route planning algorithm - The algorithm used for designing and implementing a time-window based routing approach and how to deal with idle vehicles is discussed.

Chapter 5: Push-pull planning - A method to deal with delays and disturbances in the system is presented. First max-plus algebra, which is used in the push-pull planning, is discussed.

Chapter 6: Simulations - First some practicalities regarding the implementation of the proposed routing strategy in the *AgvSorter* system are discussed, after which the simulation results and result interpretation are presented.

Chapter 7: Conclusion and recommendations - A conclusion regarding the main research question and objectives is drawn. Some recommendations for future work are also presented.

2. AGV systems

Vanderlande is researching the use of AGVs to tackle the scalability and flexibility problems that come with the use of conventional conveyor systems. This chapter first introduces the application domain of AGVs within the company within the three market segments Vanderlande distinguishes: airports, warehousing and parcel. Next, some introductory information on AGV control is presented, after which two control concepts (segment-based and grid-based) and the software models for simulating these concepts within Vanderlande are discussed.

2.1 AGV application domain

Within the market segment airports [8], the use of AGVs is in the individual baggage handling. This means the transport of baggage from and to security checks, intermediate storage and eventually to the right location to board the plane. AGVs can also be used for moving unit load devices (ULDs). These are large containers that can hold multiple pieces of baggage. Since the arrival of baggage is not constant over the day, flexible AGV systems can help here.

In the market segment warehousing [9], there are also multiple applications for AGVs. Here one can think of automated pallet trucks, shuttle systems that move individual products and AGVs that pick up and drop off products or pallets. AGVs can help to move products to their storage location and back.

Similar to the warehousing market segment, in the segment parcel [10] AGV systems can also be used for transporting parcels within a facility. Also, automated forklift might be used here. An interesting other application is the sorting of parcels as already happens within a Chinese express company [11]. Here, the packages are picked up at some location and dropped off at another location.

In the past few paragraphs some of the application domains of AGVs that Vanderlande sees within their market segments are shown. Some of the mentioned applications are already running. For the movement of for example baggage at airports, the FLEET baggage system was developed as shown in Figure 2.1a. A shuttle system (not an AGV system but similar controls) that can be used in for example warehouses called ADAPTO is shown in Figure 2.1b. The FLEET baggage system currently uses the so-called segment-based control which is discussed in Section 2.3.

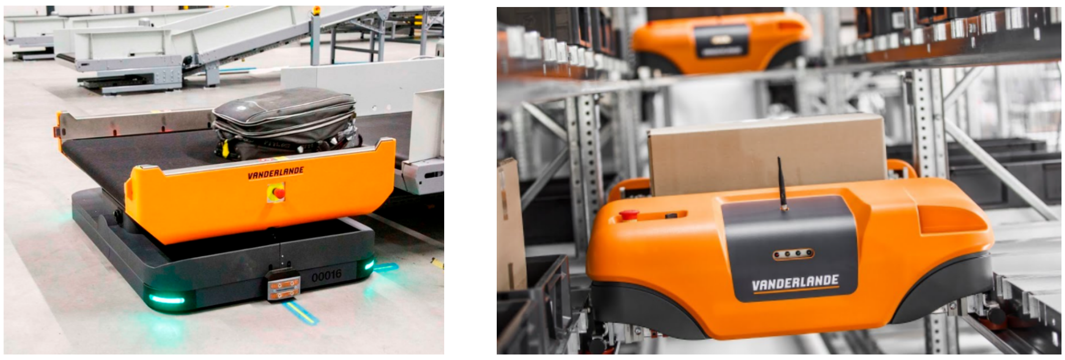

(a) FLEET baggage system [3]. (b) ADAPTO shuttle system [4].

Figure 2.1: Examples of AGV(-like) systems within Vanderlande.

2.2 Control of AGVs

For every AGV application mentioned in this chapter, similar control mechanisms can be used. At first there must be a layout over which AGVs can move. A layout is a map of for example (part of) a factory floor which can be divided in drivable areas, obstacles, pick-up points, drop-off points and so on. The control of AGVs in a certain layout can be zone-based or free. In the case of zone-based control the total layout (or part of it) is subdivided in a number of zones.

Zones can represent tiles (further introduced in Section 2.4) which together form the layout or zones represent path segments that are placed on a layout (further introduced in Section 2.3). The number of zones depends on the size and shape of the layout. The movement between the different zones is fixed and restricted, meaning that an AGV can only move between two zones if there is a path between them. In free control, AGVs can drive everywhere and in any direction (keeping the boundaries of the factory floor in sight).

In the scope of this project zone-based layouts are used and it is assumed that all path segments are unidirectional (unless mentioned differently), meaning that an AGV can only drive over it in one direction. This assumption makes the control of the systems significantly less complex. The advantage of zone-based control over free control is the fact that collisions are avoided easily by making sure only one AGV is present at a zone (or multiple zones) at the same time. In Section 2.3 and 2.4 two zone-based control approaches are describes (segment-based and grid-based respectively). Also the software models used for these approaches within Vanderlande are presented there.

The control of an AGV system can be performed offline or online. An online approach is a situation in which jobs arrive real-time, meaning that not all jobs (which include a pick-up and drop-off location combination) are known before the execution. In an offline approach all jobs are known prior to the execution. For this project an online approach is used: jobs arrive and paths must be planned and executed. Also disturbances must be taken into account in an online way.

2.2.1 Path planning

In the scope of zone-based AGV control, path planning is the activity of a controller in which a path for an AGV (for example from pick-up location to drop-off location) is determined. A path is the route an AGV traverses through the layout (so a combination of zones). There are many path planning approaches, of which some are described in Chapter 3. In this subsection some definitions on path planning are discussed.

In static path planning the path of an AGV is set before execution of the path. This means that an optimal path is calculated, after which the execution starts. In this case the path is fixed. It is possible that an AGV must wait for a long time, since the timing of the paths of other AGVs is not taken into account. In dynamic path planning only the next one or next few zones are planned ahead, which allows for continuous adaption to the new traffic situation. In some cases up to ten zones are planned ahead. After traversing two zones, the next ten zones are planned again, this is called receding horizon control. In this project static path planning is used.

Next to static or dynamic, path planning approaches can be centralized or distributed. When a centralized approach is used, one controller determines the paths of all AGVs in the system. This means that it can base its path planning on all available information in the system, including all paths of other AGVs. In the distributed path planning approach each individual AGV (or a set of some AGVs) has a separate controller. The information that the controller can use is local, meaning that it only uses information of the AGVs close to the considered AGV. Centralized approaches can result in optimal solutions, whereas decentralized distributions are less complex. In this project a centralized approach is considered.

2.2.2 Traffic control

Next to a path planning strategy to find a path through the zones in a layout, a collision and deadlock avoidance approach must be present: a traffic control approach. Deadlocks are situations in which an AGV is waiting for another AGV, which is again prohibited from moving because it is waiting for another AGV and so on. When separate path planning and traffic control is used a deadlock avoidance strategy and deadlock detection approach [7] must be present.

Whereas path planning can be an action prior to movement, traffic control is about execution of the planned path. It tells every AGV whether it can proceed along its path in present time. In this project the aim is to combine path planning and traffic control, making deadlocks less likely to occur if no disturbances are present.

2.3 Segment-based control

In segment-based control the layout of for example a factory floor is a segment-based layout. Fixed line segments over which AGVs can drive are present on the total floor. These segments are chosen in such a way that every AGV can reach all source and destination zones. Within Vanderlande, segments can be placed freely on the map. Also, segments may overlap, and can have different sizes, divert points etc. Segments must be connected to other segments at start and end points [12]. Figure 2.2 shows an example of an application of a segment-based layout. All the gray paths consist of segments over which driving in a certain direction is allowed. Note that the paths can divert, merge, contain turns etc.

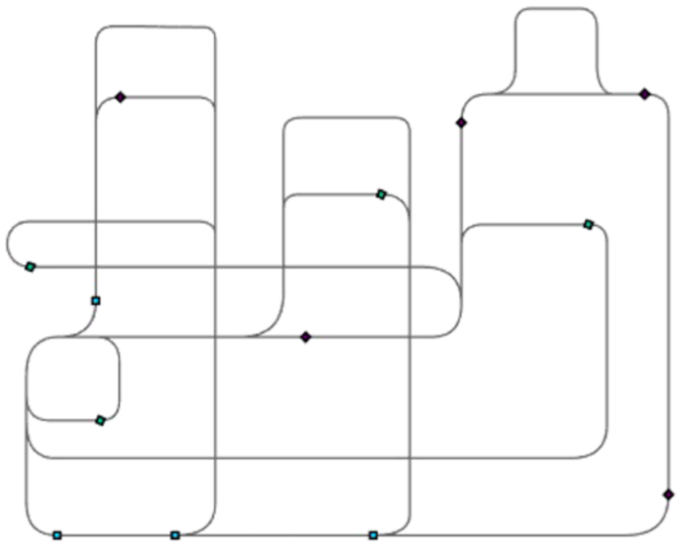


Figure 2.2: Example of a possible segment-based layout in an airport environment [12] [13].

2.3.1 Fleet management system

The Fleet Management System or FMS is the currently implemented segment-based software for controlling the FLEET AGV system [3]. This software consists of job assignment, path planning, traffic control and a charging strategy. This project does not focus on the FMS but on the *AgvSorter* system, but some ideas that are used might be interesting and useful. The charging strategy is not of interest for this project and therefore not explained.

The segments in the system have a size of approximately ten centimeters, meaning that multiple segments are occupied by a single AGV at the same time. The controller makes sure straight and divergent segments around locations where AGVs are present are blocked for other AGVs, making sure that collisions do not occur.

Jobs are currently planned using a smart job assignment strategy [12]. This job assignment strategy uses a weighted sum of driving time and job age to assign jobs to the different AGVs. When there is no job for an AGV it gets the assignment to go to a parking spot in order to not stand in the way of other AGVs.

Path planning in the FMS is performed using time windows. A path in this case is a sequence of segments. Every segment has a set of free time windows that can be reserved when planning the path of an AGV. The optimal path through these time windows is calculated using an A* algorithm [14]. Periodically the actual situation on the map is evaluated. If a vehicle is delayed, the whole planning is delayed in order to prevent overlapping time windows. The order of vehicles entering a certain segment is always the same as in the original planning.

Traffic control is performed in two ways. Since the path planning uses time windows, in theory traffic control is not needed since deadlocks are avoided and collisions do not occur. In practice, AGVs do not behave exactly as modeled. This has to do with assumptions in the model: the movement of an AGV is modeled in a simple way to avoid complexity (no acceleration and deceleration modeled for example). Furthermore, changes in the environment of the AGVs influence their behaviour (for example temperature, uneven floor etc.). Due to the fact that AGVs do not behave exactly as modeled, every 0.5 seconds an update (mostly a delay in time windows) is applied. In combination with a sensor for emergency cases this fixes the traffic control problem.

2.4 Grid-based control

Grid-based control uses a layout in which a factory floor or baggage handling hall is subdivided in zones or tiles [5]. An AGV can only travel in the directions that are defined in the grid. These directions are called edges: connections between tiles over which AGVs can drive. Also there are tiles that cannot be driven (e.g. obstacles) and there are tiles for pick-up, drop-off and charging actions. The system that Vanderlande uses, the *AgvSorter* system [5] is described in more detail in Section 2.4.1. There, also an application of a grid-based layout is shown.

Within grid-based control some important definitions must be clear. The paths between different tiles are called edges. AGVs can only move from one tile to another when an edge is present. In graphical representations, all nodes (center of a drivable tile) are represented by vertices. The mentioned edges connect these vertices. Weights are assigned to each vertex or edge and can correspond to the length of an edge or the cost (e.g. duration) for driving over this edge or vertex.

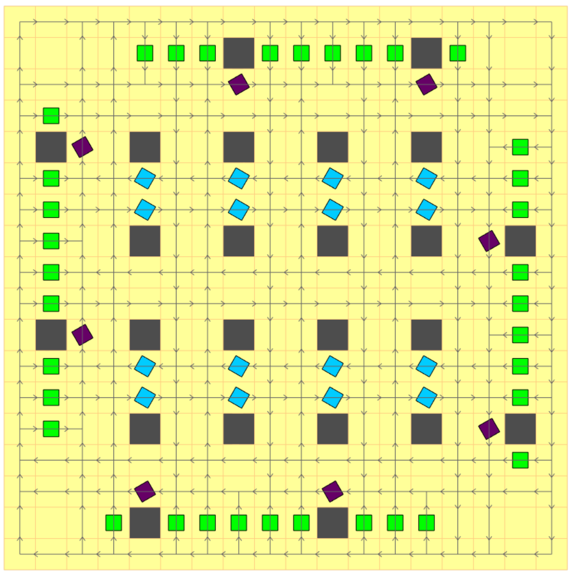


Figure 2.3: Example of a grid-based layout [5].

2.4.1 *AgvSorter* system

In this subsection the *AgvSorter* system, which is a grid-based AGV control software system, and its simulation model within Vanderlande are explained. In the last section the desired additions with respect to this system are discussed. Since the *AgvSorter* system is used in this project, the information on this topic is somewhat more extensive than in the FMS case.

System overview
As the name *AgvSorter* indicates, it is about an automated guided vehicle that is able to sort all kind of stuff like baggage and parcels. In airports it can be used for moving baggage from check-in to security check to the correct conveyor and in warehouses it can be used for sorting packages based on for example delivery address. An example is shown in Figure 2.4.

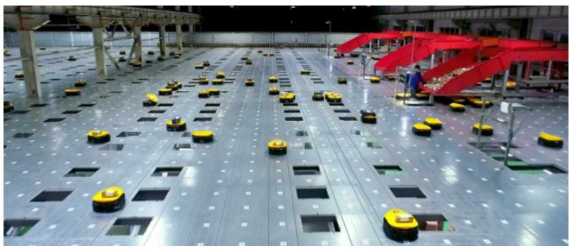


Figure 2.4: Example of a package sorting application in a grid-based environment [15].

In the *AgvSorter* a grid layout is used. AGVs can only move in directions that are indicated in the grid grid layout. When an AGV gets a job it must drive to a pick-up location where the AGV is loaded, and must then drive to a drop-off location to get unloaded. Movement is only possible over so-called edges between tiles and grid layouts can consist of obstacles, pick-up tiles, drop-off tiles (and charging and parking tiles if present).

In the *AgvSorter* model the tiles, previously called zones, are connected by drivable edges. These edges represent a path between two tiles. All paths are assumed to be unidirectional, meaning that the paths can only be driven in one direction. Bidirectional paths might be interesting in the future (and are shortly discussed in Chapter 6) but for simplicity the assumption of unidirectional paths was made in the original *AgvSorter* system.

The control of the *AgvSorter* system is based on some ground principles [5]. These are that AGVs move from tile to tile, tiles do not overlap, tiles are big enough for an AGV to make a turn in it without entering other tiles, every tile can only be reserved by one AGV and an AGV cannot cross any other tile than the current and next reserved tile.

The occupation of a tile is arranged as follows. Since the grid size is determined based on the size of the AGVs, only one AGV can be present at one tile at a time. From this it follows that a tile can only be claimed by one AGV at the same time. If multiple AGVs could claim the same tile at the same time, collisions might occur. Once the next planned tile for an AGV is claimed, this AGV is allowed to traverse it. Once an AGV has left the previous tile completely, this tile is released and thus open for new claim actions.

For each AGV multiple pick-up and drop-off locations are possible. This might be the case due to the fact that multiple drop-off locations end on the same conveyor in the end. It is also possible that there are multiple similar workstations, or similar baggage check systems. For this reason in the simulation model [5] locations might be grouped. It is also possible that AGVs can carry multiple jobs at the same time. The pick-up and drop-off location do not necessarily overlap.

Next to the fact that an AGV can have a job: a pick-up and drop-off, it can also be the case that an AGV is idle. This is the case when the arrival rate of jobs is less than the throughput (capacity) of the system. In this context a vehicle is not necessarily regarded as idle when it is not loaded, since it then might be on its way to perform a pick-up action (and thus having a job planned). There are different ways to deal with these idle vehicles, but some idle vehicle handling must be present in order to prevent idle vehicles from downgrading the performance of the system.

When a job arrives, multiple things must be arranged. At first a job assignment mechanism decides which job is assigned to which AGV. Job assignment is the activity of assigning jobs in the form of a combination of pick-up and drop-off location(s) to AGVs. This means that also the pick-up and drop-off locations are selected. This can be one or multiple locations, as described before. Then a path planning mechanism plans a route for the assigned AGV from pick-up to drop-off. In case of multiple pick-up and drop-off locations a decision is made on which pick-up and which drop-off location is chosen in order to minimize the time spent. Furthermore some traffic control is performed. This determines the actual execution of the AGV movement like what speed to drive, preventing collisions and preventing deadlocks.

Job assignment, path planning and traffic control can be handled separately or using an approach in which this is combined. The combination of job assignment, path planning and traffic control rules determines how the system functions. Whether there can be deadlocks, how much charging is needed and how many AGVs are needed are all influenced by this set of rules.

The goal in controlling AGVs in the *AgvSorter* system is to minimize the makespan. The makespan is the total time it takes to execute all jobs. Moreover, the throughput (jobs/unit time) must be maximized. This goal must be achieved in a very challenging environment. A system can consist of between 100 and 1000 AGVs, in a grid of maximally 10.000 tiles with a maximum of 100 pick-up locations and 500 drop-off locations. This means that it is a dense grid, with up to 20% of all tiles being occupied by AGVs. In the set of rules this must be taken into account: when every AGV can for example claim 2 tiles ahead, already 60% of the grid is occupied (claimed).

Simulation model
To compare different *AgvSorter* control models with each other, simulations can be performed. In order to perform these simulations, a simulation model was created within Vanderlande [5]. This model is developed in MATLAB. It is a discrete event simulator which means that the system is modeled at specific times: each time an event is scheduled the model is updated. An event can for exampel be movement of an AGV (kinematics), communication, software timers or visualization.

To be able to cope with all the earlier mentioned rules, the model is divided into multiple classes. Here, there is a clear difference between classes that have to do with the AGVs (for example AGV kinematics) and grid control (route planning, providing jobs and so on). Information is exchanged between the different classes.

A visualization of the *AgvSorter* model for a 6x6 grid is shown in Figure 2.5. In this visualization, the tiles are indicated in the form of yellow squares. A tile that has a grey color is a not drivable area (or obstacle) and the light orange and dark orange squares indicate the tiles in the planned path and currently claimed tiles respectively for AGV 1. Currently claimed tiles are tiles that are released to a certain AGV, allowing an AGV to enter these tiles. Furthermore, the diamonds are pick-up points (purple) or drop-off points (light blue). The rectangles indicate AGVs that are driving around. There are different colors for the different states an AGV can be in. It can be moving (loaded/unloaded), it can be in the process of loading or unloading and it can be not moving (loaded/unloaded).

The model allows for different approaches for job assignment, path planning and traffic control. Currently, different approaches are used for this. This is done in order to prevent the model from becoming too complex in terms of computational power. Research is being performed on other job assignment strategies and on combining path planning and traffic control (this project).

As mentioned before, job assignment happens using a job assignment strategy which uses a weighted sum of driving time and job age [12]. Path planning happens using an updated weights algorithm [6]. This algorithm is an adapted A* algorithm in which the weights of vertices are updated based on for example how long AGVs stay at a these vertices. When time proceeds and no AGVs pass the same vertex, its weight is gradually lowered. Also costs for turning are taken into account. When planning a new route, all weights (or costs) are taken into account to predict the most optimal route. Traffic control is performed using tile reservations. An AGV is only allowed to claim the next tile in its path, if no other AGV is present at that tile. If this next tile is occupied, the AGV waits.

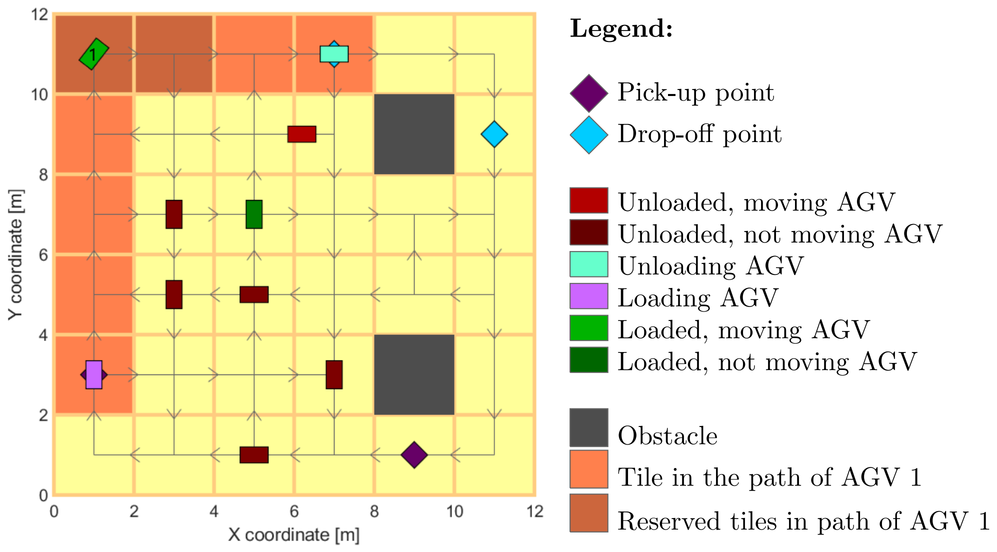


Figure 2.5: Sample layout of the grid-based *AgvSorter* system [6].

The separation of path planning and traffic control in the current model sometimes results in deadlocks. In such a situation one or multiple AGVs are waiting for other AGVs which in turn are waiting for other AGVs and thus are stuck. Therefore a deadlock avoidance and deadlock detection algorithm are present [7]. The deadlock avoidance algorithm makes sure an AGV cannot claim the next tile in its path when this eventually results in a deadlock. To also cover unpredictable situations, a deadlock detection algorithm periodically checks whether no deadlocks are present. If there are deadlocks, the simulation is terminated.

Since the grid in the *AgvSorter* model allows for flexible layouts, it is suited for simulating different environments. These might be warehouses, parcel sorting facilities and airport baggage systems. It is due to this flexibility that it is believed that this simulation model can also help in creating a standardized control mechanism for all market segments within Vanderlande.

Desired performance and improvements
One of the key performance indicators (KPIs) of the *AgvSorter* system is throughput. In this sense throughput is the amount of jobs that can be processed per unit time per area. An indicator that is related to the throughput is the makespan: the total time it takes to process all jobs. Although not good measurable, the system must be scalable since the system must work for an amount of 100 up to 1000 AGVs on floors with maximally 10.000 tiles, the system must be able to handle all cases in between. The simulation model must be able to work real-time and therefore the simulation runtime must not be too high.

In order to come up with the best possible performance, in the *AgvSorter* simulation model multiple methods for job assignment, path planning, traffic control and deadlock handling have been tested. As mentioned, currently the job assignment, path planning and traffic control are separated in the *AgvSorter* simulation model. A next step is to to combine path planning and traffic control (job assignment will still be performed separately). In this way deadlocks are automatically avoided and the performance might go up. Therefore this is the focus of this research project. The goal is to find a way to combine path planning and traffic control and to get a throughput that is as high as possible. Next to this the goal is to reduce the complexity as much as possible: real-time control must be possible. In Chapter 3 different approaches for separate path planning and different approaches for combined path planning and traffic control are discussed.

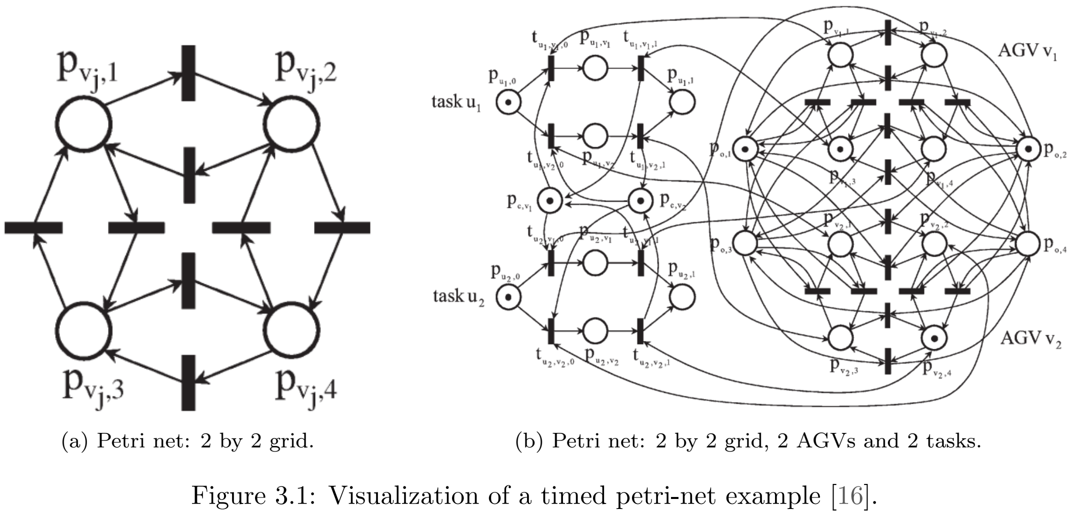
3. Combined path planning and traffic control

From the previous chapter it becomes clear that the goal of this research project is to combine path planning and traffic control in order to improve throughput, minimize makespan while not having too high complexity. In this chapter three approaches to combine path planning and traffic control (routing) into one approach are discussed. First a timed petri-net approach is introduced. Next, a time-window based scheduling approach is presented. This time-window based approach still needs some shortest path algorithm, of which some are discussed in Appendix A. The third approach is a context-aware routing approach. At the end of this chapter a short comparison and conclusion follows.

3.1 Petri-nets

In petri-net approaches, rather than using the grid layout being a graph with vertices and edges a petri-net is used. Petri-nets consist of a number of places with arcs to transitions and vice versa. All places can hold a number of tokens. Every combination of number of tokens over the places is a marking, the state at which the system is at that point. A transition in the petri-net is possible if all input places, connected with arcs, hold equal or more tokens than the weight of that arc. During a transition the tokens move resulting in a new marking. Timed petri-nets are petri-nets with a specific time duration specified for each transition, thus making it possible to assign a time duration to a certain event.

An example of a timed petri-net for an AGV application is shown in Figure 3.1 [16]. Figure 3.1a shows the petri-net in case only a 2 by 2 grid is modeled for one AGV. This still looks rather simple. However, when 2 tasks, and extra AGV and non-colliding constraints are added the petri net already looks like the one in Figure 3.1b. In the figure, *p* represents vertices that can hold tokens and *t* represents transitions between vertices. Note that the complexity increases a lot when increasing the number of AGVs, grid size and number of tasks.



(a) Petri net: 2 by 2 grid. (b) Petri net: 2 by 2 grid, 2 AGVs and 2 tasks.

Figure 3.1: Visualization of a timed petri-net example [16].

According to Wu and Zhou [17], timed petri-nets can solve the problem of conflict-free routing in an optimal way while preventing deadlocks. Therefore this option might be considered for this project. They propose an approach that can handle both bidirectional as unidirectional paths. The presented model has only few locations, few edges, few AGVs and no simulations are provided.

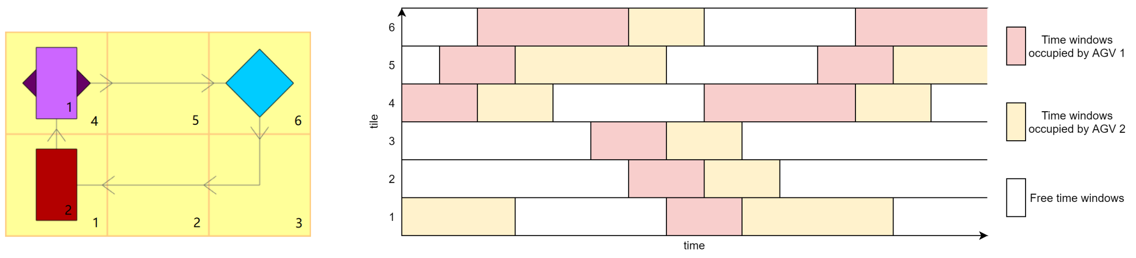
Nishi and Tanaka [16] propose a timed petri-net approach in which all the transitions take a certain amount of time. This timed approach in combination with an optimization algorithm is able to find the optimal firing sequence, resulting in routes for AGVs to follow. The resulting model is quite complex when increasing the size of the system. Therefore they use some decomposition in order to make the problem simpler. When the petri-nets are decomposed more, the complexity of the petri-nets goes down but the complexity of the optimization goes up.

Kammoun, Ezzedine, Rezg and Achour [18] introduce a flexible manufacturing systems (FMS) scheduling approach based on timed petri-nets. They conclude that the complexity of the problem increases exponentially when increasing the system. Therefore they propose a decomposition approach in combination with a genetic algorithm approach for solving large-scale scheduling problems.

Petri-net approaches thus might be interesting for solving the AgvSorter scheduling problem. However, there are problems in terms of complexity due to the nature of petri-nets. Decomposition approaches are available to reduce complexity, at some cost. Overall this approach seems too complex to tackle the grid-based AgvSorter problem. Also when having larger systems, it is hard to understand what is happening (which is already the case in the simple example of Figure 3.1) and therefore difficult to maintain.

3.2 Time-window based scheduling

In time-window based approaches for grid-based control, for each tile a list of free and reserved time slots (time windows) exist. Together with information on which tiles are connected by edges and which free time windows are reachable from each other this forms a free time window reachability graph. When planning a new route, time windows can be reserved for each tile that must be traversed. Using time windows, all other planned routes of all other AGVs are taken into account. This can be done by using some shortest path algorithm for planning a path and delaying the AGV route plan so that the path fits in the time windows. When this succeeds, a conflict-free route is found which in an ideal world without disturbances also prevents deadlocks (assuming that an AGV in the end is always routed to a parking spot, which is discussed in Section 4.3). Two examples of shortest path algorithms can be found in Appendix A. Figure 3.2 shows an example of a grid with 2 AGVs and the reserved time windows. An AGV can only reserve a new tile once that tile is free. The red areas indicate reserved time slots for AGV 1, the yellow areas indicate reserved time slots for AGV 2 and the white area indicates the free time windows. The size of the colored blocks refers to the actual time of the reservations.


1
4
5
6
2
1
2
3
6
5
4
3
2
1
tile
time
Time windows occupied by AGV 1
Time windows occupied by AGV 2
Free time windows

(a) Example layout.

(b) Open and occupied time windows.

Figure 3.2: Visualization of a time window example.

Taghaboni and Tanchoco [19] present an approach in which time windows are introduced (although not called time windows). They make use of Dijkstra's algorithm [20] for planning a shortest-distance path. This path is then, based on the maximum speed of an AGV, put in a time window table as shown in Figure 3.2. If the planned path overlaps reserved time windows for a certain tile (meaning that the tile is not free at that moment in time), two things happen. The planned route is delayed in such a way that the original route is still possible (resulting in the same path but a longer traversal time) and re-routing happens (excluding the tile in which reserved time windows overlapped). If this re-planning results in a lower traversal time, the new path is chosen. Otherwise the AGV is slowed down such that conflict is avoided. When there are multiple conflicts in terms of overlapping time windows, this process is repeated.

The proposed time-window based approach of Taghaboni and Tanchoco in combination with some shortest path planning algorithm (Dijkstra in this case) might result in lower total traversal times compared to separate path planning and traffic control. This is the case since when a conflict occurs re-planning is performed. This might result in a faster path. Since many checks have to be performed when more vehicles are present in the system, this can become quite complex. When using this approach, still some traffic control is needed in case perturbations or breakdowns occur (this is the case in all time-window based approaches).

Zhang, Peng, Wei and Kou [21] use a likewise approach. They use A* [22] to calculate the shortest path. However, not the maximum speed of an AGV is used. More realistic traversal times are used, to cover for example lower speeds in corners, acceleration and deceleration. This approach was tested for a 20 by 20 grid with few AGVs present. No simulation results are presented.

Also similar to the approach of Taghaboni and Tanchoco is the approach of Lee, Lee and Choi [23]. In this approach not only one shortest path is used but the top k loopless shortest paths are calculated offline (using Dijkstra's algorithm) for each job and saved in a routing table. Next to this offline routing table generator (RTG) there is an online traffic controller (OTC) After this, in an online way all these routes are checked in the same way as in the approach of Taghaboni and Tanchoco. However, no re-planning occurs. The path that results in the shortest traversal time is chosen and executed.

When k is set sufficiently high, the optimal solution (in an ideal world) can be found. Problems in this approach might arise when start and destination are far apart (resulting in many possible shortest paths, thus a very high k needed for optimality) or when multiple sources and/or destinations are possible. This causes the routing table to become very large, meaning that many solutions must be checked. The complexity can be lowered by using a lower k. In that case less possible solutions must be checked. However, choosing a lower value for k can result in not optimal solutions.

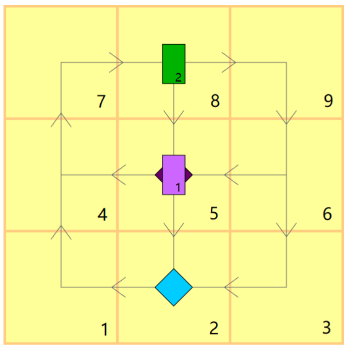
Fan, Gu, Yin, Liu and Huang [24] propose approximately the same approach as Lee, Lee and Choi, in a multi-AGV logistics center. First candidate paths are generated, then a schedule-path is created for each candidate using time window searching. After this the path with the lowest arrival time at the destination is chosen. This approach was tested with layouts upto 64 by 64 tiles and up to 1000 AGVs. Since one can always find time windows for a route, no failures in path searching occur.

An approach that is also similar to the k-shortest path approach is the approach presented by Srisvastava, Choudhary, Kumar and Tiwari [25]. The main difference is that they use agents for controlling parts of the system. The guide path agent is responsible for finding the k shortest paths (for example using the approach of Lee, Lee and Choi). A journey time database agent determines the time it takes an AGV to traverse a tile in the path. There is a tile controller agent that uses a link occupation table (time windows) to fit the planned routes in order to prevent collision and deadlocks. An online traffic controller agent then sets rules for movement of AGVs based on the information of the tile controller agent. An order agent generates new jobs and an AGV agent controls the actual movement of an AGV. All these agents communicate using communication agents.

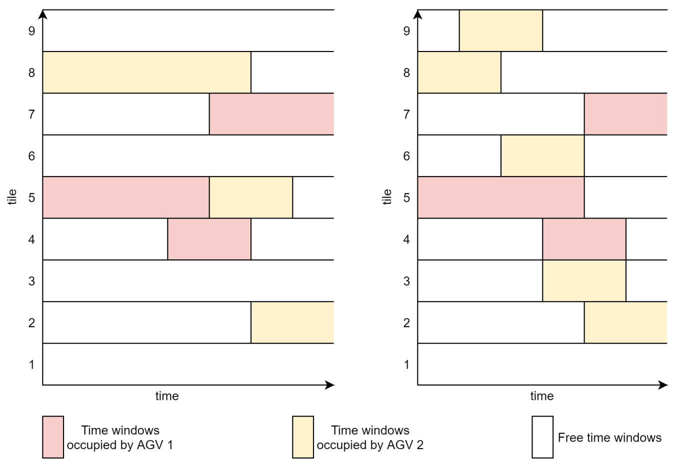
Time-window based scheduling can be used to fit pre-planned routes into a occupation graph. These occupation graphs exist for all tiles and are basically a set of free and reserved time windows with connections between occupied time windows in a path of an AGV and connections between free time windows in case time windows are reachable from each other. As mentioned, this approach still requires a planned path. Therefore the solutions might not be optimal in terms of total traversal time. However, in the k shortest path approach optimality is guaranteed (in an ideal world) when k is set to infinity (then all possible routes are covered). This is too complex for larger layouts. Note that optimality in this case means the optimal routing option given the planned path and given the occupation graph. Another way of time-window based scheduling can also be used: context-aware routing.

3.3 Context-aware routing

Context-aware routing is similar to time-window based scheduling. Occupation graphs with free and reserved time windows are again used to plan paths for AGVs. Where the previously discussed time-window based scheduling approaches need a pre-defined path (or k pre-defined paths) using some shortest-path algorithm, in context-aware routing this is not needed. In context-aware routing no path is planned on the geographical graph, but a path is planned in the time window graph (occupation graph). In the time-window example in Figure 3.3 this becomes clear. In this case AGV 1 is loading at tile 5. The destination of AGV 2 is tile 2. When a shortest path algorithm is used, the time window graph would look like the left side of Figure 3.3b (in red the reservations of AGV 1 and in yellow the reservations of AGV 2). However, the solution on the right would result in a lower traversal time for AGV 2. So, path planning in the occupation graph (context-aware routing) would give a better result. Again, the size of the reservations refers to the actual time of that reservation.



(a) Example layout.



(b) Open and occupied time windows.

Figure 3.3: Visualization of a time window example.

Since while planning a route the controller is aware of the plans of all the other AGVs, i.e. context, this is called context-aware routing. For routing through the occupation graph, for example modified A* or Dijkstra algorithms can be used. Path planning approaches that use weights for tiles and edges are not needed here since now real-time occupation information can be used rather than a prediction.

Kim and Tanchoco [26] presented this approach first. They propose an approach which is based on Dijkstra's algorithm. As in the time-window based case a list of free and reserved time windows is maintained for each tile. What is used is a time window graph in which all vertices represent free time windows and all edges represent the fact that a specific free time window is reachable from the previous time window. the complexity of their approach in the worst case is $\mathcal{O}(v^4 n^2)$ in which v is the number of AGVs and n the number of tiles. When the layouts get bigger and the number of AGVs increases, the complexity becomes very high.

Huang, Palekar and Kapoor [27] discuss an approach in which all arcs (edges) and nodes have an occupation table with blocked periods and free time windows. Using these occupation tables the original map including all routes is updated every time step in order to take into account delays due to blocked time periods in the occupation table. The complexity of the presented approach depends on the number of free time windows available. No simulations are provided for this approach.

Ter Mors, Zutt and Witteveen [1] propose a similar approach in which an agent plan (i.e. a set of resources and time intervals) must be feasible (both in terms of map route and free time windows) and reachable. An informed A*- search using a heuristic distance function makes sure the complexity of the algorithm goes down. The use of A* makes sure solutions are found faster. The complexity in the worst case for this approach is lowered to $\mathcal{O}(vn\log(vn) + vn^2)$ in which v is again the number of AGVs and n the number of tiles.

In a later publication, Ter Mors and Witteveen [28] compare their context-aware routing approach to local intersection management for bidirectional problems. In simulations it is shown that the context-aware routing mechanism performs a lot better in case no disturbances such as delayed vehicles are present. However, when these delays are introduced the performance degrades fast. The performance can be maintained when effective plan repair mechanisms are introduced. Ter Mors and Witteveen also present some plan repair mechanisms [29].

Penners [30] investigates the applicability of context-aware routing for the Adapto [4] system of Vanderlande. The approach of Ter Mors, Zutt and Witteveen was applied. Here it was concluded that the context-aware routing approach can be applied for any AGV system with orthogonal guide path tiles (as is the situation in this project, next to possible other tile shapes). Some buffer rules were introduced to avoid cyclic deadlocks. In this case a cyclic deadlock appears when a tile is visited multiple times due to other AGVs blocking further routes towards the destination. Lienert and Fottner [31] describe an adapted version of this work for controlling a shuttle system.

The work of Hvezda, Rybecky, Kulich and Preucil [32] is about the application of context-aware route planning (CARP) in an automated warehouse application with bidirectional lanes. To upgrade the performance, several random priority shuffles in the planned trajectories the AGVs that influence the optimal route of the to be planned trajectory the most are performed. If these shuffles lead to a smaller makespan, the plan of that AGVs are changed. The approach was simulated in 21 different maps with a grid size of minimum 20 by 20 using 100 vehicles and proved to perform better than existing context-aware applications.

Silver [33] describes a context-aware approach for the video-game industry. Three algorithms for finding non-colliding routes using full information of the routes of all other agents (AGVs in this case) are described. The Cooperative A* (CA*) algorithm finds paths through the time-window occupation graph. A Hierarchical Cooperative A* (HCA*) algorithm works the same way but using a heuristic to upgrade the performance. Since no full path is planned in once, deadlocks might still occur (although this is not likely).

Next to the mentioned algorithms that are similar to the work of others, Silver presents a Windowed Hierarchical Cooperative A* (WHCA*) algorithm. This algorithm only plans a part of the path in detail using context-aware routing and uses regular A* (through the geographic map) for the rest. When a part (around half) of the detailed path is executed, a new part of the path is planned in detail. This approach makes the complexity significantly lower while improving the performance (since disturbances that occurred are taken into account every time a new detailed partly path is determined). This approach might be a solution to the complexity problem in context-aware routing.

To conclude, context-aware routing is different to time-window based scheduling in the sense that rather than planning a shortest path on the grid, a route through the time-window occupation graph is determined. This means that no separate path planning approach is needed. One main drawback of context-aware routing is the complexity. However, using the concepts of for example Silver [33] these issues might be solved.

3.4 Conclusion and comparison

In previous work shortest-distance (or shortest time) approaches are used for path planning in the AgvSorter application using separate traffic control. However, in the context of this project this is not enough since the goal of this project is to combine path planning and traffic control.

Therefore, in this chapter three approaches for combining path planning and traffic control have been presented.

The first mentioned approach uses timed petri-nets. These timed petri-nets get very complicated as the number of AGVs, layout size and number of tasks increase. There are some petri-net decomposition approaches that can solve this issue. However, the optimization gets more complicated as the petri-nets are decomposed more. Moreover, when having larger systems the visualization becomes harder and harder to understand making it a difficult system to maintain.

The second approach is time-window based scheduling. In this approach a separate path planning algorithm is used to find a shortest distance or lowest cost route. These path planning algorithms can for example be Dijkstra's algorithm [20], an (modified) A* algorithm [22] or the currently in the *AgvSorter* model implemented updated weights algorithm [6]. The routes are planned on a first come first serve (FCFS) base and the determined route is fitted in a time window occupation graph. Several approaches for doing this exist. Suboptimality occurs when the shortest path determined does not result in the shortest traversal time.

The last approach discussed is the context-aware routing approach. Here the path is not planned based on shortest path but based on shortest traversal time. This is done by finding a route through the time window occupation graph rather than through the geographical layout. This approach might result in the same or higher performance, at the cost of complexity. Approaches for reducing this complexity are also presented.

For all three approaches there are some issues concerning disturbances and delays. For example: when an AGV traverses an edge slower than expected or breaks down, the whole planning might not fit any more. This can be solved by delaying all next events or another way of dealing with these uncertainties must be introduced.

4. Context-aware route planning algorithm

Following the literature on combined path planning and traffic control in Chapter 3, context-aware route planning seems be the best solution for solving the given routing problem. In this chapter the algorithm used for solving the context-aware routing problem is presented. First the main algorithm is presented, after which free time window reachability is defined, idle vehicle handling is discussed, different kind of costs are introduced, the method to plan multiple routes is explained, some information on overlap on consecutive reservation is presented and lastly the created Matlab model including some examples and practical issues are discussed.

4.1 Algorithm description

The context-aware route planning (CARP) algorithm is nothing else than a form of Dijkstra's algorithm [20] (or the A* algorithm [22] in case an extra heuristic for expected costs to the drop-off location is used) applied not on a geographical layout but on a temporal graph. More information on the working of these algorithms is presented in Appendix A. For a given grid layout, a pick-up tile r_s, drop-off tile r_d and start time t_s, the entry time into r_d and the followed route for the shortest path, or better said: the path with the lowest total costs, from r_s to r_d needs to be calculated. The basic algorithm (a modified version of the algorithm determined by Ter Mors, Zutt and Witteveen [1]) used in this project is presented in Algorithm 1.

Algorithm 1 Context-aware routing algorithm [1].

```
1:  if $\exists[f_{s,v} \in F_s | t_s \in f_{s,v}]$ then
2:      $Q \leftarrow \{(r_s, t_s, 0, 0, 0, t_s + h_{s,d,v})\}$
3:      $V \leftarrow \emptyset$
4:  while $Q \neq \emptyset$ do
5:      $(r_i, t_i) \leftarrow \text{argmin}_{(r,t)\in Q}(t_i + h_{i,d,v})$
6:      $Q \leftarrow Q \setminus \{(r_i, t_i)\}$
7:      $V \leftarrow V \cup (r_i, t_{i,\text{entry}}, r_h, t_{h,\text{entry}}, \text{stage}, t_{i,\text{entry}} + h_{i,d,v})$
8:      if $r_i \in r_d$ then
9:          return $(r_i, t_i, \text{route})$
10:     for all $f_{j,v} \in \rho(r_i, t_i)$ do
11:         $t_{\text{entry}} = \max(t_i + D(r_i), t_{j,v,\text{start}})$
12:         if constraints_met then
13:             $Q \leftarrow Q \cup (r_j, t_{j,\text{entry}}, r_i, t_{i,\text{entry}}, \text{stage}, t_{j,\text{entry}} + h_{j,d,v})$
14:             $F_j \leftarrow F_j \setminus \{f_{j,v}\}$
```

In line 1 for the start tile it is checked whether there exists a free time window $f_{s,v}$ (with v the index of the free time window) in the set of free time windows F_s for which the start time t_s is in the free time window. The time window is added to open list Q (a list with time windows that can be reached and from which the route can be continued) in line 2. The open list item added includes the pick-up tile r_s, start time t_s, three 0's and expected costs $t_s + h_{s,d,v}$ in which $h_{s,d,v}$ represents the heuristic costs from the v^{th} time window on pick-up tile r_s to the drop-off tile r_d. In all cases except the open list element at the start tile the zero entries would be filled with the predecessor tile, entry time to this tile and stage of the route planning. The concept of route planning stages is explained later in Section 6.1. Line 3 initializes the visited set V with in it the open list items that were visited, which is empty at the start.

Line 4 indicates that a loop is run as long as the open list Q is not empty. In every loop the open list item with the lowest costs (entry time t_i to tile i plus $h_{i,d,v}$, the expected costs to the drop-off tile) is chosen in line 5, removed from the open list in line 6 and added to the visited list in line 7 (r_h is the predecessor tile and $t_{h,\text{entry}}$ the entry time to the predecessor). In case the currently visited tile r_i is equal to the drop-off tile r_d, which is checked in line 8, the algorithm is terminated and the drop-off tile $r_i = r_d$, entry time to the drop-off tile t_i and the route followed to get there are returned in line 9.

When the currently visited time window is not a time window on the drop-off tile, a loop is run over all possible next time windows on next tiles in line 10. Here, a possible next time window $f_{j,v}$ must be in the set of reachable time windows ρ_{r_i,t_i} from tile r_i for which the time window starts at t_i. Free time window reachability is discussed in more detail in Section 4.2. In line 11 the entry time to the next time window is determined. This is the maximum of the entry time plus the expected duration of the reservation $D(r_i)$ $(t_i + D(r_i))$ and the start of the next time window $t_{j,v,\text{start}}$. In line 12 it is checked whether the constraints are met: these constraints are that the free time windows must be big enough to cover all the costs needed. If these constraints are met, in line 13 the new time window is added to the open list Q. A new open list item covers the following information: next tile r_j, entry time to this tile $t_{j,entry}$, current tile r_i, entry time to current tile $t_{i,entry}$, the stage in which the route planning is and the expected costs to the drop-off tile $t_{j,entry} + h_{j,d,v}$. The set of free time windows is updated in line 14.

4.1.1 Administration

After a free time window (or part of it) is removed from the set of free time windows, it is no longer available for expansion. To keep track of this some administration must be performed. Next to the fact that there must be an administration for all reservations, also occupation information must be stored. For this, a list of free time windows F_i for tile i or free set is present for all tiles. The inverse of this, the occupation set, includes information on the time windows for which the tile is not available. The same as for tiles, administration of these free sets and occupation sets can also be made for individual AGVs. One main difference is that in a route one or some reservations can be present on a tile whereas the AGV must be reserved at all times (during the route execution). Moreover, the last tile in the route of an AGV must always be reserved until infinity since in theory an AGV must be able to stay there forever.

Route administration happens in the list of visited elements (V in the algorithm). Here, information on every combination of time window and tile that was visited (thus at some point chosen from the open list) during the route planning and its predecessor and entry times is present. Since the predecessor is known for every item in the visited list, the route that must be followed can easily be found by pointing backwards in the visited list and start with the drop-off location (last visited element) in the visited list.

4.1.2 Multiple drop-off locations

In the algorithm it is also possible to use multiple drop-off locations. For example when multiple drop-off points end on the same conveyor or when there are multiple similar machines on a floor. In the algorithm this is indicated in line 8 ($r_i \in r_d$): the (in this case drop-off) location must be in the set of destinations (drop-off locations). In case one of the drop-off locations is reached, the algorithm can now terminate in line 8 of Algorithm 1. An advantage of using this algorithm is that the first drop-off location that is found is always the drop-off location that in theory can be reached the fastest.

4.2 Free time window reachability

In Algorithm 1 line 10 a loop is run over all possible next time windows which are reachable. In this section the notion of free time window reachability is discussed. A free time window is said to be reachable if it is possible to get to this time window from the time window in which the route planning currently is. The notation shown in this section is an altered version of the notation used by Ter Mors, Zutt and Witteveen [1].

A free time window is a period of time at which a tile is not reserved. The set of free time windows is equal to the previously discussed free set F_i for tile i. For tile r_i and a set of reservations $I = \{\tau_1, ..., \tau_m\}$, in which τ represents a reserved time interval, the v^{th} free time window is defined by $f_{i,v} = [t_{i,v,\text{start}}, t_{i,v,\text{end}}]$. Here, t_{start} is the start of a free time window and t_{end} is the end it. This free time window is of interest when Equation 4.1 and Equation 4.2 hold. Equation 4.1 states that the capacity of a tile $C(r_i)$, which is 1 for tiles, cannot be exceeded. This means that no more than 1 reservation can be placed at the same time for one tile. Equation 4.2 states that the free time window must be bigger than the expected duration $D(r_i)$ of the reservation.

$$\forall t \in f_{i,v} : |\{\tau_j \in I | t \in \tau_j\}| < C(r_i) \tag{4.1}$$

$$(t_{i,v,\text{end}} - t_{i,v,\text{start}}) \geq D(r_i) \tag{4.2}$$

Having the notion of free time windows alone is not enough. A free time window must also be reachable to be a candidate next route element. Therefore one can look at free time window reachability, which in Algorithm 1 happens in line 9. In general free time window $f_{j,w}$ on tile r_j is reachable from r_i at time t (meaning $f_{j,w} \in \rho(r_i, t)$ or in words: the free time window is in the set of reachable free time windows) if Equation 4.3 until Equation 4.6 hold.

$$(r_i, r_j) \in S \tag{4.3}$$

$$t \in (f_{i,v} \cap f_{j,w}) \tag{4.4}$$

$$(t - t_{\text{start},i,v}) \geq D(r_i) \tag{4.5}$$

$$(t_{\text{end},j,w} - t) \geq D(r_j) \tag{4.6}$$

Equation 4.3 states that tile r_j must be the successor of tile r_i, thus that there is a driveable edge between the tiles. Equation 4.4 states that t must be in both free time windows. Equation 4.5 states that there must be enough time for the reservation on the current tile and Equation 4.6 states that there must be enough time for the reservation on the next tile. The total set of reachable free time windows is the sum of $\rho(r_i, t)$ for all possible t's, see Equation 4.7.

$$\rho(f_{i,v}) = \bigcup_{t \in [t_{\text{start},i,v} + D(r_i), t_{\text{end},i,v}]} \rho(r_i, t) \tag{4.7}$$

4.2.1 Three layered reachability

In the context-aware route planning algorithm used for this project a three layered reachability check is used. This three layered reachability means that every time a next possible free time window is visited, it is also checked whether this requires updates on the reservation on the current and previous tile.

For example, this is needed in case a turn is needed on the current tile to go to the possible next tile. When a turn must be made on the current tile, extra time for turning is needed. But since for turning an AGV must come to a full stop, the AGV must already start decelerating on the previous tile. Therefore on both the current and the previous tile extra time is needed. Therefore the three layered check as shown in Equation 4.8 until Equation 4.10 is performed. Here $t_{h,\text{leave}}$, $t_{i,\text{leave}}$ and $t_{j,\text{leave}}$ represent the (adapted) end times of the reservation on the predecessor tile h, current tile i and successor tile j respectively. $t_{\text{end},h,v}$, $t_{\text{end},i,w}$ and $t_{\text{end},j,x}$ represent the end of the free time windows directly after the reservations on that tile (with v, w and x the indices of those particular free time window). Only if the possibly adapted reservation end time is before the end of the free time window after this reservation, the reservations are possible (and thus in the example the turn is possible).

$$t_{h,\text{leave}} \leq t_{\text{end},h,v} \tag{4.8}$$

$$t_{i,\text{leave}} \leq t_{\text{end},i,w} \tag{4.9}$$

$$t_{j,\text{leave}} \leq t_{\text{end},j,x} \tag{4.10}$$

4.3 Idle vehicles

When a route is found for an AGV and no new job is available immediately, a vehicle is idle, meaning that it does not receive new pick-up, drop-off information. This might cause a drop-off (or other) tile to be blocked by such an idle AGV. Another problem is that there might not be free time windows on the surrounding tiles to leave the tile after a new job is received. Therefore the need for idle vehicle handling arises.

The solution for idle vehicle handling is the use of parking spots. Parking spots are tiles designated for conflict-free infinite reservations and thus ideal for idle vehicles. Using the algorithm discussed before, a route is therefore always planned from the current location of an AGV to the pick-up tile the drop-off tile(s) to a parking spot: this makes sure an AGV is always able to get to a position in which it cannot block other AGVs or special locations in the grid.

The reservation that an AGV makes on a parking tile is always until infinity. When the AGV gets a new route (from the first unclaimed tile in the path after the drop-off tile of an AGV or from the parking tile or any other tile on the route in between) this reservation is removed and the parking spot is available again. The choice can be made to have a fixed parking spot per AGV or to let an AGV always route to the free parking spot it can reach the fastest (this happens in the same way as when having multiple drop-off locations, as discussed earlier).

4.4 Costs

Now it is clear when and how certain free time windows can be used for a route, it is possible to look at the individual reservations more closely. An AGV path is dependent on multiple factors. For example how fast an AGV can drive, when to turn, when to (un)load, how fast to accelerate or decelerate and many more factors play a role. In order to make an approximation for the length of a reservation on a certain tile, some cost calculations are done. An approximation model is used since every AGV is in practice different and an exact (or close to exact) model would be too complex.

For the calculation of the costs for driving a certain tile, multiple factors are taken into account in the approximation. An important thing to know is that a tile is chosen in such a way that an AGV can turn on it without having to enter another tile. Moreover, an AGV needs to stop for special actions like turning, (un)loading and waiting. The cost components which are used in the model are costs for moving, turning, (un)loading, acceleration, deceleration and some heuristic costs. Note that the costs are estimations, which is further discussed in Section 4.8.

4.4.1 Cost components

First the moving costs are considered. These costs are made for traversing a certain edge and can be calculated by the length of the edge to traverse L_{edge} divided by a constant v_m (speed), as presented in Equation 4.11. Note that for simplicity reasons the assumption here is that the speed is constant. To make the costs somewhat more realistic, extra costs are added for acceleration and deceleration later.

$$C_m = \frac{L_{\text{edge}}}{v_m} \tag{4.11}$$

Next to the fact that there are costs for traversing an edge between two tiles, there are costs for turning on a certain tile. Using Equation 4.12 these costs can be calculated. The costs for turning C_t are determined by the turning angle θ_t divided by a constant rotational speed v_t. Similar to the costs for moving, for simplicity reasons it is assumed that the rotational speed is constant.

$$C_t = \frac{\theta_{\text{turn}}}{v_t} \tag{4.12}$$

When a vehicle arrives at a pick-up or drop-off location, a loading or unloading event must take place. This means that extra costs for loading C_l and costs for unloading C_u must be taken into account. Since the costs for loading or unloading do not depend on movement by the AGV the values are constant.

As mentioned earlier, for special operations like turning, (un)loading or waiting an AGV must first come to a complete stop. Since in the costs for moving C_m only the costs for traversing an edge at constant speed are taken into account and the costs for acceleration and deceleration are quite significant, some extra costs must be included for acceleration and deceleration. For simplicity reasons the decision was made to add these costs as constants: C_a for acceleration and C_d for deceleration. The values of these constants depend on the maximum speed and acceleration and deceleration of an AGV. As an example, the costs for acceleration C_a can be calculated as shown in Equation 4.13, in which v_{max} is the maximum speed and a the acceleration of an AGV.

$$C_a = \frac{v_{\text{max}}}{a} \tag{4.13}$$

When all the costs are added up, the total costs or duration of a reservation can be determined. This is shown in Equation 4.14. In this equation also boolean operators, denoted with X, are present. When one of the actions presented above is not performed on a certain tile, the boolean operator has a value of 0. Otherwise, the boolean operator has a value of 1. Next, an example in which the cost subdivision is visible is presented.

$$C = X_m C_m + X_t C_t + X_l C_l + X_u C_u + X_a C_a + X_d C_d \tag{4.14}$$

Heuristic costs

The costs discussed so far are the visible costs. Visible in this case means that when for one of the mentioned actions extra costs are needed, this will be added to the total reservation costs and these costs will thus be visible in the reservation. However, also hidden costs are present: heuristic costs. These costs help in speeding up the routing process.

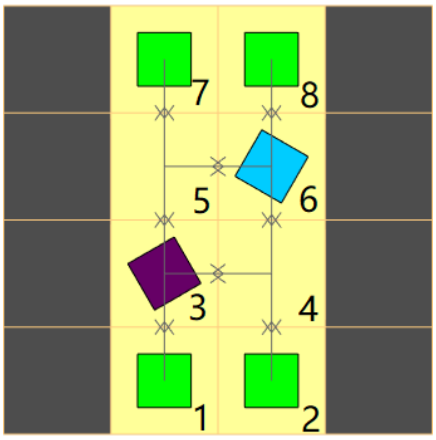
In Algorithm 1 the open list time window with the lowest costs is visited first. Part of these costs are the costs to get to that certain time window. This part of the costs can be calculated using the different reservation parts discussed before. In order to let the algorithm to find the route towards the drop-off tile faster, the expected costs to the drop-off tile, or heuristic costs can be added. Equation 4.15 shows that the total costs for visiting a certain time window on a tile $F(r_i)$ is built up of the sum of the costs to get to the time window on that tile $G(r_i, t_{i,\text{entry}})$ and the expected costs to get from that tile to the drop-off tile $H(r_i, r_d)$.

$$F(r_i) = G(r_i, t_{i,\text{entry}}) + H(r_i, r_d) \tag{4.15}$$

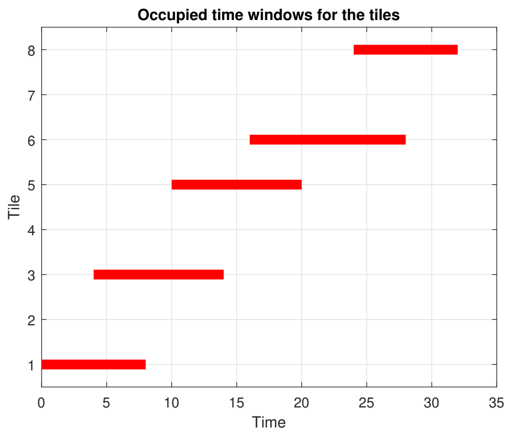
In this project the heuristic costs are determined using the Euclidean distance multiplied by the earlier mentioned move speed v_m. The Euclidean distance is the shortest distance between the center points of the current tile and the destination tile. The choice for Euclidean distance was made due to the fact that this method is more general and can also be used for tile with different shapes.

4.4.2 Example

To illustrate how the costs are built up, an example is presented in Figure 4.1. The meaning of the different colors is presented on the left side of Figure 2.5, the additional green squares indicate parking spots. For this example a route was planned from tile 3 (pick-up) to tile 6 (drop-off) using an AGV that starts at parking tile 1. The starting angle is $-\frac{1}{2}\pi$ (heading down) and the costs are defined as follows: 2s moving costs, 2s turning costs, 2s (un)loading costs, 1s acceleration costs and 1s deceleration costs. Figure 4.1 shows that the route starts at parking tile 1, follows tile 3, 5 and 6 and ends at parking tile 8.



(a) Example layout.



(b) Reservations for the route.

Figure 4.1: A simple example of a route planned.

How the costs in the example are built op is shown in Figure 4.2. Here the route parts correspond to the consecutive reservations in Figure 4.1b (e.g. route part 1 refers to the reservation on tile 1, route part 2 refers to the reservations on tile 3 and so on). As a first example the reservation on tile 1 (route part 1) consists of a turn of 180 degrees (4s), acceleration (1s), movement (2s) and deceleration (1s): 8s in total. Another example is that reservation on tile 3 (route part 2) consists of acceleration (1s), movement (2s), deceleration (1s), loading (2s), acceleration (1s), movement (2s) and deceleration (1s): 10s in total. Part of the costs is red encircled, this is further explained in Section 4.6.

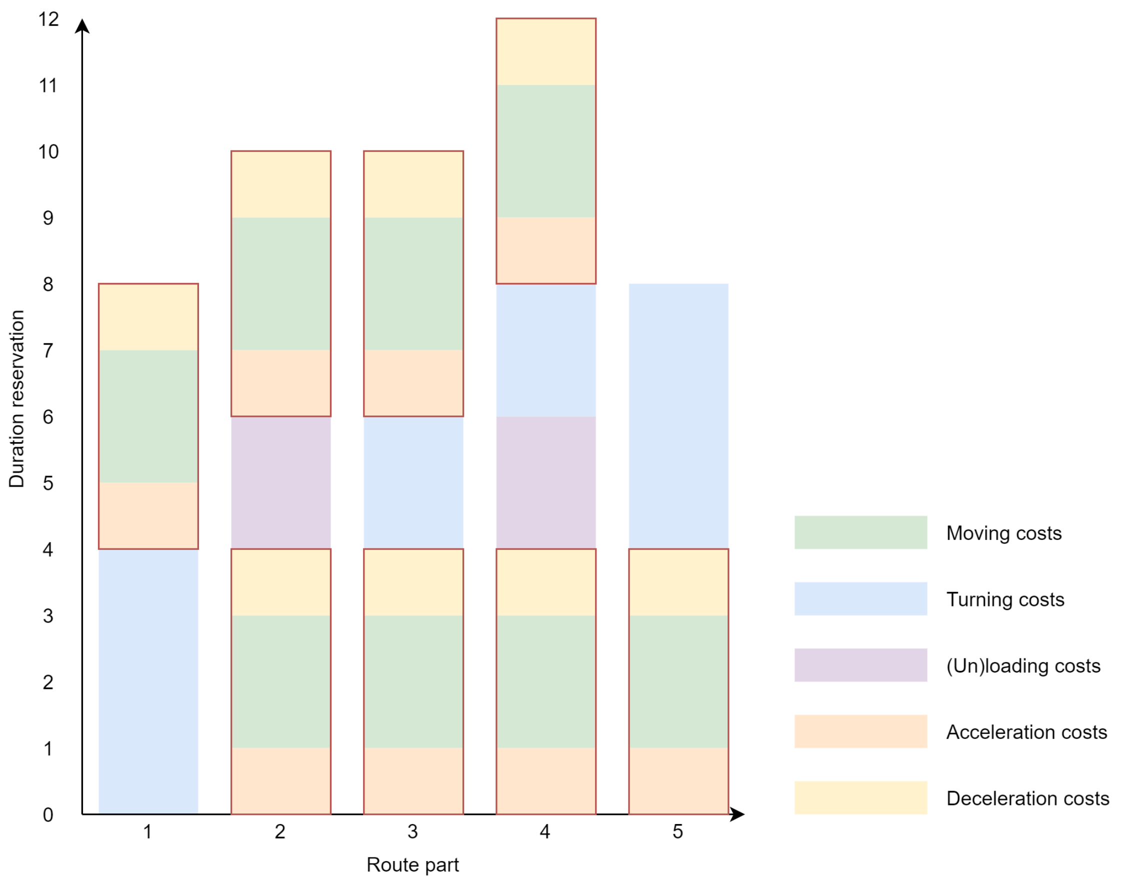


Figure 4.2: Subdivision of the costs in the reservations of Figure 4.1.

4.5 Planning multiple routes

The paragraph on idle vehicle handling discussed the fact that the route of an AGV is always planned from start location to pick-up location to drop-off location to parking spot. Since from a parking spot, AGVs cannot block paths from being planned, it is assumed that all AGVs in the system start at a parking spot. This means that when an incoming job is the first job on that AGV, the route always starts at a parking spot. However, when a new job is assigned to an AGV before it is back at the parking spot, this does not necessarily have to be the case.

In case a new job is assigned to an AGV before the AGV reaches its parking spot, it is possible to plan the new route from the drop-off tile of the previous route or from any other tile on the route to the parking spot, but after the drop-off tile. To do this, a procedure is followed.

When a new job is assigned to an AGV it can first be checked whether it is possible to plan a route directly from the previous drop-off location. When this is not possible it can be checked whether this is possible from the tile in the route after the drop-off tile and so on. When such a route is found, all remaining reservations in the previous route (including the infinite parking reservation) are removed and replaced by the new route. If such a route is not found, the AGV first returns to the parking spot and a route is planned from this parking spot on.

4.6 Overlap

When an AGV traverses an edge between two tiles there is a period of time at which the AGV occupies both tiles: there must be an overlap in the reserved time windows. In the algorithm this overlap starts as soon as an AGV starts traversing the edge between the two tiles and the overlap ends as soon as an AGV has left the edge between the two tiles. In other words, the overlap is equal to the expected time it takes an AGV to drive from the center point of the one tile to the center point of the other tile. In the algorithm the costs in the overlap include acceleration (if the AGV start from a standstill), move costs and deceleration (if the AGV must come to a standstill). In Figure 4.2 for an example case the overlapping costs are red encircled. Note that the overlapping costs of consecutive reservations are equal (thus overlap). In Section 4.8 the fact that using this overlap over-estimates the actual costs is discussed.

4.7 Implementation

In order to do actual routing with the algorithm presented, a model was created in Matlab R2020a. Here, all the aspects mentioned are used as well as main Algorithm 1. What is desired from the routing mechanism is that it determines routes for AGVs to drive and provides information on occupied and free time windows on tiles. In this section some challenges and assumptions in the implementation are presented.

One of the advantages of using temporal routing rather than routing through the geographical layout is that it is, without any problems regarding deadlocks and traffic jam, better possible to use bidirectional lanes. In the implementation model also layouts with bidirectional lanes can be used. In order to make sure not too many options are taken into account and thus the model does not get unnecessary complex a constraint was set: route planning cannot proceed to the same tile as the previous tile in the route so far unless the current tile is a special tile like a parking, pick-up or drop-off station.

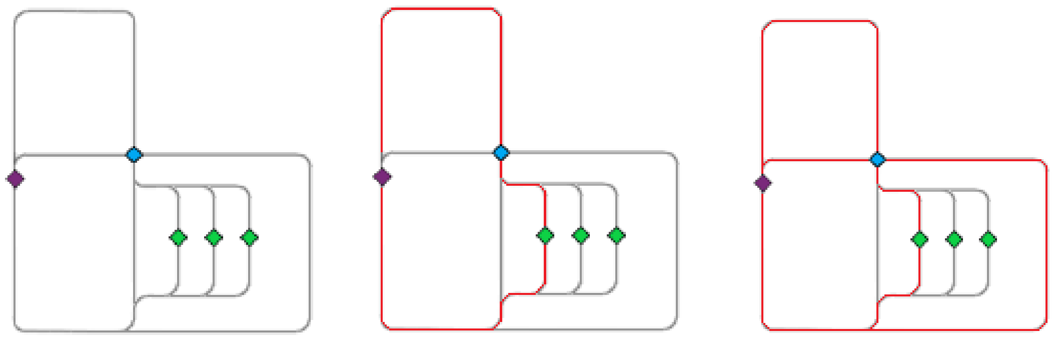
The administration for free and occupied time windows must be stored for every open list element (the same open list elements as discussed in the first part of this chapter). This is due to the fact that otherwise, while planning, a reservation on a tile which might not be used in the final route can block or delay other reservations on this tile. When using a separate administration per open list item only the reservations of the route followed so far are taken into account and no other reservations that have nothing to do with the current route are considered.

In order to prevent idle vehicles from blocking tiles and causing jams, parking spots were added to the system. However, a problem in the optimality of routes found might arise. Figure 4.3 shows a segment-based layout in which this problem becomes clear. Assume that the driving direction is clockwise and that a route must be planned from a parking spot (green diamonds) to the pick-up location (purple diamond), drop-off location (blue diamond) and back to the parking spot. Note that in the example, turning is not possible at the drop-off location. The shortest total route to get to the parking spot via the pick-up and drop-off location is shown in 4.3b. However, the most optimal route from pick-up to drop-off location is shorter but this causes the total route to be longer, as shown in Figure 4.3c.

Since the goal is to minimize the time from pick-up to drop-off, the solution in Figure 4.3c is the desired solution. This might result in longer total routes but since a dense grid is assumed, jobs are mostly planned from the previous drop-off location of an AGV and therefore not the full route must be traversed. Next to this, in a grid layout there are far more options to travel to a location than in the segment-based example.

To make sure that in the Matlab implementation model the solution in Figure 4.3c is chosen, a stage system was introduced. Open list items can be in a stage: routing to pick-up location (stage 1), routing to drop-off location (stage 2) or routing to parking (stage 3). A constraint was set so that always an open list item in the highest stage is chosen. Since the drop-off location in the example is found earlier using the shorter route, this route will be used if it is possible to reach a parking spot from there.

An example of a route planned using the Matlab implementation, and thus the algorithm described in this chapter, is presented in Figure 4.4. Here, two jobs are in the system which both have pick-



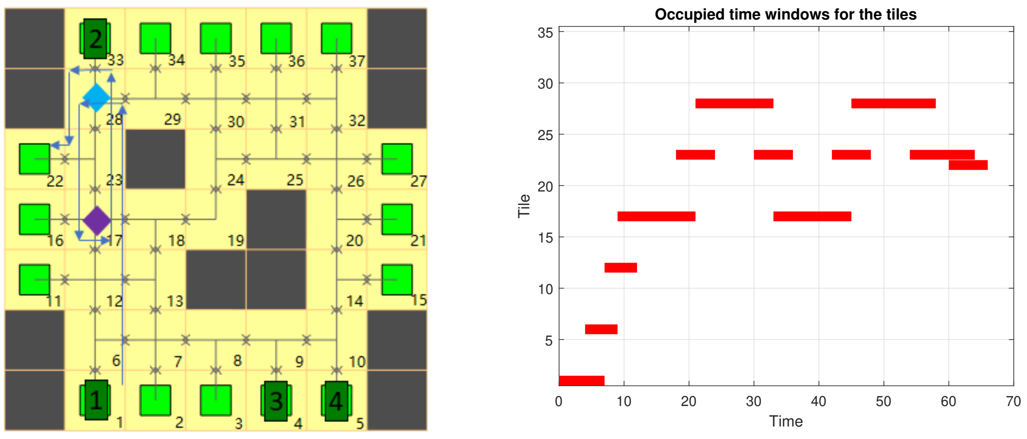
(a) Segment-based layout.

(b) Fastest route start location, parking, pick-up, drop-off, parking.

(c) Fastest route pick-up to drop-off.

Figure 4.3: Segment-based visualization of the optimality problem.

up tile 17 and drop-off tile 28. AGV 1 is used for both jobs and starts at parking tile 1. Costs for moving, turning, loading and unloading are 2, costs for acceleration and deceleration are 1. AGV 1 starts heading down, loading must be performed heading down and unloading must be performed heading left. Figure 4.4a shows the layout and the planned route and Figure 4.4b shows the reservations on the different tiles corresponding to this route. Note that in Figure 4.4b only the occupied time windows for AGV 1 (and not for AGV 2, 3 and 4) are shown.



(a) Layout including route.

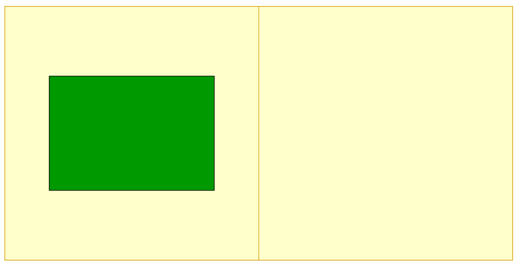
(b) Visualization of the reservations.

Figure 4.4: An example of a planned route for AGV 1 using the Matlab model.

4.8 Shortcomings cost estimation

As discussed in this chapter in the algorithm cost estimations are used to reserve time windows on tiles. These estimations are, as the word says, estimations and therefore per definition not exactly correct. In this section some of the shortcomings of these cost estimations are discussed.

To start with, the overlap in reserved time windows for two consecutive tiles in a route is over-estimated. Figure 4.5 shows how this over-estimation works. In the example there are two tiles and the green AGV drives from the left tile to the right tile. In the algorithm overlap in time windows starts at the position of the AGV in Figure 4.5a, when the AGV starts accelerating (if it starts from a standstill). This overlap ends when the AGV is at the location in Figure 4.5b (after decelerating if the AGV needs to stop on this tile).


(a) The start of the overlap in reservations.

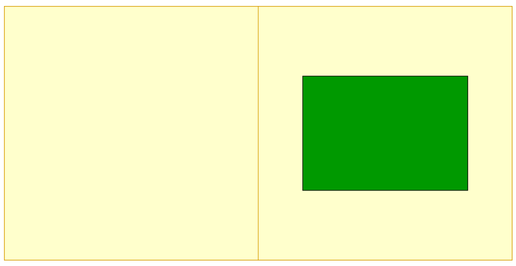
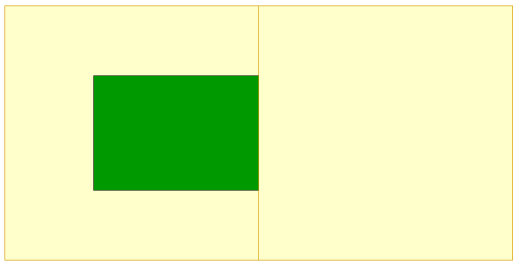

(b) The end of the overlap in reservations.

Figure 4.5: Visualization of the start and end of the overlap in reservations on two consecutive tiles in a route as used in the algorithm.

The actual start and end of a reservation should be as shown in Figure 4.6. Here, in Figure 4.6a the location of the AGV at the moment that the overlap should start is indicated (the first moment that the AGV actually enters the tile) and in Figure 4.6b the location of the AGV at the moment that the overlap should end is indicated. When Figure 4.6 is compared to Figure 4.5 it becomes clear that the overlap in time windows in the algorithm starts too early and ends too late: there is an over-estimation in expected costs.


(a) The start of the overlap in reservations.

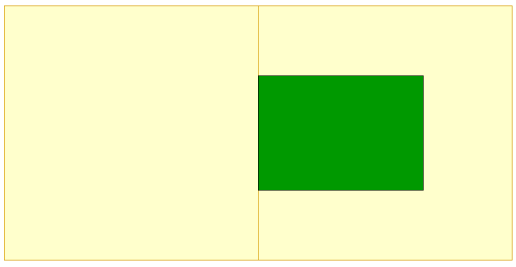

(b) The end of the overlap in reservations.

Figure 4.6: Visualization of the start and end of the overlap in reservations on two consecutive tiles in a route as in execution.

The reason that the mentioned assumption for overlap costs was used is that for square tiles it is relatively easy to calculate the expected point in time that an AGV actually enters or leaves a tile, but that for other tile shapes this is not so generic. Therefore this issue might be subject for future research.

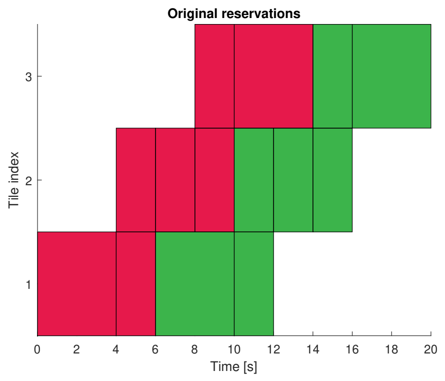
Next to the over-estimation of overlap in reservations, there is also an under-estimation in costs in the algorithm. For calculating moving costs the maximum speed is used and for calculating turning costs the maximum rotational speed is used. Since in reality the speed is not always maximum, there is an underestimation in costs here. In contrast to the costs for movement, for the costs of turning no rotational acceleration or rotational deceleration are taken into account meaning that the algorithm uses an under-estimation of the actual costs for turning.

The fact that the costs are estimations and therefore are at many moments over-estimated and/or under-estimated has an influence on the route planning. Therefore a more accurate cost estimation might be subject of future research. However, in execution this is not a big problem (apart from the routes possibly being not optimal). The inaccuracy in cost estimation (and basically all inaccuracies in the planning and possible delays in execution) is solved by the push/pull mechanism discussed in Chapter 5.

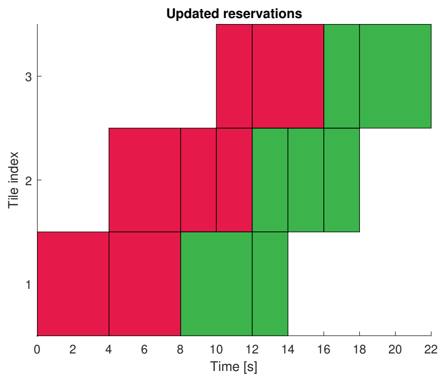
5. Push-pull planning

So far in this report the context-aware routing algorithm was discussed. In an ideal world the routes and time planning following from this algorithm would be enough to execute the job in the system. However, in the algorithm a number of assumptions (for example in the costs) were made for simplicity reasons. Moreover, the behavior of AGVs might not be as expected. Therefore reservations can in reality take longer or shorter than expected. This causes the planning to possibly not be feasible any more. To make the planning suitable for execution again, a push-pull planning is used.

A push-pull planning means that in case a reservation ends earlier or later than expected, the reservations after this reservation (those in any way related to that reservation) are pulled back or pushed up in time in the planning. An example of a push action is shown in Figure 5.1. In Figure 5.1a the original reservations are shown and in Figure 5.1b part of the reservations are pushed forward in time due to the fact that the reservations starting at $t = 4$s are extended. A pull-action happens in a similar way.



(a) The original reservations.



(b) The pushed reservations.

Figure 5.1: An example of a push-action.

In this chapter the push-pull method used is discussed. This method uses max-plus algebra [34] [35] [36], which is introduced first. After this the standard solution for solving max-plus linear systems and an alternative (cheaper) solution method are provided. The chapter continues with some comments on the interaction between the route planner and the push-pull planning and ends with an overview of a full push-pull operation in the context of this project.

5.1 Max-plus algebra

As mentioned, max-plus algebra [34] [35] is used to perform push-pull operations in this project. Max-plus refers to the two operators that are used for this method: the max-operator and the plus-operator. These two operators are shown in Equation 5.1 and Equation 5.2 respectively.

$$x \oplus y = \max(x, y) \tag{5.1}$$

$$x \otimes y = x + y \tag{5.2}$$

In the max-plus algebra, the regular number 0 is replaced by $\epsilon = -\text{Inf}$. Equation 5.3 shows how this relates to a regular addition of two numbers. The maximum of two numbers of which one is $-\text{Inf}$ always equals the non-infinity number and is thus equal to adding 0. The regular number 1 is replaced by 0 in the max-plus algebra. Equation 5.4 shows how this relates to a regular multiplication of two numbers: multiplying something with one is the same as adding zero to that same value.

$$0 + x = x \qquad \begin{aligned}\epsilon \oplus x &= \max(\epsilon, x)\\ &= x\end{aligned} \tag{5.3}$$

$$1 \cdot x = x \qquad \begin{aligned}0 \otimes x &= 0 + x\\ &= x\end{aligned} \tag{5.4}$$

In max-plus algebra, also matrix multiplications are possible [35]. Equation 5.5 shows how this works. The max operator $\oplus$ is now used to replace the regular addition of multiplied terms in the matrices. The plus operator $\otimes$ replaces the regular multiplication of individual terms. The total matrix multiplication in this case to the maximum of two terms.

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \otimes \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = (x_1 \otimes y_1) \oplus (x_2 \otimes y_2) = \max(x_1 + y_1, x_2 + y_2) \tag{5.5}$$

5.1.1 Max-plus for time-window reservations

To relate the max-plus relations to the reservation system resulting from the context-aware routing algorithm, an example is presented in Figure 5.2. In this example two AGVs are present (red and blue) and both AGVs have two reservations, as shown in the figure. In Equation 5.6 the start and end times (s_i and e_i respectively) for reservation i are given in terms of start and end times of reservations and minimum duration d_i of the reservation. Since these are all expressions which can be written using max-operators and plus-operators, max-plus algebra is suited for this system. Note that in this example s_i and e_i correspond to t_{entry} and t_{leave} in Chapter 4 respectively.

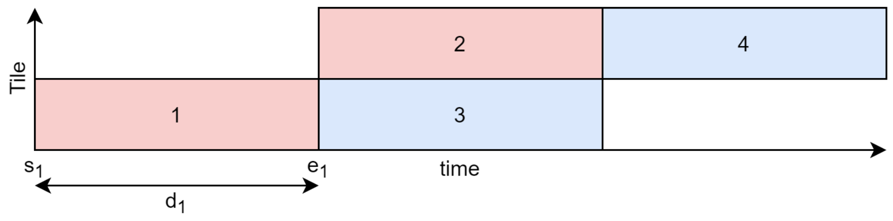


Figure 5.2: Simple reservations example.

$$\begin{aligned}
s_1 &= 0 & e_1 &= s_1 + d_1 = d_1 \otimes s_1\\
s_2 &= e_1 = 0 \otimes e_1 & e_2 &= s_2 + d_2 = d_2 \otimes s_2\\
s_3 &= e_1 = 0 \otimes e_1 & e_3 &= \max(e_2, s_3 + d_3) = (0 \otimes e_2) \oplus (d_3 \otimes s_3)\\
s_4 &= \max(e_2, e_3) = (0 \otimes e_2) \oplus (0 \otimes e_3) & e_4 &= s_4 + d_4 = d_4 \otimes s_4
\end{aligned} \tag{5.6}$$

Using the expressions in Equation 5.6 for the example in Figure 5.2 a max-plus matrix can be built, as shown in Equation 5.7 (matrix A). Note that $\mathbf{x}$ is the state vector with the start and end times of reservations and that $\mathbf{b}$ is a vector with initial information such as the start time of the first (independent) reservation(s). The max-plus matrix gives exactly the same information as the mentioned expressions, but in matrix form. As an example, the expression for e_3, corresponding to the 6^{th} row in the max-plus matrix, can be rewritten as in Equation 5.8 (if there is no initial condition for this value in $\mathbf{b}$). Note that all the values that are equal to $\epsilon = -\text{Inf}$ are not added to this expression, since they do not play a role: taking the maximum of a finite number and $-\infty$ will always return this finite number.

$$\underbrace{\begin{bmatrix} s_1 \\ e_1 \\ s_2 \\ e_2 \\ s_3 \\ e_3 \\ s_4 \\ e_4 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ d_1 & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & 0 & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & d_2 & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & 0 & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & 0 & d_3 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & 0 & \epsilon & 0 & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & d_4 & \epsilon \end{bmatrix}}_{A} \otimes \underbrace{\begin{bmatrix} s_1 \\ e_1 \\ s_2 \\ e_2 \\ s_3 \\ e_3 \\ s_4 \\ e_4 \end{bmatrix}}_{\mathbf{x}} \oplus \mathbf{b} \tag{5.7}$$

$$e_3 = (0 \otimes e_2) \oplus (d_3 \otimes s_3) = \max(e_2, s_3 + d_3) \tag{5.8}$$

5.1.2 Cycles in max-plus for time-window reservations

As became clear in Chapter 4, a next reservation on an AGV does not start as the current one ends. There must be some overlap in the reservations. Therefore the actual situation is not exactly the same as in the example in Figure 5.2, but it looks like the example in Figure 5.3. As an example, reservations 2 and 3 in Figure 5.3 are exactly the same in terms of start -and end time, but are reservations on different tiles.

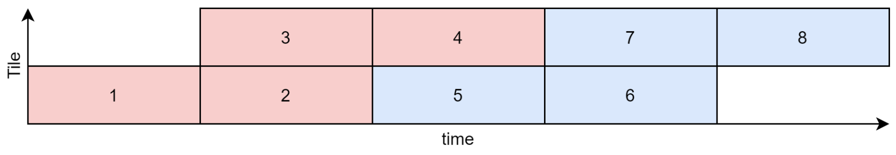


Figure 5.3: Simple reservations example including overlap.

The expressions for the start and end times of the individual reservations, in case of overlapping reservations, can be set up in a similar way as in Equation 5.6. The end values of overlapping reservations are dependent on each other, as becomes clear in Equation 5.9. In this example the end time of reservation 2 is dependent on the end time of reservation 3 and vice versa: a cycle is found. Since in Section 5.2.2 the notion of no cycles in the system is used, there is the need to solve this.

$$\begin{aligned} e_2 &= \max(s_2 + d_2, e_3) \\ e_3 &= \max(s_3 + d_3, e_2) \end{aligned} \tag{5.9}$$

Since it is known that the end times of overlapping reservations must always be the same, a solution to the cyclic referencing problem can be to replace the cyclic expressions with a new one. This new expression for the example in Equation 5.9 is shown in Equation 5.10. Since the start times of overlapping reservations are always the same, and the fact that the duration of overlapping reservations must always be equal, this expression solves the problem. In reality, when updating the duration of reservations, the duration of overlapping reservations is updated equally. Therefore the values d_2 and d_3 are equal. The max-operator is added to make sure the constraints are never violated, even in case the duration of overlapping reservations are not updated equally.

$$e_2 = e_3 = s_2 + \max(d_2, d_3) \tag{5.10}$$

5.2 Solving max-plus linear system

The system identified in Equation 5.7 is a max-plus linear system in the form $\mathbf{x} = A\mathbf{x} \oplus \mathbf{b}$ [37]. To calculate all the start and end times, and thus perform the push-pull operation, it is needed to solve this linear system. In this section first the standard solution method using the Kleene star operator [36] is presented. After this a far cheaper solution method using topological sorting [36] [38] is presented.

5.2.1 Standard solution method

For the linear system $\mathbf{x} = A\mathbf{x} \oplus \mathbf{b}$ a solution is needed in order to calculate the (possibly updated) start and end values of the reservations. According to Chung [37], if a solution $\mathbf{x} = A^*\mathbf{b}$ exists, it is a solution to the linear system. The proof for this is shown in Equation 5.11 [37] using A^* as in Equation 5.13. When performing operations like in Equation 5.12 it is possible to find an expression for A^*.

$$A(A^*\mathbf{b}) \oplus \mathbf{b} = AA^*\mathbf{b} \oplus \mathbf{0b} = (\mathbf{0}AA^*)\mathbf{b} = A^*\mathbf{b} \tag{5.11}$$

One constraint in the method of Chung [37] is that there can only be cycles in the system with non-positive weights. If this is the case, the terms with A as in Equation 5.12 get smaller as k increases. When k is chosen sufficiently large, the last term (including $\mathbf{x}$) can be neglected since it is very small and the solution comes to $\mathbf{x} = A^*\mathbf{b}$ with A^* the Kleene star operator [36] as defined in Equation 5.13.

$$\begin{aligned}
\mathbf{x} =& \mathbf{b} \oplus A\mathbf{x} \\
\mathbf{x} =& \mathbf{b} \oplus A(A\mathbf{x} \oplus \mathbf{b}) \\
\mathbf{x} =& \mathbf{b} \oplus A\mathbf{b} \oplus A^2 x \\
\mathbf{x} =& \mathbf{b} \oplus A\mathbf{b} \oplus ... \oplus A^{k-1}\mathbf{b} \oplus A^k\mathbf{x}
\end{aligned} \tag{5.12}$$

$$A^* = \oplus_{i=0}^{\infty} A^{\oplus i} = \mathbf{0} \oplus A \oplus A^2 \oplus ... \oplus A^{\infty} \tag{5.13}$$

As stated above, the Kleene star method can be used in case there are no cycles with non-positive weights. With the information in Section 5.1.2 the max-plus matrix A used in the context of this project can be cleared from cycles. Therefore this is a special case: the situation in which there are no cycles at all. This means that the above method can be used to calculate the start and end values of reservations, present in $\mathbf{x}$. The Kleene star operator A^* can be used to find a solution $\mathbf{x} = A^*\mathbf{b}$ for the max-plus linear system $\mathbf{x} = A\mathbf{x} \oplus \mathbf{b}$. However, there is a less complex method to solve the special case in which there are no cycles at all. This method is discussed in Section 5.2.2.

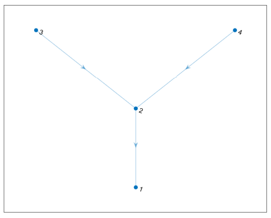
According to Bahalkeh [36] the above mentioned method using the Kleene star requires $\mathcal{O}(2n^4 - 2n^3 + 2n^2)$ operations to calculate the Kleene star operator A^* and another $\mathcal{O}(n)$ operations to then calculate the actual start and end times of the reservations. Here, n is two times the number of reservations (the number of columns of the max-plus matrix), since in $\mathbf{x}$ for every reservation there is a start and end time present.

5.2.2 Topological sorting solution method

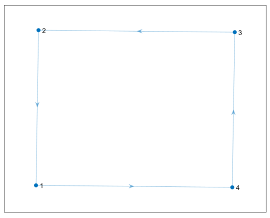
The previously discussed standard solution method using the Kleene star operator can be used to calculate the start -and end values of the reservations in the system. An important note is the fact that the there are no cycles in the max-plus matrix in this context: the graph of the max-plus matrix is a directed acyclic graph (DAG) [36]. Therefore another, cheaper solution method can be used in this context.

This cheaper solution method is called topological sorting [36], introduced first by Kahn [38]. What this means is to sort the variables expressed in a certain way so that newly calculated values are only a function of already determined values. As an example two matrices A and B are shown in Equation 5.14. These matrices can be viewed as max-plus matrices with its −Inf values replaced by 0. When directed graphs are created of these matrices, the figures in Figure 5.4 can be created.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 & 2 \\ 4 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.14}$$



(a) The directed graph of A.



(b) The directed graph of B.

Figure 5.4: Directed graphs of two matrices.

What can be extracted from the directed graph in Figure 5.4a is that the values represented by nodes 3 and 4 can be calculated using the value represented by node 2 and that the value represented by node 2 can be calculated using the value represented by node 1. Node 1 represents an independent variable which is set using initial conditions. There are no cycles so this is a directed acyclic graph (DAG). This graph shows that a topological order for calculating the variables can be for example 1-2-3-4.

The directed graph in Figure 5.4b shows that for example the value represented by node 4 can be calculated using the value represented by node 3, which can be calculated using the value represented by node 2, using the value represented by node 1, which can be calculated using the value represented by node 4: there is a cycle. This is a directed, yet not acyclic graph and therefore no topological order of calculating the values can be determined.

It must be noted that topological ordering results in a possible order of calculation, but this solution is not necessarily unique. In case there are independent variables it might be the case that the topological order is different. In the example of Figure 5.4a another possible order is 1-2-4-3.

Kahn's topological ordering algorithm removes nodes without dependencies one by one and adds them to the topological sorting [38]. When nodes are removed (if it is a DAG), new nodes without dependencies appear until the full order is determined.

There are n nodes in the graph (equal to two times the number of reservations in the system) in the context of this project: one for every start time of a reservation and one for every end time of a reservation. Since every node only must be visited once, performing at most n operations, the number of operations in this algorithm is at most $\mathcal{O}(|V| + |E|)$ [38] in which $V = n$ the number of vertices or nodes and $E = n^2$ the maximum number of edges (if there would be an edge from every node to every other node including itself). This makes the number of operations at most $\mathcal{O}(n^2 + n)$. After this another $\mathcal{O}(n)$ operations are needed to then calculate the actual start and end times of the reservations. Note that this is a lot cheaper than the standard solution method.

5.3 Interaction route planner and push-pull planning

In reality, the path planner and push-pull planning do not work separately. The two models must work together in order to send feasible route information to the AGVs. In this section the interaction between the path planner and push-pull model is discussed.

In general the interaction between the route planner and push-pull planning works as follows. The route planner plans a route and sends (part of) this route to an AGV. Periodically, AGVs send updates on their actual location in the layout. This information is input to the push-pull model. If the planning is now infeasible, based on the information sent by the AGV, the push-pull model updates the reservations such that the planning is feasible again. Using the updated set of routes and reservations, a new route can be planned and/or new information on the location of AGVs can be sent after which the push-pull planning model does its work again.

5.4 Overview push-pull operation

In this chapter so far some things were discussed. To start with the max-plus base for defining relations between reservations was introduced which resulted in a max-plus linear system. Next, a solution for cycles in max-plus relations due to overlapping reservations was presented. After this, the topological sorting solution method for solving the presented max-plus linear system was introduced. Using the above mentioned, the push-pull operation can be executed. In this section the different steps in the push-pull operation are explained.

To start with, it is checked which reservations are currently active. Currently active reservations are reservations that are currently being executed and are found using information on the current location of the AGVs. For the currently active reservations the minimum duration might be updated. This is needed since a reservation that is currently executed cannot have its end time in the past (then this reservation would be infeasible, meaning that the location of the AGV does not match the planning). The set of reservations, (possibly updated) durations of the reservations and set of currently active reservations is fed to the push-pull planning model.

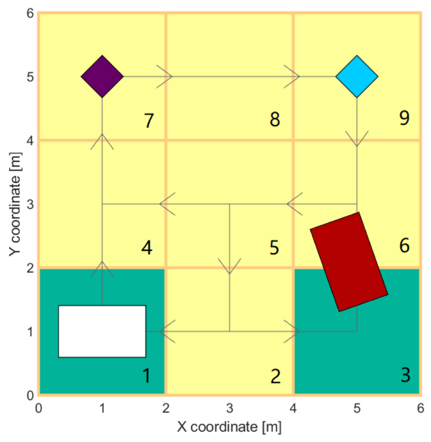
Since all reservations for an AGV that were scheduled before the currently active reservation of an AGV are in the past, these do not have any purpose any more. Therefore in a next step all reservations prior to the currently active reservation per AGV are removed. In case the first reservation for an AGV is the currently active reservation, nothing is deleted.

In the previous step the set of active and future reservations was determined. With this set of reservations and the durations of these reservations, in a similar way as in Section 5.1.1 a max-plus matrix can be created. This matrix is similar to that in Equation 5.6. In a later step this max-plus matrix is used to calculate the possibly updated start -and end times of the reservations.

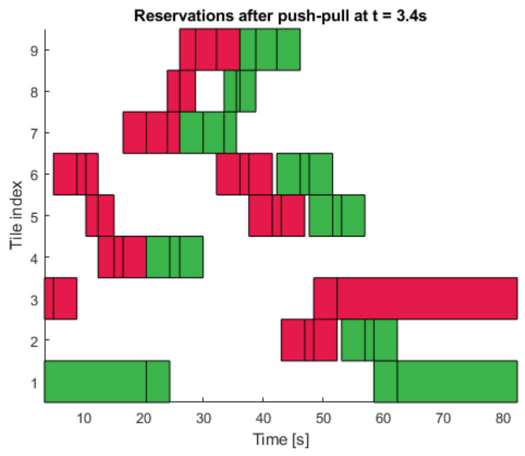
The max-plus matrix created is a square matrix with its size equal to two times the number of reservations (since for every reservation there is a start and end time). For this max-plus matrix it is checked which rows are dependent on which columns. Here, a $-$Inf entry (also called the max-plus zero element) is not regarded as a dependency. Using these dependencies a directed acyclic graph (DAG) as in Figure 5.4 is created. With this directed acyclic graph, the order of calculation is determined using the topological sorting method as described in Section 5.2.2. This step is performed to determine an order in which the start and end times of the reservations can be calculated using only start and end times listed earlier in the order.

For the independent variables, these are the start times of the first reservations per AGV, the initial conditions (current time or other pre-defined time) are used. Using these variables and the max-plus matrix, all start and end times of the reservations are calculated in the order determined using the topological sorting method. The reservations are updated using the (possibly updated) start and end times. By doing this, it is made sure that the set of reservations and thus the planning is feasible again.

Figure 5.5 up and until Figure 5.8 show an example of a push-pull operation. In these figures on the left the layout and state of the system are shown (the tiles are numbered and there are two AGVs in the system). On the right side of the figures the corresponding time-window reservations are shown. Figure 5.5 shows the situation before a push-pull action (thus showing the resulting reservations after the previous push-pull operation). Figure 5.6 shows the situation before finished reservations are deleted. The AGV has moved as can be seen on the left of the figures, but the reservations are still the same as can be seen on the right. Figure 5.7 shows the situation after finished reservations (in this case reservations for tile 3 and 6) are deleted. The changes with respect to the previous situation are encircled in the right figure. Note that this planning is infeasible since for the red AGV there is no active reservation at the current time (which is physically impossible). Figure 5.8 shows the resulting updated reservations after the push-pull operation: the planning is feasible again. The location at which the pull took place is encircled and the start time of the reservations figure is now 4.1s.

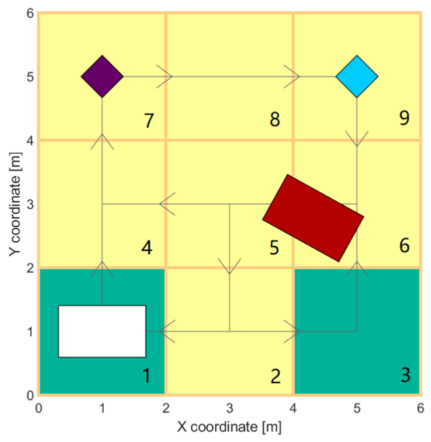


(a) State of the system.

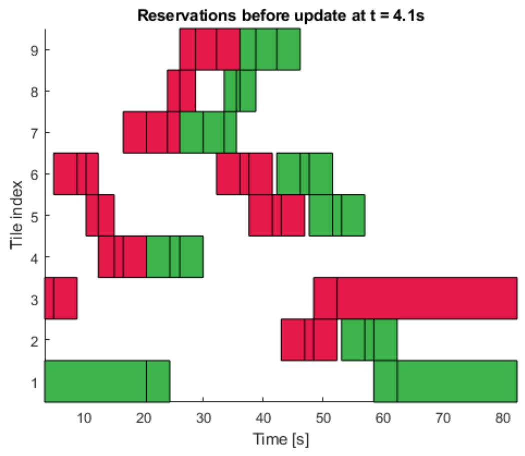


(b) Corresponding reservations.

Figure 5.5: Step 1: situation before the push-pull operation.

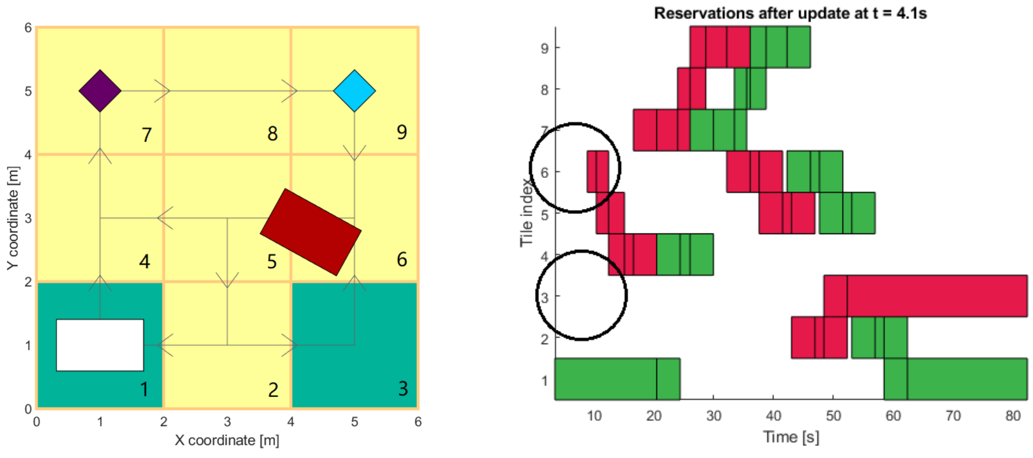


(a) State of the system.



(b) Corresponding reservations.

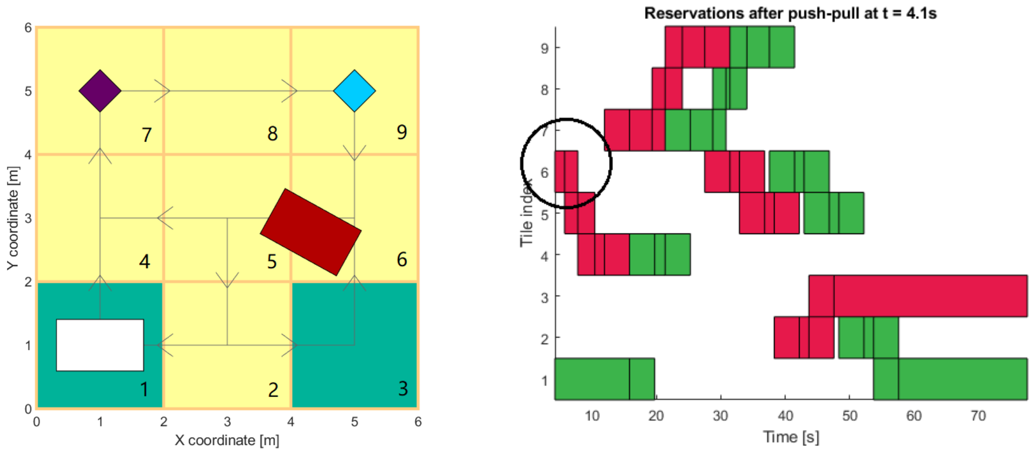
Figure 5.6: Step 2: situation before deleting finished reservations.



(a) State of the system.

(b) Corresponding reservations.

Figure 5.7: Step 3: situation after deleting finished reservations.



(a) State of the system.

(b) Corresponding reservations.

Figure 5.8: Step 4: situation after the push-pull operation.

6. Simulations

In order to compare the context-aware routing strategy to the existing separate path planning and traffic control strategy, simulations were performed in the *AgvSorter* simulation environment. In this Matlab model next to the original separate path planning and traffic control strategy, the context-aware routing strategy was added. In this chapter first some implementation design choices are discussed in Section 6.1. Next, the simulation set-up is presented in Section 6.2 after which the simulation results are shown in Section 6.3. Here, also an interpretation of the results is presented.

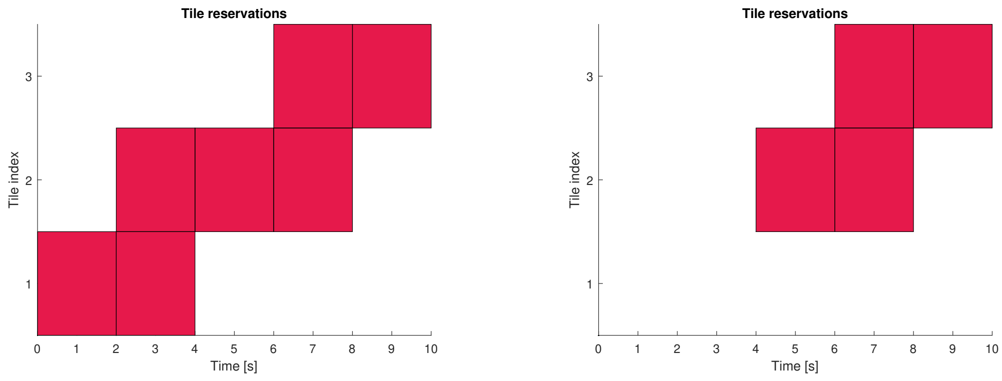
6.1 AgvSorter implementation

The *AgvSorter* simulation model has a discrete event simulation engine. Everything that happens is captured in an event, for example job assignment, AGV movement and so on. In the same way the context-aware routing process is captured in an event. This section describes the main steps taken in such a context-aware routing event. Reservations are deleted and updated if needed, a push-pull action is performed, new jobs are planned, tiles are claimed and in the end the next context-aware routing event is scheduled. The context-aware routing event including the mentioned steps occurs periodically.

6.1.1 Delete and update reservations

The first step in the context-aware routing event is to delete reservations that are finished. This is the case when an AGV leaves a tile. Figure 6.1 can be used to explain how this works. When an AGV leaves a tile, the AGV agent sends a signal to the context-aware routing mechanism. In Figure 6.1a the reservations before a signal that the AGV left tile 1 are shown, whereas Figure 6.1b shows the reservations after the processing of the left-tile signal. Note that the reservations on tile 1 are removed, including overlapping reservations (in the example situation this is a reservation on tile 2).

After the finished reservations are removed, reservations are updated in case they are delayed. For this it is necessary to look at the reservations after processing left tile signals (meaning that only reservations which are still active or planned in the future remain). In case there are reservations in this set with end time lower than the current time, the minimum duration is updated to make sure the end time of that reservation is equal to the current time. By doing so, these reservations are made feasible again: a reservation can only be present if it is currently executed or planned in the future, it cannot be in the past. Any violations in the relations with other reservations that the update in minimum durations causes, are solved in the push-pull operation.



(a) Original set of reservations. (b) Reservations after a left tile trigger.

Figure 6.1: Reservations before and after an update.

6.1.2 Perform push-pull operation

In this step of the context-aware routing event, the push-pull operation as described in Chapter 5 is performed. A pull is performed in case the start of the first reservation of an AGV is later than the current time. This is due to the fact that an AGV cannot just disappear until the time that there is a reservation: in reality the first reservation for the AGV already started. A push is performed in case the end time of the first reservation for an AGV is earlier than the current time. Note that a push-pull operation is one single action (no separate actions for push and pull), as described in Chapter 5.

6.1.3 Check for new jobs and plan new jobs

Once the push-pull operation is performed, it is checked whether there are new jobs that must be planned. When a job enters the system, the AGV agent sends a signal to the context-aware routing mechanism. The job is then placed in the job queue for the AGV. There is a job queue per AGV and the jobs in this queue are processed in a first come first serve (FCFS) manner.

It can be the case that not all jobs in the job queue are processed immediately. There is a maximum allowed number of jobs to be planned ahead to reduce complexity. Therefore it can be the case that a job remains in the queue, although a context-aware routing event occurs. When the job queue for an AGV is not empty and the maximum number of jobs to plan ahead is not reached, a new route is planned using the context-aware routing algorithm as described in Chapter 4. When a job is planned, it is removed from the job queue.

6.1.4 Claim tiles

After the step in which jobs might or might not be planned, tiles can be claimed. Claiming a tile in this context means that an AGV gets the right to drive on a certain tile. If a tile is not claimed, an AGV is not allowed to drive on that tile. The difference between making reservations and claiming is that making a reservation means to plan a time window in which an AGV is expected to occupy a certain tile (and reservations are therefore made for all tiles in the path before execution), whereas claiming tiles happens only for some tiles in the path ahead and happens during execution.

Constraints on claiming tiles are that a tile can only be claimed by one AGV at a time and AGVs can only claim consecutive tiles in its route (starting at the tile(s) at which the AGV is currently located) Moreover, the order of events, in this context the order of AGVs passing a certain tile and the order of tiles in AGV routes as set in the routing algorithm, must stay the same. The order of events must stay the same since this guarantees feasible execution without deadlocks. Another constraint is that only one AGV can claim a tile at once, preventing collisions to occur.

Claiming tiles happens as follows. An AGV is allowed to claim a fixed number of tiles in its route ahead, starting at the first tile in its route. Since the routes are planned in time, also a time-based condition was added to this. If possible, an AGV is allowed to claim the tiles in its route that (in the planning) start between the current moment in time and a fixed step in time ahead. In total this claiming strategy means that at least one tile is claimed (the tile at which the AGV is currently located) and at most the maximum of a fixed number and the number of tiles in its route between the current time and a fixed time step ahead. Similar to claiming tiles, tiles are released in the *AgvSorter* model as soon as there is a left tile signal.

6.1.5 Set next event

The last step in the context-aware routing event is to set the next context-aware routing event. Since this event happens periodically, it is scheduled again for a fixed time ahead. The reason why the event must occur periodically is that jobs that come in must be processed using the context-aware routing algorithm, disturbances must be handled using the push-pull mechanism and tile reservations and claims must be managed. This all in order to keep the planning and execution feasible.

6.2 Simulation set-up

In this section the simulation set-up is discussed. In order to perform simulations and compare results, three different layouts where chosen. Two of these layouts are airport application layouts and one is a parcel sorting layout. First an introduction to the layouts is presented after which the input parameters including of vehicles simulated per layout (simulation points), the simulation time per replication and number of replications per simulation point are discussed.

6.2.1 Layouts

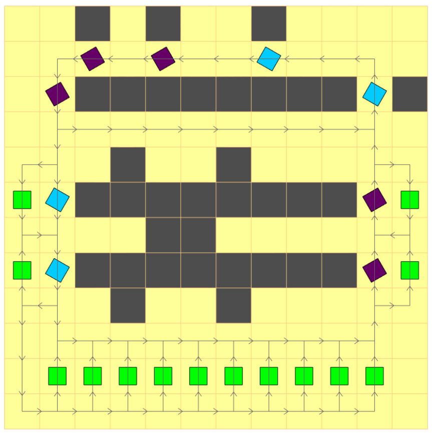
The three layouts used are introduces in Table 6.1. Here the layout size, number of drivable tiles, number of edges, number of pick-up points, number of drop-off points and number of parking spots are first introduced. The three layouts were chosen since these are capable of handling different number of vehicles (12, 20 and 36 respectively) and have different AGV interaction.

The first two layouts, in the remainder of this report referred to as layout 1 and layout 2 respectively, resemble airport terminals. Jobs (bags in this application) must be picked up at conveyors and dropped off at other conveyors, making the number of pick-up points approximately equal to the number of drop-off points. Layout 2 is bigger and more vehicles can be used in this layout.

The third layout, in the remainder of this report referred to as layout 3, resembles a parcel sorting facility and is shown in Figure 6.3a. There are some pick-up points and many drop-off points to sort the packages. Figure 6.3b shows the same layout but with bidirectional lanes. For the bidirectional situation few simulations were performed due to the fact that the *AgvSorter* model was not built for bidirectional lanes and therefore many simulation errors came up. Some simulations were performed using the bidirectional layout to show that the context-aware routing algorithm can handle this.

Table 6.1: Information on the simulation layouts.

Layout name	Layout 1	Layout 2	Layout 3
Layout size	12x12	28x28	18x18
Number of drivable tiles	75	306	296
Number of edges	90	393	424
Number of pick-up points	5	11	8
Number of drop-off points	4	11	36
Number of parking spots	14	22	36


(a) Layout 1.

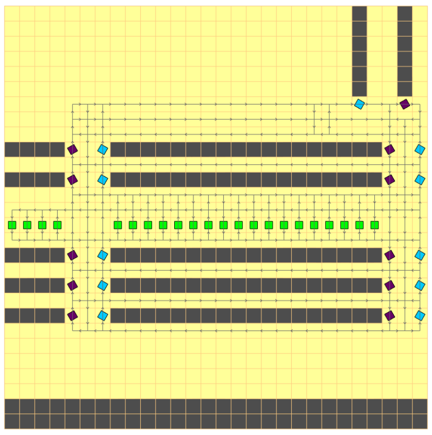
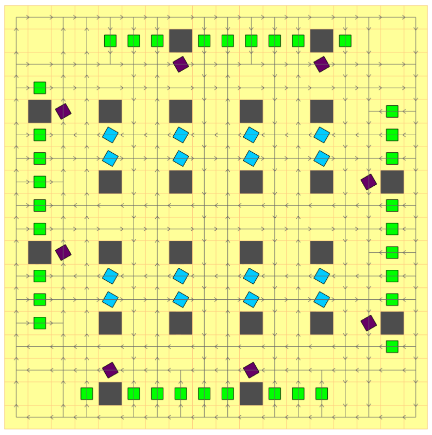

(b) Layout 2.

Figure 6.2: Two different layouts.


(a) Layout 3, unidirectional.

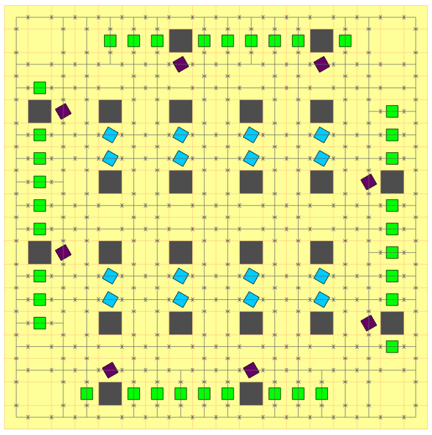

(b) Layout 3, bidirectional.

Figure 6.3: Layout 3.

6.2.2 Input parameters

In the mentioned layouts the tiles are squares with sides of size 2.08 m and the AGVs are squares with size 1.370 m by 0.8118 m. In every simulation the following input parameters were taken the same. The maximum drive speed equals 2.5 m/s, the rotational speed equals 1.05 rad/s (or a 90 degree turn in 1.5 s), the acceleration equals 2.5 m/s^2 and the deceleration equals -1.35 m/s^2. Using these input parameters and the equations in Section 4.4 the model can calculate the costs and therefore the routes for the AGVs.

Next to some layout and AGV-specific input parameters there are some simulation input parameters. The cycle interval, which is the time between two context-aware routing events, equals 0.5 s for all simulations. For the claim action in the context-aware routing event, as discussed in Section 6.1.4, the time to claim ahead is 6 s and the number of tiles to claim ahead is 4. For every replication different load files were used to make sure every simulation is unique and mean results can be created. A load file is a set of pick-up and drop-off combinations which represent the jobs that must be executed.

The simulation points, meaning the number of AGVs that were simulated, are based on earlier simulations performed using the separate path planning and traffic control approach (from now on referred to as classic grid-based control or classic GBC). A simulation time of 3600 s per replication and 10 replications per simulation point gave results (using different load files) which were comparable, therefore this was taken as the standard. Due to some simulation errors and complexity issues, not all simulation points used in the classic GBC simulations could be performed. Also not in all cases the simulation time of 3600 s and number of replications of 10 could be reached for the same reasons. Therefore some changes to the above mentioned simulation set-up were applied. To give an overview, Table 6.2 up and until Table 6.5 show the number of AGVs, number of replications and simulation time for the performed simulations for layout 1, layout 2, layout 3 (unidirectional) and layout 3 (bidirectional) respectively.

Table 6.2: Simulation information for layout 1.

Number of AGVs	1:12
Number of replications	10
Simulation time [s]	3600

Table 6.3: Simulation information for layout 2.

Number of AGVs	1:10	12	14	15
Number of replications	4	4	4	4
Simulation time [s]	3600	2000	2000	2000

Table 6.4: Simulation information for layout 3 (unidirectional).

Number of AGVs	1:10	12	15	18	21	24	27
Number of replications	3	3	3	3	3	3	3
Simulation time [s]	3600	3600	3600	3600	3600	3600	2000

Table 6.5: Simulation information for layout 3 (bidirectional).

Number of AGVs	1:9	13
Number of replications	1	1
Simulation time [s]	3600	2000

For the simulations using the classic grid-base control (classic GBC) strategy, results of simulations that were already performed within Vanderlande were used. The numbers of vehicles per layout (simulation points) are shown in the result figures. For every simulation point, 1 replication of 3600 s simulation time is present.

6.3 Simulation results

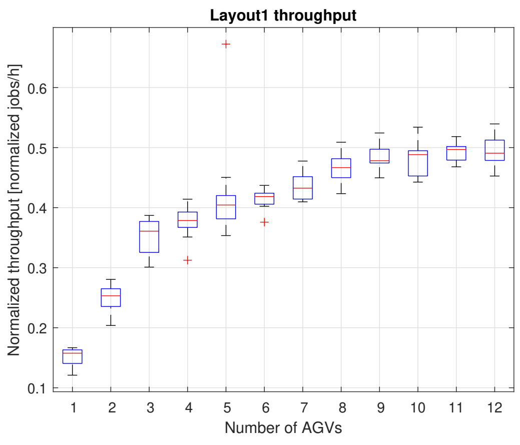
The simulation results are discussed based on three key performance indicators (KPIs): mean throughput, mean occupancy and mean item lead time. Throughput is the mean number of jobs handled per hour, mean occupancy the mean percentage of time that an AGV was loaded and mean item lead time is the mean time it took AGVs from pick-up to drop-off. These are no separate indicators. For example a lower item lead time can cause the throughput to be higher.

The results presented in this section do not necessarily say something about real-life performance. However, the simulation results are used to give an indication on how the context-aware routing algorithm performs with respect to the classic separate path planning and traffic control (classic GBC) approach. It must be noted that the data in this report was normalized as the real data is confidential.

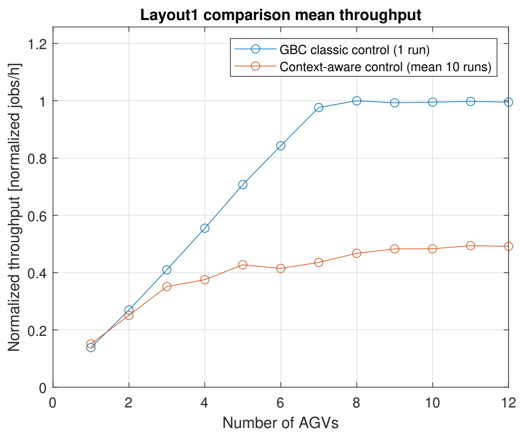
In the next part the results are discussed. First the simulation results are presented per layout. A comparison between the simulation results using classic GBC and context-aware control is made after which the results are interpreted.

6.3.1 Layout 1

In Figure 6.4a a box plot [39] of the normalized throughput using the context-aware routing algorithm is shown. For the ten replications per simulated number of AGVs the dispersion in results is visible here with the red line being the median. Figure 6.4b shows the same (mean) results, but simulation results using the classic GBC are also presented here. For higher number of vehicles the throughput of the context-aware method is approximately half of the throughput using classic GBC control.



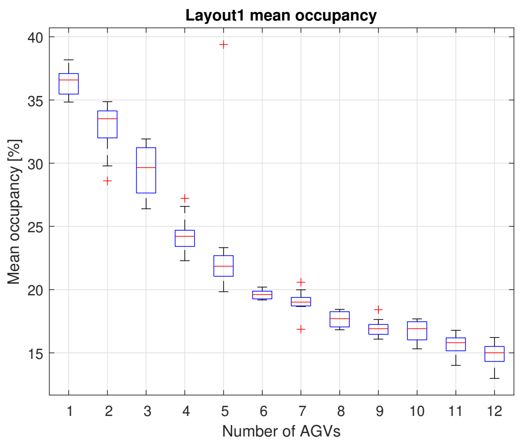
(a) Box plot of the normalized throughput.



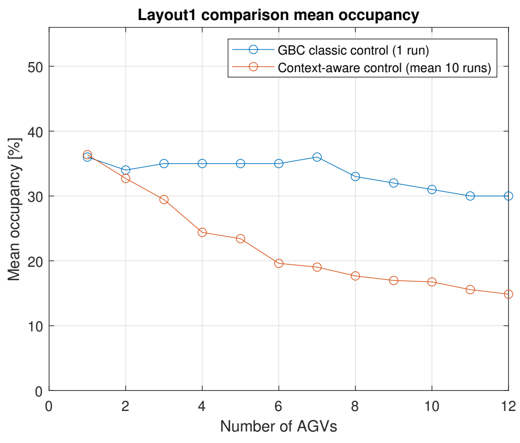
(b) Comparison of the normalized throughput.

Figure 6.4: Normalized throughput results for layout 1.

The resulting mean occupancy results are shown in a box plot in Figure 6.5a. Here the dispersion in the simulation results is shown. In Figure 6.5 the mean occupancy is compared to the classic GBC simulation results. The mean occupancy of the AGVs (for higher number of vehicles) is approximately half of that in the classic GBC case. This results indicate that in the context-aware control case vehicles are delayed when not loaded, causing the mean occupancy to go down. It also corresponds to the throughput results since when more time is needed for driving unloaded, there is less time to handle jobs which causes the throughput to go down.



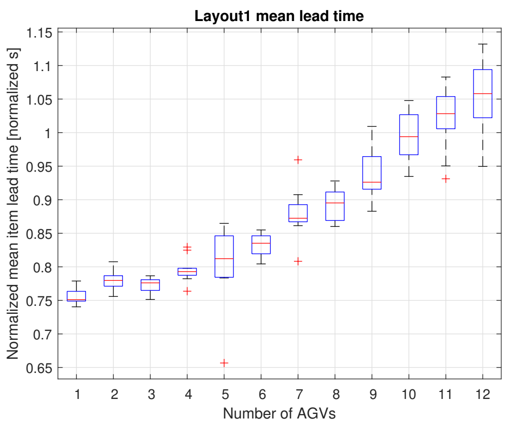
(a) Box plot of the mean occupancy.



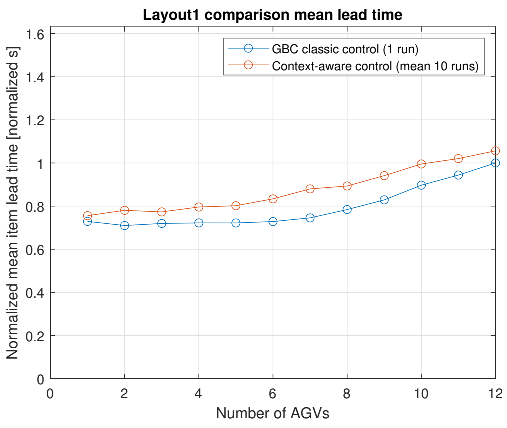
(b) Comparison of the mean occupancy.

Figure 6.5: Mean occupancy results for layout 1.

The normalized mean item lead time results are shown in a box plot in Figure 6.6a. Again here the dispersion in simulation results is shown. Figure 6.6b shows the comparison with the classic GBC mean item lead time results. In contrast with the earlier results, the mean item lead time is closer to that in the classic GBC case. This supports the above mentioned claim that vehicles are delayed when not loaded. In Section 6.3.4 the reasons for this are discussed.



(a) Box plot of the normalized mean item lead time.



(b) Comparison of the normalized mean item lead time.

Figure 6.6: Normalized mean item lead time results for layout 1.

6.3.2 Layout 2

Due to simulation and complexity issues, only the simulations indicated in Table 6.3 were performed for layout 2. This means that there are only 4 replications per number of AGVs. For a box plot, more data points are needed to give interesting results and therefore no box plots are presented for the simulation results of layout 2. Recommendations to solve the simulation and complexity issues are presented in Section 7.2.

In Figure 6.7 the normalized throughput results for the context-aware control simulations are presented together with the results using classic GBC. Similar to layout 1, for higher number of vehicles the throughput is lower for the context-aware control simulations. However, the relative difference between context-aware control and classic GBC seems to be smaller than for layout 1: for the context-aware control situation the maximum throughput in layout 1 is approximately 0.5 times that of the classic GBC situation, wheras in layout 2 it looks like this is little under 0.7.

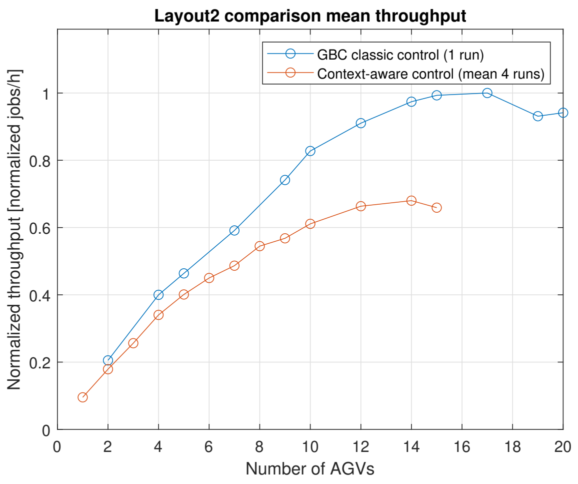


Figure 6.7: Normalized throughput results for layout 2.

Mean occupancy simulation results for both control methods are presented in Figure 6.8. As was the case for layout 1, the mean occupancy is lower using context-aware control rather than classic GBC. This results supports the claim that in the context-aware control case, vehicles are delayed when not loaded (causing the mean occupancy to go down). This also corresponds with the throughput results: when vehicles spend more time in an unloaded condition there is less time to execute jobs, hence the throughput is lower.

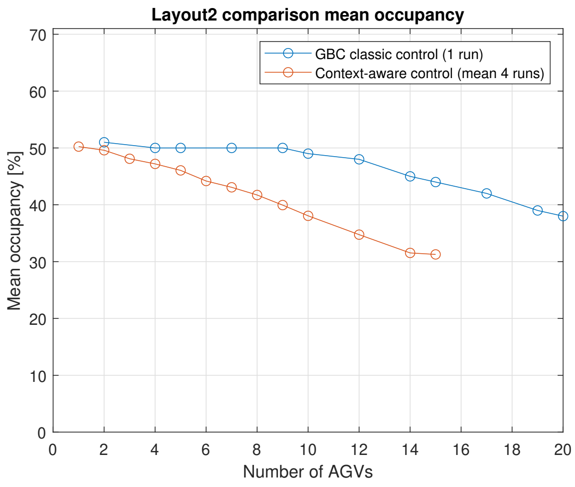


Figure 6.8: Mean occupancy results for layout 2.

The results for the normalized mean item lead time are presented in Figure 6.9. The results are comparable to those for layout 1: the mean item lead time using context-aware control is somewhat higher than for classic GBC, but much closer than the throughput and mean occupancy. The results combined indicate that once loaded, jobs are executed approximately as fast as in the classic GBC situation, but vehicles are more delayed when not loaded. More discussion on this is presented in Section 6.3.4.

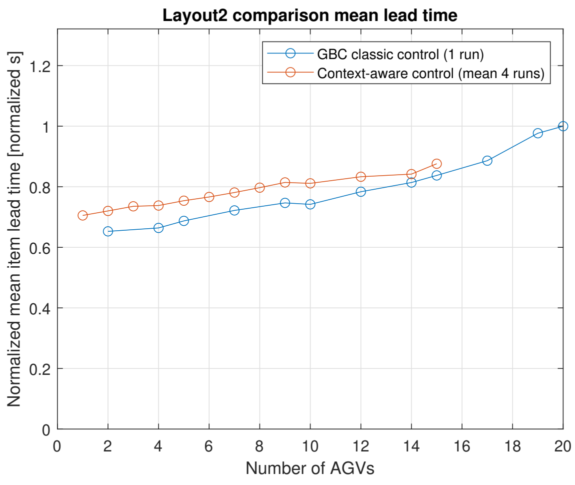


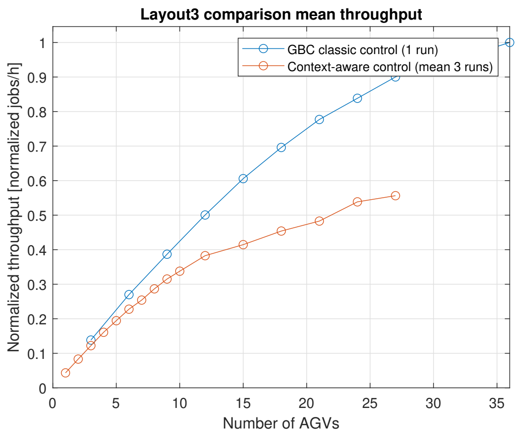
Figure 6.9: Normalized mean item lead time results for layout 2.

6.3.3 Layout 3

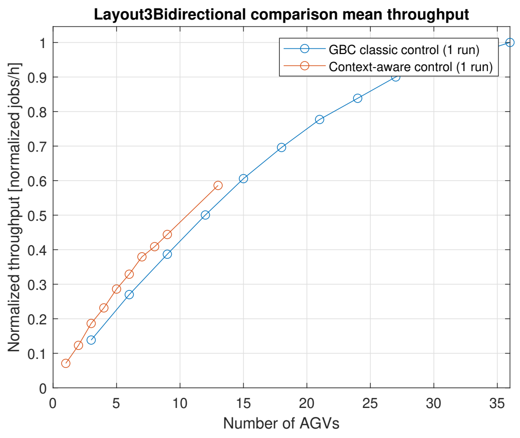
For layout 3, the simulation points as indicated in Table 6.4 (unidirectional) and Table 6.5 (bidirectional) were performed. As mentioned, model complexity issues (on which some recommendations are discussed in Section 7.2) mainly cause the fact that less simulations are performed for this layout (hence no box plots are presented). In the bidirectional layout an additional issue is that the *AgvSorter* model is not yet capable of handling bidirectional layouts that good. It must be noted that the results for the bidirectional layout are all based on one simulation per number of AGVs. Therefore these results must be viewed as an indication rather than a complete result.

In Figure 6.10a the throughput results for layout 3 simulations (unidirectional) using context-aware control are compared to the results using the classic GBC control. Similar to the results for layout 1 and layout 2, for higher numbers of AGVs the throughput is lower using the context-aware control (the maximum throughput seems to be at around 0.6 times the maximum throughput in the classic GBC situation).

The bidirectional throughput results in Figure 6.10b show that the throughput (as far as the simulations go) is higher using context-aware control rather than classic GBC. This means that there is a positive effect on the performance when bidirectional lanes are used (note: this is an indication since only one simulation per number of AGVs was performed).



(a) Normalized throughput results, unidirectional.

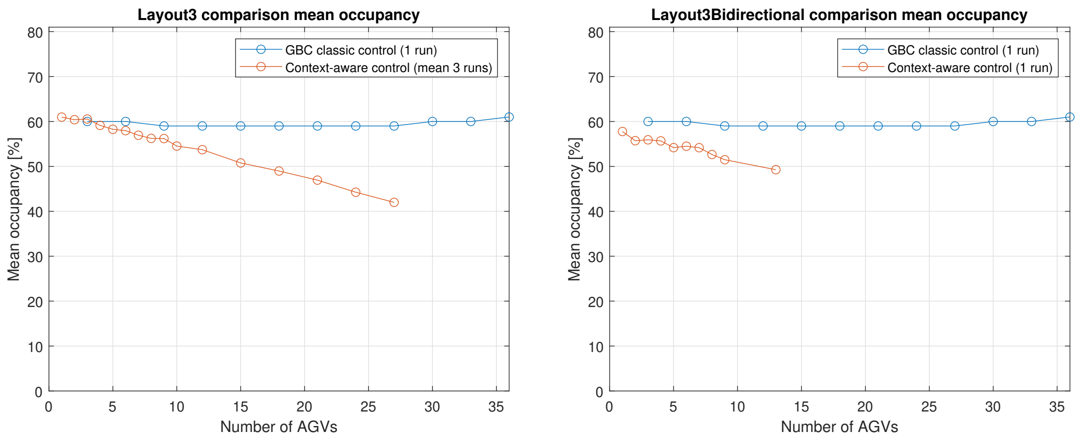


(b) Normalized throughput results, bidirectional.

Figure 6.10: Normalized throughput results for layout 3, unidirectional and bidirectional.

The mean occupancy results for layout 3 (unidirectional) are presented in Figure 6.11a. As discussed in the other simulation results, when having higher numbers of vehicles the mean occupancy is lower using context-aware control rather than classic GBC. This again indicates that vehicles are delayed when not loaded.

Figure 6.11b shows the mean occupancy results for layout 3 (bidirectional) compared to the classic GBC results. Whereas the throughput (for the performed simulations) was higher using context-aware control, the mean occupancy is still lower than in the classic GBC situation. This indicates that the positive effect of using bidirectional lanes (possibly shorter paths) causes the throughput to be higher. However, this result shows that also when using bidirectional lanes, vehicles are delayed more (compared to the classic GBC situation) when not loaded: an observation that was also done in the results of the other layouts.

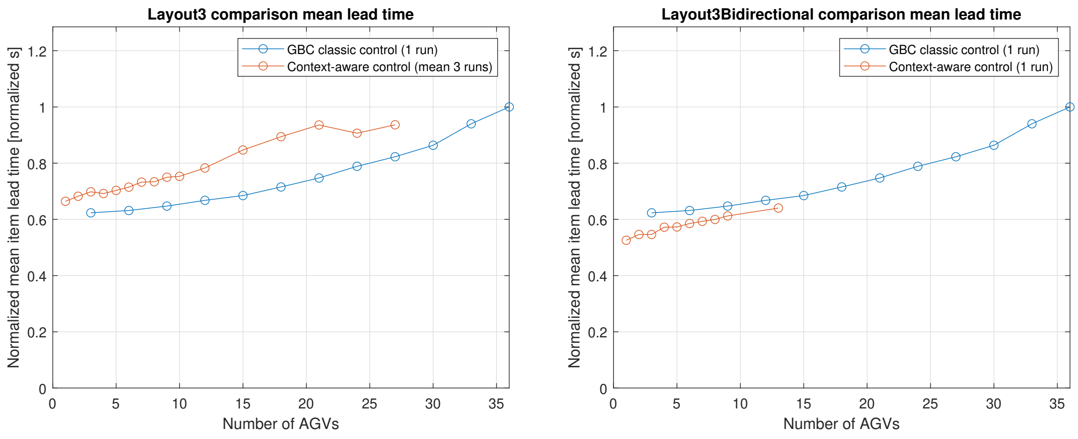


(a) Mean occupancy results, unidirectional.
(b) Mean occupancy results, bidirectional.

Figure 6.11: Mean occupancy results for layout 3, unidirectional and bidirectional.

Mean item lead time results for layout 3 (unidirectional) are shown in Figure 6.12a. The mean item lead time using context-aware routing is higher than when using classic GBC. This matches with the throughput and mean occupancy results. Furthermore, the fact that the mean item lead time using context-aware control versus classic GBC is closer than the other performance indicators is something that was observed in layout 1 and layout 2 too.

In the mean item lead time results for layout 3 (bidirectional) in Figure 6.12b it is shown that (for the executed simulations) the mean item lead time is lower using context-aware control compared to classic GBC. This is explainable: a lower item lead time causes higher throughput, something that was observed earlier. What can also be recognized is that the throughput, mean occupancy and mean item lead time results show that using bidirectional lanes reduces the mean item lead time in such a way that the throughput goes up, but that the vehicles are still delayed when not loaded (lower mean occupancy).



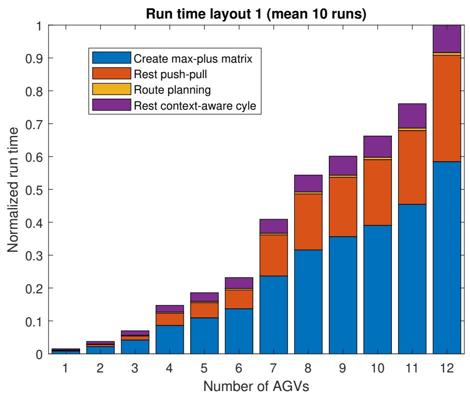
(a) Mean lead time results, unidirectional.
(b) Mean lead time results, bidirectional.

Figure 6.12: Mean lead time results for layout 3, unidirectional and bidirectional.

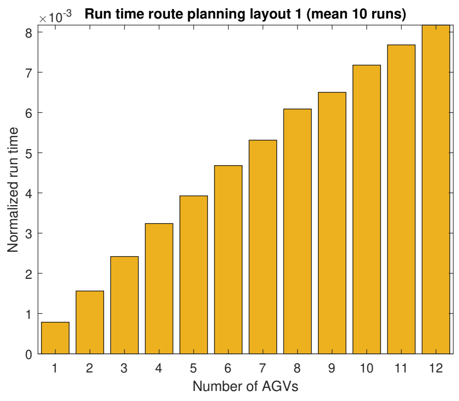
6.3.4 Result interpretation

For the context-aware control simulations of layout 2 and layout 3 it was not possible to simulate exactly the same number of AGVs as for classic GBC. This is mainly due to implementation, simulation and complexity issues. Some recommendations on the mentioned issues are presented in Section 7.2. However, there are trends visible in all of the results and therefore in this section some particularities regarding these results are discussed.

To get an insight in the computation time issues, in Figure 6.13 the division in computation time for layout 1 is presented. Here the subdivision of the total mean computation time among the different partsin the context-aware algorithm for ten runs per number of AGVs is shown. In Figure 6.13a is shown that the total computation time increases (it looks like exponential growth) when the number of AGVs becomes higher. Also it becomes clear that most of the computation time is used by the push-pull mechanism, and in specific the creation of the max-plus matrix. Figure 6.13b shows that for layout 1 the computation time for route planning is small and seems to grow approximately linear with increasing vehicle numbers. The information in this figure is also shown in Figure 6.13a from which it can be observed that the time for planning routes is small compared to the total computation time. Other costs that are shown in Figure 6.13a are computation time for the rest of the push-pull planning (for example calculating the new start and end times, topological sorting and so on) and computation time for the rest of the context-aware cycle (here the computation time for for example removing and updating reservations and claiming tiles are included).



(a) Normalized mean division in computation time.



(b) Normalized mean routing time.

Figure 6.13: Normalized computation time results for layout 1.

From the context-aware simulations, but also from the classic GBC simulations, it becomes clear that more vehicles does not necessarily mean better performance. This is due to the fact that in case the grid gets more dense (more vehicles per m^2), the AGVs spend more time waiting for each other. At some point an optimum is reached after which the throughput does not decrease (it might even decrease) in combination with the item lead time going up.

From the throughput results in Figure 6.4b, Figure 6.7 and Figure 6.10a it seems that the optimum in throughput is lower using context-aware control rather than classic GBC. When a new route is planned, first the context-aware routing algorithm tries to plan a route from the first unclaimed tile on. When the grid is dense (meaning there are more AGVs per m^2) there are few free time windows on the tiles making the chance that there are no reachable free time windows available on the successor tile larger. When no route is found, the algorithm plans a route from the parking tile on, from which always a route can be found since the reservation on the parking tile is until infinity. Driving to a parking spot before going to the pick-up location of the next job mostly causes delays, which is visible in the simulation results in the form of lower throughput. Since an AGV is always not loaded when driving to a parking spot, the part of the route where the delay occurs, this effect is also visible in the mean occupancy results from the simulations: the mean occupancy is lower. This effect is only visible when AGVs are not loaded so there is no or little effect on the item lead time. This is also visible in the simulation results: the mean item lead time using context-aware control is close to that using classic GBC. A possible solution for the mentioned issue is presented in Section 7.2.

Another reason for the throughput being lower and the mean item lead time being higher using context-aware control versus classic GBC is the nature of context-aware routing. When making a planning, the routes are set and thus the order of AGVs passing a certain tile is set. When an AGV is delayed, all other AGVs having to pass the tiles in the route of that AGV might be delayed. Figure 6.14 shows this problem. Here, the route of the encircled lower AGV is shown in orange. Since this AGV is the first (in the planning) allowed to pass the lowest horizontal row of orange tiles in its route, all the white AGVs must wait until this AGV has passed, causing delays and thus lower throughput. Note that also the problem discussed in the previous paragraph is visible in the figure: the white vehicles must wait due to the fact that they are on a parking spot. In Section 7.2 an idea for future work to solve this issue is presented.

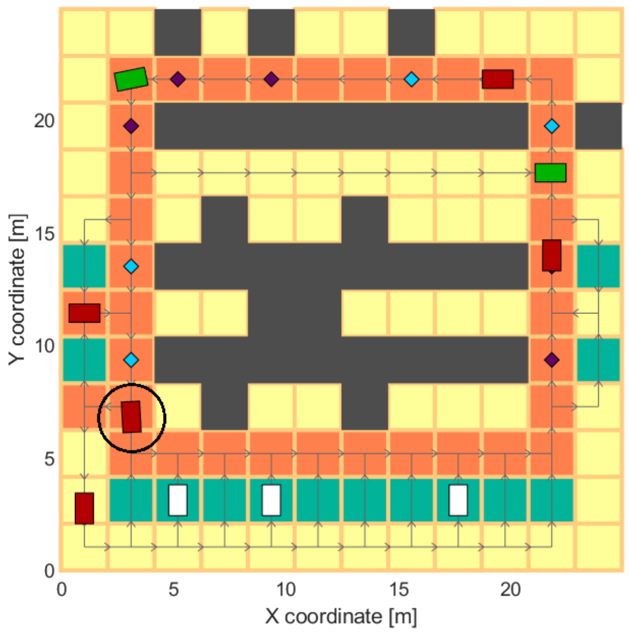


Figure 6.14: Delayed AGV causing other AGVs to wait.

An observation that can be made based on the simulation results is that for lower numbers of vehicles, the context-aware control results resemble the classic GBC results better than for high numbers of vehicles. For lower numbers of vehicles there is less interaction between AGVs and, as mentioned in earlier paragraphs, this interaction is in context-aware control the cause for AGVs going back to its parking and thus delays. Therefore the problems discussed in the previous paragraphs are not a big problem when having low numbers of vehicles, but are a problem when using many vehicles.

Another observation is that the layout influences the performance. For example: in layout 1 the maximum throughput using context-aware control was around 0.5 times the maximum reached throughput using classic GBC whereas for layout 2 this was around 0.7. The reason for this is expected to be in the fact that for layout 2 there are more options to drive from one location to another than in layout 1 (more possible routes) which reduces the interaction and thus delay of AGVs. A further study on the influence of different layout (parts) is not within the scope of this research and therefore discussed in the recommendations in Section 7.2.

Especially for layout 2 there were some problems in the simulations using higher numbers of vehicles. The problem here is that in case there are many reservations on certain tiles, the algorithm is not able to find solutions (not even from the parking spot on). As discussed in Chapter 4 when using the context-aware routing algorithm, planning a route from the parking spot on should always be possible since in the most extreme case an AGV can plan its route infinity time ahead. The reason why an error occurs is most likely an implementation mistake, on which recommendations are made in Section 7.2.

For layout 3 also simulations using bidirectional lanes were performed. Although it was not possible to simulate higher numbers of vehicles, the first results look promising. Despite all the above mentioned issues and shortcomings, the throughput seems to be even higher using context-aware control rather than classic GBC. Also the mean item lead time is lower using context-aware control. The fact that the earlier in this section mentioned issues (for example driving to parking spots before starting new jobs) are also present here is visible in the lower mean occupancy using context-aware control rather than classic GBC. Note that these results are based on single simulations per number of vehicles and that therefore no conclusions can be drawn from this. The upper mentioned points are therefore just an indication that bidirectional lanes might be interesting to look at.

7. Conclusions and recommendations

The project described in this report is a research in combining path planning and traffic control in such a way that routing can be performed in the *AgvSorter* system: an item sorting model developed within Vanderlande. In the research the goal was to design a real-time routing strategy for the *AgvSorter* system in which path planning and traffic control are combined while being able to deal with idle vehicles and disturbances. In this chapter first the main conclusions are discussed after which a number of recommendations are presented.

7.1 Conclusions

To reach the above mentioned goal first a literature study was performed on different approaches to combine path planning and traffic control in a routing strategy. The context-aware routing approach as proposed by Ter Mors, Zutt and Witteveen [1] seemed to be the most suited approach for reaching this goal. In this context-aware approach the path planning is performed by searching for reachable time windows on consecutive tiles in order to find a path from source to destination. In an ideal world the resulting planning is deadlock-free. In reality, for execution some extra steps (push-pull planning) are needed. In the original separate path planning and traffic control approach it is almost impossible to use bidirectional layouts since in execution this would quickly result in deadlocks. The context-aware approach in combination with a push-pull planning is able to handle bidirectional layouts. The routing algorithm works for bidirectional layouts, but some adjustments must be made to the *AgvSorter* simulation model in order to handle bidirectional layouts without problems.

In this project a new version of the original context-aware routing algorithm of Ter Mors, Zutt and Witteveen [1] was created. In this version cost estimations are used to determine the estimated time needed for time window reservations on different tiles in the paths. For example costs for moving, turning, (un)loading, accelerating and decelerating are included. In order to deal with idle vehicles (vehicles that finished a job and have no new job yet) and to make sure always a route can be found, parking spots were be added in the layouts. A parking spot is the only location at which AGVs can in theory stay until infinity. Routes for AGVs are always planned from current tile to pick-up tile to drop-off tile to parking tile. This prevents AGVs from blocking drop-off (or other) tiles while waiting for new route information. In order to make sure routing goes faster, heuristic costs were added.

The context-aware routing algorithm using estimated costs is (as the word indicates) an estimation and is per definition not exact. Next to this, in reality AGVs might behave different than expected causing delays in reservations (or shorter reservations than expected) and thus a mismatch in planning and reality. Therefore a push-pull mechanism was designed to be able to deal with disturbances in the system. This push-pull mechanism uses max-plus algebra to calculate (possibly) new start and end times for time window reservations on tiles. Time window reservations can be pushed forward in case AGVs are delayed and thus reservations take longer than expected, or can be pulled backwards when reservations are finished earlier than expected. The push-pull mechanism is used to keep the planning with time-window reservations feasible and real-time execution possible.

For three different layouts simulations were performed. The performance achieved using the context-aware algorithm is worse than using the separate path planning and traffic control approach. The throughput is for all layouts lower and the item lead time higher (the difference is bigger for a higher number of vehicles) using context-ware control. These results can be explained by the fact that when using context-aware control in dense grids it is harder to plan routes directly from a tile in the route of an AGV (since then less time windows are available and time windows might not be big enough). This causes AGVs to drive to their parking spot first, which often takes more time (hence lower throughput). Another reason for this is that delayed AGVs often cause other AGVs to be delayed since the order of AGVs allowed to pass a certain tile is determined by the context-aware routing algorithm before execution and kept the same during execution (despite disturbances and unexpected behaviour). Although the *AgvSorter* model is not fully ready for bidirectional layout use, first simulations show some promising results: higher throughput and lower item lead time (even with the above mentioned delay issues). Recommendations for more research on the upper mentioned topics are presented in Section 7.2.

This research shows that it is possible to combine path planning and traffic control in a routing approach using context-aware routing. Dealing with idle vehicles can be done using parking spots and disturbances are handled by a push-pull mechanism. Although the simulation results show that algorithm at this point performs worse than the classic separate path planning and traffic control approach there are some promising options to further develop the context-aware algorithm.

7.2 Recommendations

In this section a number of recommendations to improve the work in this project and ideas for future research are discussed. Recommendations on improvements in the implementation are presented in Appendix B. The points discussed there are not necessarily design recommendations but recommendations in the implementation used for creating the results in this project..

As discussed in Section 4.8 there is a difference in the estimated costs and actual execution costs. In order to be able to perform better route planning these costs can be made more exact, meaning that the planning and execution are more alike. Another point to look at in the time window reservations is the fact that setting a fixed cycle interval for the context-aware routing event makes it almost impossible for reservations to start and end exactly after each other. This is due to the fact that claiming tiles is not a continuous process, but this happens once every cycle. For example when the next tile in the route of an AGV is not claimed but already free, the AGV must wait until the next context-aware cycle before the tile can be claimed and driving is allowed. For this it might be an idea to release and claim tiles separately from the context-aware routing cycle.

Also shown in Section 4.8 is that for square tiles it is rather easy to calculate the exact overlap time for two consecutive time windows (on two consecutive tiles, using constant speed and acceleration/deceleration). However, to generalize this to other tile shapes is not that straightforward. The cost calculation using different tile shapes might be topic of future research. In a broader perspective: topic of future research might be to generalize the whole context-aware routing algorithm for different tile shapes. Starting points can be reserving multiple tiles at the same time (due to for example AGVs being bigger than the tile size) and push-pull planning in case of multiple reservations at the same (handling relations between reservations in this).

Scalability is one of the main issues in the proposed context-aware routing approach. When using larger layouts and higher numbers of vehicles, the approach gets more complex. This has multiple reasons. Firstly, routing takes more time since when having larger systems there are more occupied and free time windows through which routes must be planned. Secondly (and by far the biggest issue, which is shown in Section 6.3.4) the push-pull planning gets increasingly expensive when the system gets larger. This is mostly due to the fact that for every extra reservation, two rows and two columns are added in the max-plus matrix. In order to make the context-aware routing algorithm a feasible option, research on reducing complexity is needed.

One of the options for future research regarding reducing complexity is to reduce the number of cost components. Furthermore, the max-plus matrix is built up using relations and other information in the reservations. This means that the information is present in different locations, and there might be no need to actually build the max-plus matrix (reducing complexity). Another idea might be to only use the max-plus matrix in case a new route is added and to only use the order of vehicles allowed to pass a tile (per tile) for execution. This looks like the logical time approach as discussed by Seibold [40]. The most promising part to reduce complexity is in the push-pull planning (and in specific the creation of the max-plus matrix) since Figure 6.13 showed that this is by far the most time consuming part of the model.

In the context-aware routing event a left tile signal is used for updating and deleting finished reservations. For more accurate knowledge on which reservations are active also an enter tile signal is needed. This can be explained using Figure 6.1. Here, an enter tile signal for tile 2 would result in the most left lower reservation to be removed earlier (in the current situation it is removed when the left tile signal for tile 1 is sent) and thus a better push-pull planning. This enter tile signal is not yet present in the *AgvSorter* model. Also for symmetry reasons it would be elegant to add the possibility to send this enter tile signal.

In Section 6.3.4 it became clear that planning a route from the first unclaimed tile in the route of an AGV is often not possible, causing AGVs to drive to their parking spot first and thus causing delays. A possible solution for this might be to use greedy time window reservations. This means that in case there is not enough time to fit the time needed for a reservation in a certain free time window, this free time window is extended (and all later reservations are pushed forward in time) causing the new reservation to fit anyways. An advantage of this is that AGVs are not likely to drive to their parking any more, but this greedy method might also cause other AGVs to be delayed. Periodic rerouting might help to solve this problem.

Another point that became clear from Section 6.3.4 is that AGVs often wait for a delayed vehicle to pass a certain tile first. This is due to the fact that planning routes in time fixes the order of AGVs allowed to pass a certain tile. An optimization in this would be the possibility to switch order of AGVs in case the AGV allowed to drive the tile first is expected to arrive there later than another AGV. Research on this must be performed since switching the order of AGVs might cause the planning to not be feasible anymore and thus might result in a deadlock situation.

In previous two paragraphs some suggestions for improvements in the model were presented. However, in case computational complexity is no issue there is another (better) solution. The context-aware routing algorithm is able to find optimal routes, given its context or the current state of the system. Here, context is the set of reservations on tiles that are already present in the system, the fixed costs and so on. Finding the optimal route given the context does not mean that the total solution is optimal. Therefore future research might be focused on designing a routing strategy that (this might be using rerouting) searches for the actual optimal total solution (yielding maximum throughput and minimum item lead time) rather than just searching the optimal solution for one route given the state of the system.

One of the conclusions of this research is that when using the context-aware routing algorithm, bidirectional layouts are a possibility and that the first simulation results are very promising. The expectation is that when the upper two solutions for reducing delays are implemented, bidirectional layouts result in an even better performance. However, in order to make this work some effort must be put in making the *AgvSorter* model fully ready for bidirectional routing. A side note is that bidirectional routing for higher numbers of vehicles might also cause more delays since AGVs can interact from more sides. Therefore simulations for higher numbers of vehicles are needed to verify whether this is indeed the case. It is expected that also the layout plays a role in this. For example, in layout 1 of the simulations in Section 6.3 it is expected that bidirectional lanes would not be beneficial (since AGVs block each other when driving in opposite direction) but in the case of layout 3 it might be.

The last point in the upper paragraph raises another question: what is the influence of layouts on routing using the context-aware algorithm in general? A study on which factors in layouts influence the results positively (and negatively) can be performed. Also the point of bidirectional routing can be taken into account here: layouts that are suited for using context-aware routing might work good in an unidirectional situation but might not work good in a bidirectional situation. Again the example of layout 1 versus layout 3 holds here: bidirectional lanes are probably not beneficial for layout 1 in combination with higher numbers of vehicles.

Bibliography

[1] A. W. T. Mors, J. Zutt, and C. Witteveen, "Context-aware logistic routing and scheduling," in *In Proceedings of the seventeenth international.* AAAI Press., 2007. [Online]. Available: http://dutiih.st.ewi.tudelft.nl/terMorsZuttWitteveen-icaps-2007.pdf iii, 15, 17, 18, 48

[2] Vanderlande.com. Company profile. Accessed on 21-09-2020. [Online]. Available: https://www.vanderlande.com/about-vanderlande/company-profile/ 1

[3] ——. Fleet. Accessed on 21-09-2020. [Online]. Available: https://www.vanderlande.com/airports/evolutions/fleet/ 1, 3, 5

[4] ——. Adapto. Accessed on 24-09-2020. [Online]. Available: https://www.vanderlande.com/warehousing/innovative-systems/storage-asrs/adapto/ 1, 3, 15

[5] ——. Agvsorter. Accessed on 21-09-2020. [Online]. Available: https://vikipedia.vanderlande.com/display/SIM/AgvSorter+Model#AgvSorterModel-AgvSortersystem 1, 6, 7, 8

[6] K. J. C. Fransen, "A path planning approach for AGVs in the dense grid-based agvsorter," Master's thesis, Eindhoven University of Technology, 2019, https://pure.tue.nl/ws/portalfiles/portal/138900912/Graduation_Report_Karlijn_Fransen_20190721.pdf. 1, 8, 9, 16, 55

[7] M. v. Weert, "Deadlock avoidance and detection for the grid-based AGV-sorter system." Master's thesis, Eindhoven University of Technology, 2019, https://research.tue.nl/en/studentTheses/deadlock-avoidance-and-detection-for-the-grid-based-agv-sorter-sy. 1, 4, 9

[8] Vanderlande.com. Airports. Accessed on 24-09-2020. [Online]. Available: https://www.vanderlande.com/airports/ 3

[9] ——. Warehousing. Accessed on 24-09-2020. [Online]. Available: https://www.vanderlande.com/warehousing/ 3

[10] ——. Parcel. Accessed on 24-09-2020. [Online]. Available: https://www.vanderlande.com/parcel/ 3

[11] H. Shi. Robots sorting system in chinese express company. Youtube. Accessed on 18-09-2020. [Online]. Available: https://www.youtube.com/watch?v=hzxlDE95XcY&feature=emb_title 3

[12] J. A. A. Jacobs, "The Design and Implementation of a Job Assignment Strategy in Automated Guided Vehicles." Eindhoven University of Technology, Tech. Rep., 2020, preparation phase report. 5, 8

[13] Vanderlande.com. Fleet bag future-proofing baggage logistics. Accessed on 28-09-2020. [Online]. Available: https://vikipedia.vanderlande.com/pages/viewpage.action?pageId=580097071#FLEETBag%22Futureproofingbaggagelogistics%22-WhatdrivesFLEET 5

[14] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, "Multi-heuristic a*," *The International Journal of Robotics Research*, vol. 35, 2016. 5

[15] CGTN. Watch an army of robots efficiently sorting hundreds of parcels per hour. Youtube. Accessed on 18-09-2020. [Online]. Available: https://www.youtube.com/watch?v=jwu9SX3YPSk 7

[16] T. Nishi and Y. Tanaka, "Petri net decomposition approach for dispatching and conflict-free routing of bidirectional automated guided vehicle systems," *IEEE Transactions on Systems, Manufacturing and Cybernetics*, vol. 42, 2012. 11

[17] N. Wu, "Resource-Oriented Petri Nets in Deadlock Avoidance of AGV Systems," in *Proceedings of the 2001 IEEEInternational Conference on Robotics Automation*, 2001. 11

[18] M. A. Kammoun, W. Ezzeddine, N. Rezg, and Z. Achour, "applied sciences FMS Scheduling under Availability Constraint with Supervisor Based on Timed Petri Nets," 2017. 12

[19] F. Taghaboni and J. Tanchoco, "A lisp-based controller for free-ranging automated guided vehicle systems," *International Journal of Production Research*, vol. 26, 1988. 12

[20] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959. 12, 16, 17, 55

[21] W. Zhang, Y. Peng, W. Wei, and L. Kou, "Real-Time Conflict-Free Task Assignment and Path Planning of Multi-AGV System in Intelligent Warehousing," *Chinese Control Conference, CCC*, vol. 2018-July, 2018. 13

[22] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, "Multi-heuristic a," *Proceedings of the 7th Annual Symposium on Combinatorial Search, SoCS 2014*, vol. 2014-January, 2014. 13, 16, 17, 55

[23] J. H. Lee, B. H. Lee, and M. H. Choi, "A real-time traffic control scheme of multiple AGV systems for collision free minimum time motion: A routing table approach," *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans.*, vol. 28, 1998. 13

[24] Z. Fan, C. Gu, X. Yin, C. Liu, and H. Huang, "Time window based path planning of multi-AGVs in logistics center," *Proceedings - 2017 10th International Symposium on Computational Intelligence and Design, ISCID 2017*, vol. 2, 2018. 13

[25] S. C. Srivastava, A. K. Choudhary, S. Kumar, and M. K. Tiwari, "Development of an intelligent agent-based AGV controller for a flexible manufacturing system," *International Journal of Advanced Manufacturing Technology*, vol. 36, 2008. 13

[26] C. W. Kim and J. M. Tanchoco, "Conflict-free shortest-time bidirectional AGV routeing," *International Journal of Production Research*, vol. 29, 1991. 14

[27] J. Huang, U. S. Palekar, and S. G. Kapoor, "A labeling algorithm for the navigation of automated guided vehicles," *Journal of Manufacturing Science and Engineering, Transactions of the ASME*, vol. 115, 1993. 14

[28] A. W. ter Mors and C. Witteveen, "Comparing context-aware routing and local intersection management," in *Belgian/Netherlands Artificial Intelligence Conference*, 2012. 15

[29] ——, "Plan repair in conflict-free routing," 2009. 15

[30] L. T. M. E. Penners, "Investigating the effict of layout and routing strategy on the performance of the Adapto system," Master's thesis, Eindhoven University of Technology, 2014. 15

[31] T. Lienert and J. Fottner, "No more deadlocks - Applying the time window routing method to shuttle systems," *Proceedings - 31st European Conference on Modelling and Simulation, ECMS 2017*, 2017. 15

[32] J. Hvezda, T. Rybecky, M. Kulich, and L. Preucil, "Context-Aware Route Planning for Automated Warehouses," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2018-November, 2018. 15

[33] D. Silver, "Cooperative pathfinding," *Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2005*, 2005. 15

[34] J.A.W.M. van Eekelen, *Modelling and control of discrete event manufacturing flow lines*, 2008. [Online]. Available: https://research.tue.nl/en/publications/modelling-and-control-of-discrete-event-manufacturing-flow-lines 27

[35] J. Komenda, S. Lahaye, J. L. Boimond, and T. van den Boom, “Max-Plus Algebra and Discrete Event Systems,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 1784–1790, 2017. 27, 28

[36] Esmaeil Bahalkeh, “Efficient Algorithms for Calculating the System Matrix and the Kleene Star Operator for Systems Defined by Directed Acyclic Graphs over Dioids,” Ph.D. dissertation, 2015. 27, 30, 31

[37] M. Chung, “Eigenvalues and Eigenvectors in the Max-Plus Algebra,” 1995. 30

[38] A. B. Kahn, “Topological sorting of large networks,” *Commun. ACM*, vol. 5, no. 11, p. 558–562, Nov. 1962. [Online]. Available: https://doi.org/10.1145/368996.369025 30, 31

[39] Michael Galarnyk, “Understanding Boxplots,” 2018. [Online]. Available: https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51 39

[40] Z. Seibold, *Logical Time for Decentralized Control of Material Handling Systems*, ser. Wissenschaftliche Berichte des Instituts fuer Foerdertechnik und Logistiksysteme des Karlsruher Instituts fuer Technologie. KIT Scientific Publishing, 2016. [Online]. Available: https://books.google.nl/books?id=b9uyDQAAQBAJ 49

[41] J. Y. Yen, “Finding the k shortest loopless paths in a network,” *Management Science*, vol. 17, 1971. 55

[42] D. Eppstein, “Finding the k shortest paths,” *SIAM Journal on Computing*, vol. 28, 2002. 55

A. Shortest path algorithms

Since a time-window based path planning strategy still needs some path planning approach, in this appendix some different approaches that can be used are discussed. These approaches can either be based on finding the shortest distance between source and destination or finding the path with the lowest costs. In this appendix Dijkstra's shortest path algorithm, the (modified) A* algorithm and k-shortest paths algorithm are discussed.

Shortest-distance algorithms search for paths from the source to the destination based on the shortest distance along the defined path segments on the geographical graph. The weights of the edges correspond to the length of the edges. The most well-known algorithm for calculating the shortest path is the Dijkstra algorithm [20]. This algorithm starts by calculates the costs from the source vertex to every output vertex. A vertex is (in the context of a grid-based layout) the middle of a tile. All the costs are then updated and the unvisited vertex with the lowest cost is then visited, followed by the same procedure. This continues until the destination vertex is visited. The shortest route can then be tracked down. Figure A.1 shows an example of the working of this algorithm. The unvisited vertices are shown in white and the visited vertices are shown in red. The shortest path from start (S) to destination (D) is shown in yellow.

The approach proposed by Dijkstra only takes into account the costs to get to a certain vertex. In order to direct the search more towards the destination, the A*-algorithm can be used [22]. By using a heuristic function $F(x) = G(x) + H(x, x_d)$ the search can be directed. In this function, $F(x)$ are the costs belonging to a certain vertex. $G(x)$ includes the costs to get to that vertex from the source vertex and $H(x, x_d)$ gives an estimation of the costs to get from that vertex to the destination vertex. The A* algorithm therefore does not have to look at all outgoing vertices (when a certain vertex is visited), but just the vertices that direct towards the destination. Note that when heuristic value $H = 0$, the algorithm is the same as Dijkstra's algorithm. Compared to Figure A.1 the A* algorithm works almost the same, except for the fact that the extra term for the remaining distance is added. This is shown in Figure A.2. In the left upper corner the value for cost to get to that vertex from start node S is indicated and in the right upper corner the estimation of the costs from that vertex to the destination vertex is indicated (heuristic value). The total costs are shown in the middle of each vertex. In this example the A* algorithm visits only 6 of the 8 locations, meaning that only 6 steps have to be performed compared to 8 in Dijkstra's algorithm.

As mentioned in Section 2.4.1, the separate path planning and traffic control strategy uses a modified A* algorithm [6]. Next to the fact that edge weights have a part that has to do with the distance of that edge, the weights of vertices change according to the time an AGV spends on that vertex. Also extra costs are introduced for turning here. This approach gives a more realistic view of the costs for traveling and outputs the path with the lowest expected costs (not necessarily the shortest path on the graph).

Other algorithms are able to calculate a number of shortest paths: the k shortest paths. With the method of Yen [41] it is possible to calculate the k shortest, loopless paths in which the complexity only rises linearly with k. Loopless in this case means that a vertex is not present more than once in a path. Eppstein [42] developed an algorithm that calculates the k shortest paths by applying the Dijkstra algorithm in reverse: from destination to source. All paths found to the visited nodes are saved in order to trace back which were the k shortest routes.

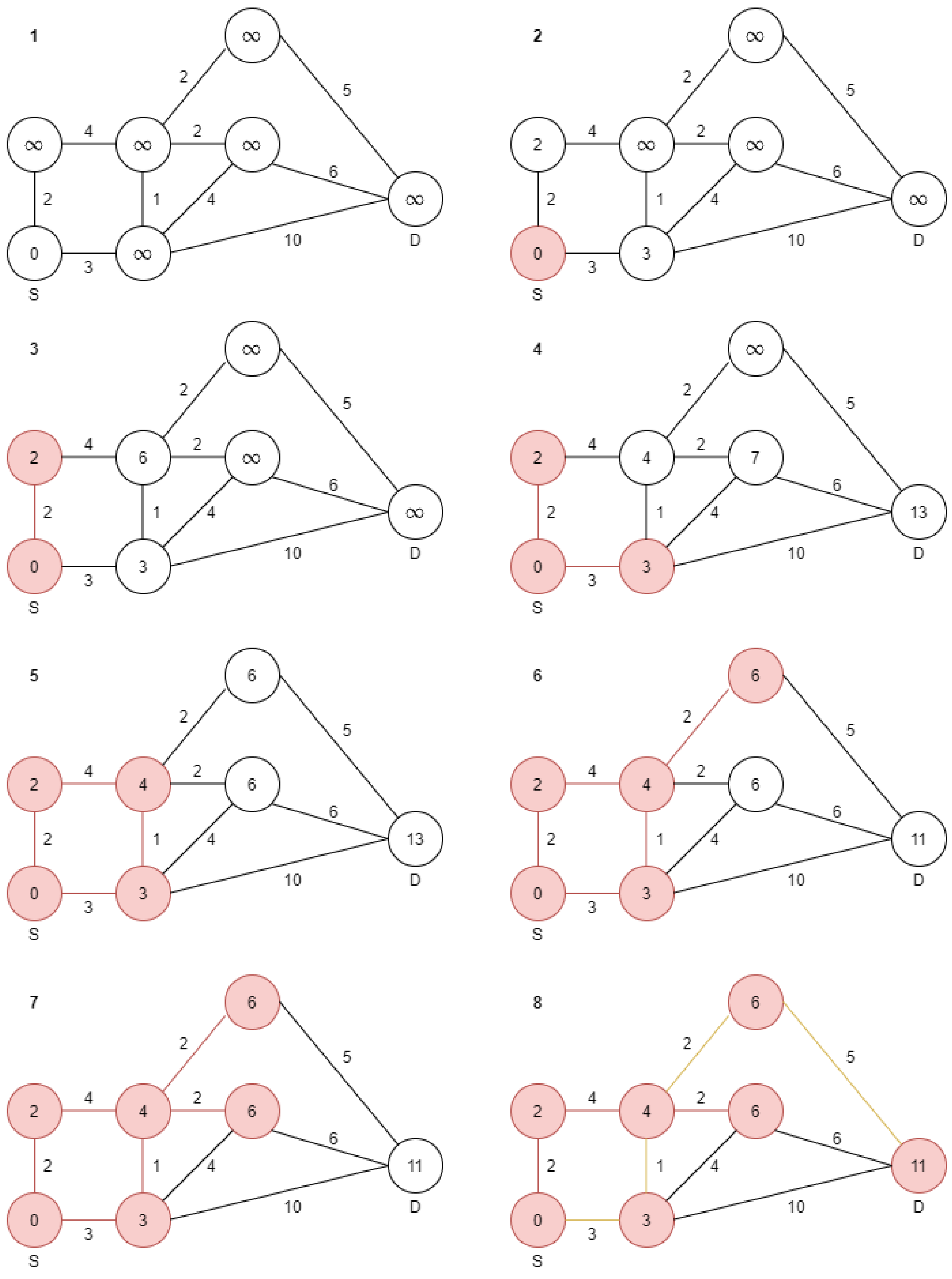


Figure A.1: Example of the working of Dijkstra's algorithm.

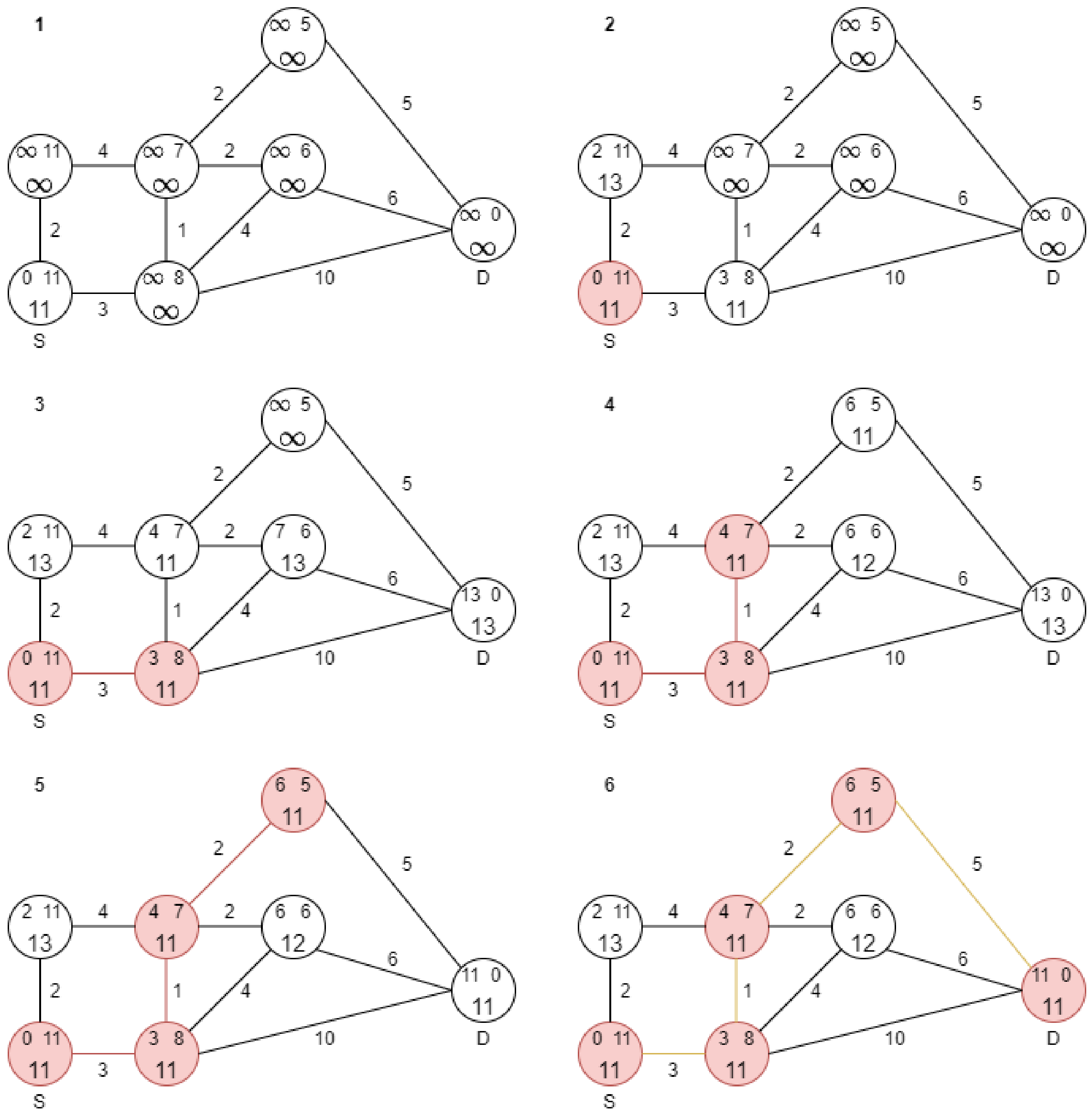


Figure A.2: Example of the working of the A* algorithm.

B. Implementation recommendations

In this appendix some recommendations regarding the implementation of the context-aware routing algorithm and the push-pull model are presented. Since these recommendations are not design choices but implementation choices, they are not discussed in the main text.

At first it is recommended to look at the reservations structure. For path planning a different reservation structure is used than in the push-pull model. In the context-aware routing cycle there is a conversion in these structures. To reduce computational complexity and improve readability and simplicity of the implementation it might be good to use only one reservations structure. One of the main challenges in this is that the relations between reservations are harder to define if the same reservation structure as in the route planner is used.

Another point that has to do with computational complexity of the model is the fact that the max-plus matrix is re-created ever iteration of the context-aware routing cycle. It is recommended to create a method to update this matrix in order to reduce the time needed for computing the matrix. The max-plus matrix is a sparse matrix (in the sense that most entries are equal to $-\text{Inf}$ and only up to three values per row are not) but in the administration of the implementation it fully saved. An idea is to save this matrix as a sparse matrix in order to reduce the amount of calculations and administration.

A next idea for reducing computational complexity regarding the max-plus model is to only compute the max-plus and dependency matrix (and perform push-pull) in case a new route is planned. The dependency matrix is similar to the max-plus matrix but has false entries where the max-plus matrix has value $-\text{Inf}$ and has true entries otherwise. If no new route is planned, the planning stays the same and for execution the order of AGVs allowed to pass certain tiles can be used. A side note to this is the fact that this probably only works for lower numbers of vehicles. As an example: when a cycle time of 0.5 s is used, 60 vehicles are driving around and the mean item lead time is 30 s, on average every cycle a new route is planned and the above mentioned ideas would not have an influence. When it comes to this, the earlier mentioned point of only updating the max-plus and dependency matrix in stead of re-creating these every iteration might be a better solution.

As could be seen in Section 6.3 and discussed in Section 6.3.4 for the simulations of layout 2 there were some cases in which no route could be found. This is a problem that should not occur since planning a route from a parking position (up to which previous routes are always planned) must always be possible (in the most extreme case this route starts at time infinity). Therefore expectation is that this is no problem in the algorithm but in the implementation, probably in the part where successor tile-time window combinations are added to the open list. It is recommended to spend some attention to this in order to be able to perform simulations (without failures) for all layouts. The occurrence of not finding a route would probably decrease in case the greedy algorithm as discussed in Section 7.2 is used, but this does not mean that the problem in the implementation is not there.

At this point the implementation model uses many functions and variables to perform the route planning. It is recommended to reduce the number of functions drastically to improve readability and simplicity. As an example there are many functions used for initialization which are probably not necessary and there are also multiple functions fulfilling the same goal (e.g. calculating the costs for moving). Next to the fact that it is recommended to reduce the number of functions and variables it is also recommended to perform some optimization to reduce the computational complexity of this algorithm. In case there are many reservations in the system, the route planning also seems to take a lot of time so improvements are welcome. Some ideas on this are to reduce the number of times the occupation set is updated, to make for example calculations for moving costs in one function and to only use either an occupation set (occupied time windows) or a free set (free time windows) in stead of both.