Graduation project at Vanderlande

Development of a deadlock recovery algorithm for grid-based AGV systems

Graduation project report

Manufacturing Systems Engineering
Department of Mechanical Engineering

By	J.H.N. Somers 0876720
TU/e supervisor	dr.ir. A.A.J. Lefeber
Vanderlande supervisors	ir. K.J.C. Fransen dr.ir J.A.W.M. van Eekelen

DC 2022.002

Eindhoven, January 21, 2022

Abstract

To use large scale AGV systems many control problems need to be solved. A Grid-Based Controls model is developed within Vanderlande to study and improve control algorithms. Deadlock handling is one of the control problems. The goal of the project is to develop a deadlock recovery algorithm for grid-based AGV systems. The algorithm is developed with maximising throughput in mind and balancing the complexity of the added code. The algorithm is used in grid-based layouts, which means that the layout is divided into grid elements called tiles. In a deadlock, a circular dependency is present of AGVs that need the tile held by another AGV to proceed. This has the result that AGVs wait indefinitely. To resolve a deadlock, the circular dependency needs to be broken. This is done by rerouting one of the involved AGVs, called resolving AGVs.

In literature, dedicated deadlock recovery infrastructure is often used to make sure an AGV can always directly divert. It is shown that all deadlocks can be recovered without the use of dedicated deadlock infrastructure. To do so, an algorithm is given, including system requirements, that lets AGVs always reach their destination. The actual algorithm tested and implemented in the Grid-Based Controls model is different for efficiency reasons. The algorithm is based on a modular framework in which each sub-algorithm can be adjusted without influencing the consecutive calculations. The framework consists of four main parts and is established after identifying the necessary recovery steps. These steps are found by analysing the formation of deadlocks in a range of settings and layouts. Different alternatives of sub-algorithms are designed to create different deadlock recovery algorithm variants.

The first sub-algorithm creates a resolvability ranking of involved AGVs. The ranking reflects which AGV is expected to be the best resolving AGV. Multiple ranking methods are created based on AGV and system properties. Next, a set of involved AGVs is appointed as resolving AGVs, by choosing the AGVs in the top of the ranking. The number of resolving AGVs is varied during testing. A path planner tries to plan a deadlock-free recovery path for resolving AGVs. Path planning during deadlock recovery is done with a slightly adapted variant of the default path plan algorithm developed by Fransen [1]. If the initial resolving AGV(s) cannot resolve the deadlock, an extra resolving AGV needs to be found. Two different methods are tested for this sub-algorithm.

Exploratory simulations are done on a small layout to compare algorithm variants. The results of these simulations created a list of algorithm variants to test on full-scale layouts. It is shown that deadlock algorithm variants that lower congestion around a deadlock the most, give the best system throughput. Because of this, the Least Extra Costs Path ranking gives the best results. The ranking generally provides a resolving AGV that wants to get away from the congested deadlock area and can easily do so. Furthermore, increasing the number of initial resolving AGV is only useful under specific circumstances. And lastly, the two methods to choose an extra resolving AGV do not show significant throughput differences, thus the simplest method is chosen.

The original deadlock handling technique is deadlock avoidance. The final implemented deadlock recovery algorithm is tested in combination with different deadlock avoidance settings. Simulations show that the throughput of systems using deadlock recovery without deadlock avoidance can equal that of systems that include deadlock avoidance. A requirement to level throughput without deadlock avoidance to systems including deadlock avoidance, is a small deadlock detection interval. Moreover, only using deadlock detection and recovery as deadlock handling method can lower the computation time by a factor four, given the smallest deadlock detection interval tested.

Preface

I was eight years old when I travelled for the first time by airplane. The experience of walking through Schiphol Airport, boarding a small propeller plane and arriving in a rainy Ireland was wonderful. However, the most mesmerising part of the holiday was the fact that our suitcase disappeared behind a check-in desk and neatly presented itself again on a conveyor belt on the other side of the Channel. Its a nice way to conclude my master Manufacturing Systems Engineering within the company that created the experience that sparked my enthusiasm for mechanical engineering and automation: Vanderlande.

It was a great opportunity to help solve a genuine problem during my graduation project. It felt good to develop an algorithm and program code that is directly implemented and used in the existing simulation model of the team. I want to truly thank Karlijn Fransen en Joost van Eekelen as my company supervisors. Thank you for the nice brainstorm sessions and sending me home with more questions than answers. I'm also grateful for the weekly meetings with my thesis supervisor Erjen Lefeber who helped me to align my thoughts and look critically at my own work.

My graduation project at Vanderlande marks the end of my time at the TU/e. What started as a big guess proved to be a great experience. The 62^{nd} Board of W.S.V. Simon Stevin, dispuut HenK, housemates of Octo Alpha and all other friends: a big thank you to you all.

Glossary

Term	Definition
AGV	Automated Guided Vehicle
- Blocked	An AGV for which the next tile of its path is occupied
- Connected	A blocked AGV which is connected to a deadlock, but not part of the deadlock
- Involved	A blocked AGV which is part of a deadlock
- Resolving	An AGV whose path is replanned to resolve a deadlock
Chute	Exit slide in a baggage or parcel sorting system
Grid density	Ratio of the number of AGVs in a system divided by the number of driveable tiles in the layout
Handling point	Location in a layout where an AGV can interact with the environment
Job	Task to bring a parcel or baggage item from a pick-up to a drop-off location
Layout	Area on which an AGV system operates including possible obstacles and handling points
Resolvability ranking	Ranking of involved AGVs of which AGV is most suitable to be resolving AGV
Segment	Driveable link between two tiles
- Deadlock	The segment leading to the next involved AGV in the deadlock
- Divert	A outward segment that might be used in a deadlock resolving path
- Prohibited	A segment receiving a temporary extra high weight to prevent it is chosen in a recovery path
Tail	Set of connected AGVs
Tile	Grid element in a layout
- Involved	A tile on which an involved AGV is located.
Throughput	The number of jobs that can be processed per unit time

Contents

1 Introduction

Vanderlande is a market leader in automation solutions of logistic processes [2]. The company consists of three business segments: airports, parcel and warehousing. For the airport segment, the company develops baggage handling systems. These systems are installed at more than 600 airports around the globe, including twelve of the twenty biggest airports in the world. The parcel segment is centred around sorting parcels as quickly as possible, which is done about 52 million times a day by all their parcel equipment combined. In warehousing, processes are automated by for instance automated storage and retrieval systems or picking stations.

Especially in the airport and parcel segment, the solutions of Vanderlande use conveyor belts to transport and sort baggage or parcels, see Figure 1.1. With this proven technology, high throughput can be achieved. Throughput is the number of bags or parcels that are delivered per time instance. It is the most important performance measure of sorting systems. The conventional conveyor belt systems have some drawbacks. Low flexibility is one of them, since conveyor belts need fixed infrastructure. Expanding or adapting a layout is therefore expensive and time-consuming. The need for fixed infrastructure makes conveyor belt systems also not very scalable. Lastly, conveyor belt systems are also not robust. If a motor on the main conveyor belt fails, the entire system might come to a complete standstill. Expensive redundant systems need to be installed to achieve enough availability of the systems of high demanding customers. To overcome the mentioned disadvantages of traditional conveyor belt systems, Vanderlande develops systems that use Automated Guided Vehicles (AGVs). In these systems, each AGV individually transports an item from a pick-up to a drop-off location. AGVs can therefore execute the same transport and sorting tasks as conveyor belt systems.

Figure 1.1: Parcel sorting system using conveyor belts [3].

AGV systems come with their own challenges. Many control problems have to be solved. Vanderlande uses a Grid-Based Controls model to develop control algorithms for AGV systems. The division of tasks among AGVs is one of them. Finding an efficient path between the sources and destinations is also an important task. Furthermore, control methods need to be implemented to avoid collisions and handle deadlocks. A deadlock consists of a chain of blocked AGVs. A blocked AGV is an AGV for which the next tile of its path is occupied. The chain of blocked AGVs creates a situation in which the AGVs wait indefinitely on each other to move.

Thesis objectives
A deadlock has a negative influence on the performance of an AGV system. In the worst case, a deadlock can even stall the complete system. Consequently, it is important to have a strategy to handle deadlocks. A deadlock avoidance algorithm is implemented by Van Weert [4]. This algorithm checks if the next step of an AGV leads to a deadlock. If that is the case, the AGV has to wait and can temporarily not go to the next tile of its path. Many deadlocks can be avoided with the deadlock avoidance algorithm, although this is not the case for all deadlocks. Being able to recover from a deadlock is hence a necessity if the AGV controls are to be implemented at customers. That is why a deadlock recovery algorithm is designed during this project. The algorithm should be robust to make sure it can be used in different settings and layouts. Furthermore, throughput loss due to deadlocks should be minimised, as throughput is the most important performance measure for customers. Lastly, the computational complexity should be taken into account to assure the algorithm can be executed in real-time and the code is easily maintained. The aim of this project is therefore as follows:

Development of a deadlock recovery algorithm for grid-based AGV systems, keeping in mind the effect on system throughput and computational complexity.

The development of a deadlock recovery algorithm is split into the next five objectives.

- Explore deadlock recovery techniques in literature and determine which methods can be applied to the grid-based AGV system.
- Create a deadlock recovery framework. To do this, possible deadlock recovery steps and approaches must be identified.
- Implement the found deadlock recovery framework in the existing Grid-Based Control model including different variants.
- Determine the most suitable deadlock recovery algorithm based on simulation results.
- Compare deadlock handling strategies, including the developed deadlock recovery algorithm.

Report structure
The report continues with the following structure. In Chapter 2 the use of AGV systems within Vanderlande is explained as well as the Grid-Based Controls model used during the project. The structure of deadlocks and the different deadlock handling techniques are elaborated in Chapter 3. In Chapter 4, it is shown under which conditions deadlocks can be recovered. The actual tested and implemented deadlock recovery algorithm differs from the algorithm of Chapter 4 and is worked out in Chapter 5. Next, exploratory simulations and results are shown in Chapter 6. Based on the exploratory simulations, a set of deadlock recovery algorithms is tested on full-scale simulations. These simulations and results are discussed in Chapter 7. Lastly, the report is concluded in Chapter 8 in which also recommendations for future work are given.

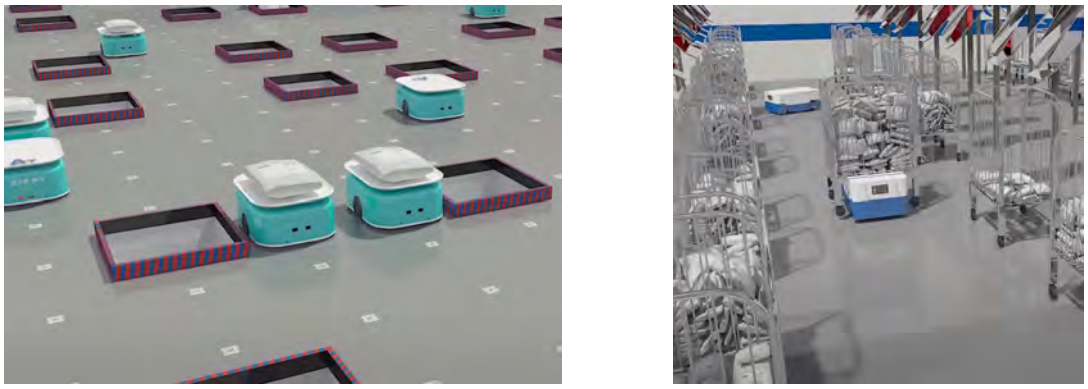
2 AGV systems within Vanderlande

This chapter dives deeper into different AGV applications and the general control problems that need to be tackled. Section 2.1 explains possible AGV applications within Vanderlande over the three business segments. Secondly, Section 2.2 describes the functioning of the Grid-Based Controls model of Vanderlande which is used during this project.

2.1 AGVs within Vanderlande

Automated Guided Vehicles are material transport devices that can drive without the need for a human operator. This property makes AGVs suitable for automated logistics, which is the core business of Vanderlande. AGV systems are used and investigated to enhance the flexibility and scalability of transporting and sorting systems over the traditional conveyor belt systems. AGV systems can potentially be used in all market segments of Vanderlande since all market segments involve transporting goods in some way.

In the parcel segment, the goal is to sort parcels as quickly as possible with the need for maximal reliability [3]. AGVs can be used for this goal, potentially with different sizes of AGVs. The 'Zippy' AGV is a nice example of the use of AGVs in the parcel segment [5]. Each AGV is tasked with delivering one parcel to the right chute. Bigger AGVs can also be used to load and unload parcel containers, see Figure 2.1.



(a) AGVs sorting parcels to chutes (b) AGV transporting full parcel containers

Figure 2.1: Two examples of the use of AGV in the parcel market segment [5].

In the market segment warehousing, many goods need to be sorted, transported and stored. AGVs can assist in moving incoming goods to the right storage location. The ADAPTO system is an example of a shuttle system that is currently used for warehousing to automatically store and retrieve products, see Figure 2.2a. AGVs can also be used in the airport segment of Vanderlande [6]. AGVs can serve as baggage sorting devices and/or as transport systems. FLEET is a baggage handling system, based on the use of AGVs, that is currently used in some smaller projects. Figure 2.2b shows the FLEET system in a check-in area of an airport.

AGVs can provide a solution to the need of Vanderlande to provide systems with higher robustness, scalability and flexibility. Much less infrastructure is needed and the overall system can proceed if just one AGV shuts down. The main challenge, however, is achieving enough throughput with AGV systems [7].

(a) ADAPTO storage and retrieval system [8].

(b) FLEET system use at Dallas Airport [9].

Figure 2.2: Examples of current AGV usage within Vanderlande.

2.2 Grid-Based Controls model

AGVs can be controlled based on different type of layouts. AGV systems that use grid-based layouts are not yet implemented in real-life by Vanderlande. However, Vanderlande uses a Grid-Based Controls simulation model in MATLAB. The Grid-Based Controls model is built and continuously being developed to test and simulate algorithms for grid-based AGV systems. This Grid-Based Controls model is used during this project. Therefore, this section explains the structure of the Grid-Based Controls model.

In a grid-based layout the driveable area in a layout is split into virtual grid elements, called tiles, see Figure 2.3. A tile is a driveable zone of the total area in which the AGVs operate. Such an operating area can for instance be a baggage sorting area. An AGV can only drive from tile A to tile B if a segment is present between the two tiles. The segments have specified driving directions. Most of the time unidirectional segments are used.

The Grid-Based Controls model has some ground principles [10]. The first principle is that tiles do not overlap. Another principle is that AGVs move from tile to tile. Besides that, AGVs do not touch any other tiles if they move from from tile A to tile B. An AGV has to reserve a tile before it can drive on it. To avoid collisions, a tile can only be reserved by one AGV at a time. Furthermore, each tile is big enough that an AGV can turn within a tile, without touching other tiles.

AGVs in the Grid-Based Controls model perform sorting tasks. An AGV has to first retrieve a job at a pick-up location to start the sorting task. A pick-up location is for instance the infeed from the check-in desk at an airport. Thereafter, the job must be delivered to a drop-off location. This might be a chute that brings a suitcase to the correct truck. Drop-off locations might be grouped when they correspond to the same exit. This can be the case on four sides of a chute. In that case, it does not matter to which of the grouped drop-off locations the AGV drives. Apart from driveable tiles, there can also be obstacles in the layout, such as concrete columns or the location of chutes. Figure 2.3 illustrates how obstacles, handling locations and the AGV status are visualised in the Grid-Based Controls model. Furthermore, the driving direction on each tile is also visible by the directed segments.

The control methods in the Grid-Based Controls model are as follows. Job assignment happens online in a dynamic manner. An AGV gets a new job whenever it is finished with its current job. It is sent to the nearest pick-up location to get a new job. Often an infinite supply of new jobs is used, with destinations of jobs that are randomly chosen. At a pick-up point, jobs are distributed to AGVs in a first come first serve manner.

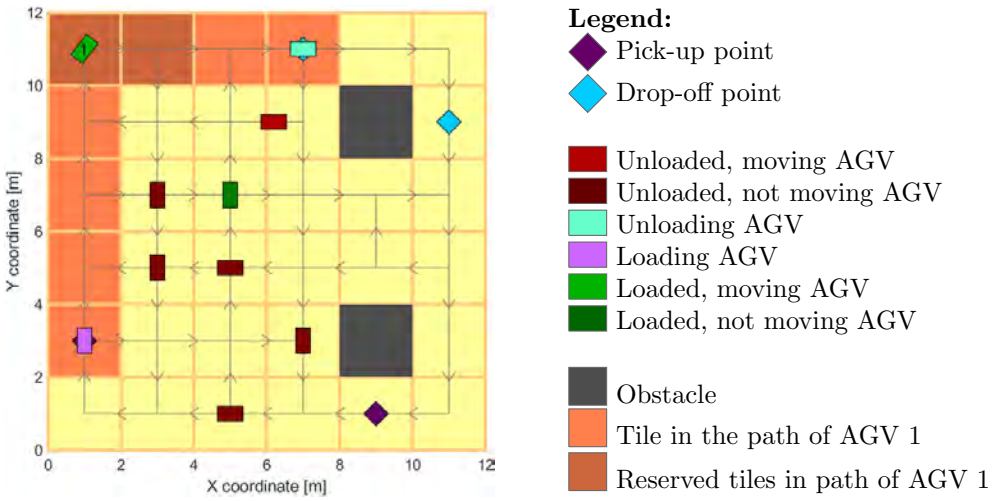


Figure 2.3: Sample layout in the Grid-Based Controls model with AGVs [11]. The status of an AGV is visualised by different colors. An AGVs path is shown in (dark) orange.

Paths are only planned per task. For instance, when an AGV loads a parcel at a pick-up point, the AGV only receives a path to the destination of the parcel, not any further. For path planning the method of Fransen et al. [1] is used. This method is an A* path planning algorithm combined with dynamic node weights. The algorithm searches for the cheapest deadlock-free path. Each tile is represented by a node in a graph. Each node has a weight that can change during simulation. The weights of the nodes increase as AGVs are standing still on the nodes, and dampen out if AGVs start driving again. The idea of this method is to plan paths outside of congested areas and distribute AGVs more evenly over the whole layout.

Lastly, traffic control in the Grid-Based Controls model makes sure that AGVs do not collide or enter a dead- or livelock. Traffic control is independent of path planning. Thus, the path execution is decoupled from path planning. An AGV can only continue its path if it has reserved the next tile(s) of the path. Traffic control makes sure that only one AGV can reserve a tile at the same moment to prevent collisions. An AGV tries to reserve multiple tiles in a row to be able to continue driving without stopping. An AGV has to wait until another AGV releases a tile, if the next tile to reserve in the path is reserved by another AGV. An AGV only releases a tile if the AGV fully moved to the next tile. The second part of traffic control consists of a deadlock avoidance algorithm implemented by Van Weert [4]. The deadlock avoidance algorithm prevents that an AGV is allowed to reserve a tile that leads to a deadlock. Deadlocks are elaborated further on in Chapter 3. On the other hand, in a livelock AGVs do move, but there is no general progress. Livelocks are currently no problem within the model.

So, there are algorithms for all general AGV control steps in the Grid-Based Controls model, including a deadlock avoidance algorithm. However, not all deadlocks can be avoided [4]. Once a deadlock occurs, the full system will stall in the end, because paths are blocked by the deadlock. Currently, there is no deadlock recovery method in place. This means that deadlocks can only be resolved by manual intervention. Manual intervention is not wanted as seen from a performance perspective, but also from the perspective of full automation. Furthermore, current control methods are geared towards avoiding deadlocks no matter the costs, since deadlocks cannot be recovered easily. Developing a deadlock recovery algorithm reduces the costs of a deadlock, which might open up possibilities to make other control steps more efficient. In other words, the addition of a deadlock recovery algorithm increases the quality of the AGV controls. To find a suitable recovery algorithm, different types of deadlocks and deadlock handling methods are discussed next in Chapter 3.

3 Deadlock characterization

In order to develop a deadlock recovery algorithm, the characteristics of deadlocks need to be understood. A deadlock recovery algorithm can only be made if it is properly known what deadlocks are and how they are formed. So, this chapter dives deeper into deadlock conditions (Section 3.1) and different types of deadlocks (Section 3.3). In between, Section 3.2 describes the four general deadlock handling techniques. These are elaborated to understand the impact of deadlock handling on the type of deadlocks that can occur.

3.1 Deadlock conditions

Deadlocks have been first researched in the context of multiprogramming operating systems. Tasks are distributed and concurrently executed to enhance the utilization of the system resources. The division of tasks may create a situation in which two tasks both wait on the completion of the other task. In this situation both processes wait forever to proceed, meaning that there is a deadlock. Coffman [12] describes the four necessary and sufficient conditions for a deadlock. The four Coffman conditions translated to AGV systems are:

- 'Mutual exclusion' condition: AGVs claim exclusive control of the tiles they reserved. Only one AGV is allowed on a tile.
- 'Wait for' condition: AGVs hold tiles already allocated to them while waiting for the reservation of additional tiles. Tiles are only released if an AGV is fully located on the next tile in its path.
- 'No preemption' condition: Tiles cannot be emptied before an AGV completes the use of a tile. An AGV cannot leave a tile spontaneously.
- 'Circular wait' condition: A circular chain of paths exists, such that each AGV holds one or more tile(s) that are needed by the next AGV in the chain.

The first three Coffman conditions are always true for AGV systems. Figure 3.1 shows an example of a deadlock situation, in which also the circular wait condition is satisfied. Each red block in the figure depicts an AGV. The arrow on the AGV shows the desired driving direction.

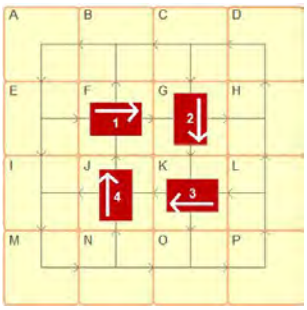


Figure 3.1: Deadlock with four involved deadlocks [4].

A deadlock resolution method can only handle deadlocks in the AGV system by preventing or resolving a circular wait condition, since the first three Coffman conditions are always true.

The circular waiting condition arises if AGVs are close to each other and AGVs want to reserve tiles which are in the path of another AGV. A deadlock can happen anywhere in a layout where AGVs encounter each other and their paths cross. However, such an encounter is way more prone to happen in areas of a layout where many interactions between AGVs happen. That is why deadlocks generally occur in congested and dense areas of a layout. Handling points, such as pick-up locations, are therefore likely to experience deadlocks.

3.2 Deadlock handling

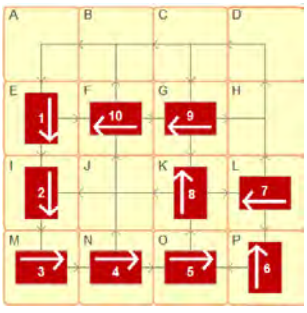
A deadlock occurs if there exists a circular wait condition between AGVs. There are multiple ways to deal with the condition. Deadlock handling methods can be divided into four main categories [13]: ignorance, prevention, avoidance, and detection and recovery. With deadlock ignorance, there is no strategy in place to deal with deadlocks. This strategy can be used if deadlocks occur infrequently and controlling deadlocks is technically or financially difficult. Deadlock prevention methods are deadlock handling techniques in which resources are granted in such a way that deadlocks never occur. Deadlock prevention strategies are used before path execution is started. An example of deadlock prevention is reserving all the needed tiles of a path at once. Deadlock prevention policies tend to be very conservative. The third deadlock handling strategy is deadlock avoidance. Deadlock avoidance methods determine if a tile can be granted by looking at the resulting state. The tile can be granted, if the resulting system state is safe. These methods happen online and usually require large computations. The last deadlock handling strategy is detection and recovery. With this method, resources are granted without any check. However, a detection method checks periodically if a deadlock has occurred. A recovery procedure is started, if a deadlock is found. Using deadlock detection and recovery is believed to perform worse than deadlock avoidance, since the method is reactive instead of proactive [14].

In the Grid-Based Controls model, a deadlock avoidance algorithm is chosen as the main deadlock handling method. The method checks if a circular wait condition holds if an AGV would reserve a tile. An AGV is only allowed to reserve a tile if this circular wait condition does not hold. The current deadlock avoidance method does a proper job in avoiding most deadlocks. However, not all deadlocks can be avoided if paths of AGVs are not completely known [4]. Thus, there is a need for an additional deadlock recovery algorithm.

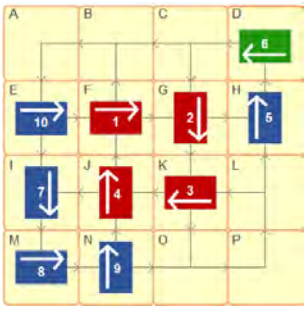
3.3 Deadlock types

Deadlocks arise when a circular dependency is present. Deadlock avoidance is used in the Grid-Based Controls model to avert the circular dependency. The use of deadlock avoidance introduces multiple types of deadlocks. The three general types of deadlock are discussed below, since deadlock recovery should be able to resolve all of them.

Monocycle deadlocks
The deadlock type that corresponds with the definition of Coffman is the monocycle deadlock. In this type of deadlock, all the involved AGVs directly wait on the release of a tile that is occupied by another AGV in the circular wait chain. Figure 3.2 shows two examples of monocycle deadlocks.



(a) Monocycle deadlock with ten involved AGVs.



(b) Monocycle deadlock with tails.

Figure 3.2: Two monocycle deadlocks. Red AGV are involved AGVs, blue AGVs are connected AGV and the green AGV is a non-blocked AGV [4].

Figure 3.2a shows a monocycle deadlock with ten involved AGVs. The ten AGVs clearly satisfy

the circular wait condition which confirms the existence of a deadlock. Figure 3.2b shows that not only involved AGVs can be affected by a deadlock. AGV 1, 2, 3 and 4 are involved in the deadlock. However, AGV 7, 8, 9 and 10 are also unable to proceed due to the deadlock. These AGVs are called connected AGVs and can also be referred to as the tail of the deadlock. AGV 6 is not affected by the deadlock as it can reserve its next tile.

Multicycle deadlocks

Multicycle deadlocks occur when monocycle deadlocks are avoided. There are multiple imminent deadlock cycles for a specific tile in this type of deadlock [15]. Multiple AGVs can reserve their next tile, but doing so results in a deadlock. The reservation is therefore prohibited by the monocycle avoidance algorithm. Strictly speaking, the circular wait condition of Coffman does not hold for this type of deadlock, since multiple AGVs can still move. The number of potential deadlock cycles in multicycle deadlocks can vary. Figure 3.3 show an example with respectively two and three imminent deadlock cycles. Figure 3.3a shows an imminent deadlock cycle by AGV 3, 4 and 5. The deadlock cycle would be complete if AGV 2 reserves tile K. Similarly, AGV 1, 2 and 6 form an imminent cycle. This imminent cycle becomes a deadlock if AGV 5 reserves tile K. Since AGV 2 and 5 are both not allowed to reserve zone K a multicycle deadlock arises. AGV 2 and 5 are called prohibited AGVs. Figure 3.3b shows a three-cycle deadlock. In a three-cycle deadlock one imminent cycle is connected to two other imminent cycles. In this case, the imminent cycle with AGV 3 and 7 is connected to two other imminent cycles.

(a) Two-cycle deadlock.

(b) Three-cycle deadlock.

Figure 3.3: Two examples of multicycle deadlocks. Red AGVs are involved deadlocks, orange AGVs are prohibited deadlocks [4].

Inevitable Multicycle Deadlock

In a multicycle deadlock, a monocycle deadlock occurs within a few steps. That is why a multicycle deadlock avoidance algorithm tries to prevent multicycle deadlocks. Avoidance of multicycle deadlocks can, however, introduce another type of deadlock: an inevitable multicycle deadlock.

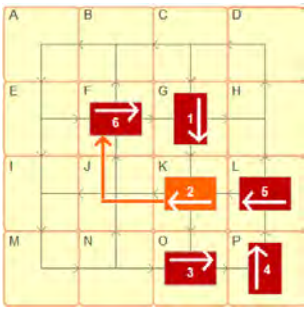


Figure 3.4: Inevitable two-cycle deadlock [4].

Figure 3.4 shows an example of an inevitable two-cycle deadlock. The avoidance algorithm pro-

hibits AGV 2 to reserve tile J, since this reservation leads to a two-cycle deadlock of Figure 3.3a. Despite a two-cycle deadlock is avoided, the system is in deadlock since no AGV can move, because AGV 2 is not allowed to reserve its next tile.

Avoiding inevitable two-cycle deadlocks, can lead to inevitable three-cycle deadlocks. New more complex deadlocks are introduced each time a simpler type of deadlock is avoided. The more complex the deadlock becomes the less frequently it happens. However, in practise not all deadlocks can be avoided due to limited computation time and the fact that not all paths of AGVs are known.

To summarise this chapter, deadlocks occur when the four Coffman conditions are met. The 'circular wait' condition is the only condition that can be prevented or broken in AGV systems to avoid, prevent or resolve deadlocks. The current main deadlock handling method is deadlock avoidance. Different types of deadlocks occur, depending on the deadlock avoidance algorithm used. As stated above, deadlock avoidance cannot avoid all deadlocks. A deadlock recovery algorithm is needed to create robust controls in which all deadlocks can either be avoided or resolved. Therefore, the next Chapter 4 proposes an algorithm and discusses conditions under which all deadlocks can be recovered.

4 Deadlock recoverability

Chapter 3 shows that deadlocks can occur in many different ways. For instance, the layout, number of AGVs in a system and the job distribution can have a big influence on then size, shape, location and the number of the deadlocks that occur. It is key that a deadlock recovery algorithm can make a system deadlock-free, no matter the deadlock differences. To this end, a study is done to see if deadlocks can always be resolved and if so, under which conditions. The result of this study is shown in this chapter. An algorithm, including boundary conditions, is presented which guarantees that each deadlock can be resolved. This is done by making sure that each AGV can reach each tile in the layout. The found algorithm can be implemented in the Grid-Based Controls model, if an absolute guarantee is needed that each AGV can reach its destination. However, the actually implemented deadlock recovery algorithm, which is described in Chapter 5, is different from the algorithm presented in this chapter. The implemented algorithm is different, since the algorithm presented in this chapter comes at the cost of high inefficiency. Implementation of the found algorithms in the Grid-Based Control model would also require many unwanted adjustments of the existing controls.

Many different system layouts and controls can be used in AGV systems. For the propositions in this chapter some principles, already applied within Vanderlande, are used to confine the problem. First, only layouts are regarded that use non-overlapping tiles. Next to that, each tile is big enough to fully contain an AGV. So, an AGV is that is located on a tile it does not occupy any other tiles. Segments can exist between tiles to connect them. Moreover, only strongly connected layouts are considered. These are layouts in which there is a path from each tile to each other tile in the layout, and implies that there are no sinks or sources. Lastly, the path of all AGVs in the system can be altered.

The propositions in this chapter have a strong link to graph theory. To aid the propositions a few definitions are elaborated below.
Path: a sequence of segments in which all segments are distinct.
Directed path: a path in which the orientation of all segments is the same.
Directed layout: a layout in which the segments have a specific direction.
Strongly connected directed layout: a directed layout in which there is a directed path from any tile to any other tile in the layout.
Directed circuit: a non-empty directed path in which the first tile is equal to the last tile.
Subcircuit: a circuit whose tile and segment sets are subsets of the tile and segments set of another circuit.
Directed cycle: a directed circuit in which only the first/last tile occurs twice, all other tiles only occur once.
Directed non-cycle: a directed circuit that is not a cycle.
Common tile: a tile which is part of multiple cycles.

Two propositions are used to show when deadlocks deadlock can be resolved. The split is based on the layout that is used. Only strongly connected layouts are considered. This means that there is a path from each tile to each other tile in the layout. A directed circuit can be formed by connecting the paths from and to a tile. A circuit is a path that starts and ends at the same tile. Circuits can either be cycles or not. The circuit is also a cycle if the start/end tile is the only tile that occurs twice in the circuit. A circuit is called a non-cycle if the circuit is not a cycle. Figure 4.1 shows an example of circuit being a cycle or not. In most of the layouts used at Vanderlande, a cycle can be formed between each tile pair. The first proposition focuses on showing recoverability for these layouts. The second proposition discusses recoverability for layouts in which there is a non-cycle between at least one tile pairs.

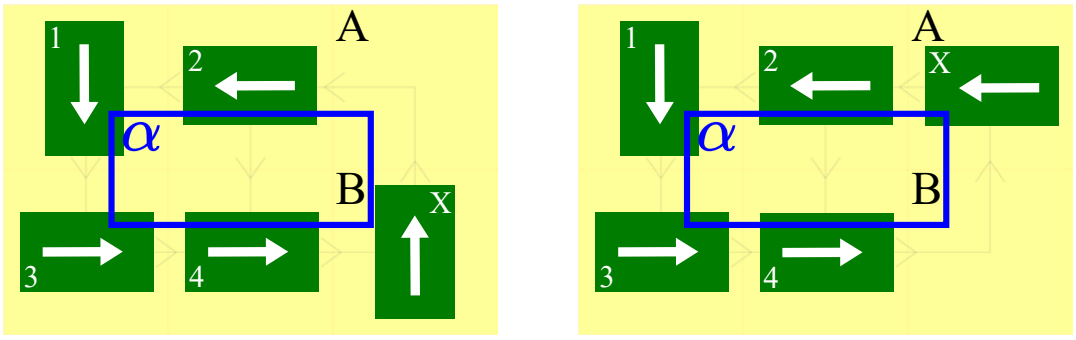
(a) A directed circuit, and a cycle (b) A directed circuit, but not a cycle (non-cycle)

Figure 4.1: Examples of small strongly connected directed layouts with circuits formed by connecting paths from tile A to B and tile B to A.

First, a general lemma is discussed to help the two following propositions.

Lemma 1 *Each tile in a circuit can be emptied if that circuit includes an empty tile, movement of an AGV to the empty tile is not restricted and AGVs outside the circuit do not move.*

Given a circuit α with an empty tile A, there is an AGV X in that circuit which has empty tile A in front, when following the direction of the circuit, see Figure 4.2a. AGV X can move to tile A since it is empty. The previous tile of AGV X, tile B, becomes empty if AGV X drives to tile A, see Figure 4.2b. This means that the empty tile moves one step backwards compared to the direction of the circuit α, if an AGV which is located in the circuit drives to the empty tile. The procedure of moving AGVs along the circuit to the newly created empty tile can be repeated indefinitely. Thus, each tile in the circuit can be emptied.



(a) Starting situation (b) Empty tile relocated within circuit

Figure 4.2: Each tile in circuit α can be emptied by moving AGVs in the direction of the circuit.

4.1 Deadlock recoverability in a layout with cycles

This section discusses the proposition for layouts in which a directed cycle can be formed between each tile pair. This means that from each tile A, a circular path can be found via each other tile in the layout in which only tile A occurs twice in the path. Lemma 1 shows that the empty space in a circuit can be moved around. Lemma 2 extends the same idea to the movement of AGVs within a cycle.

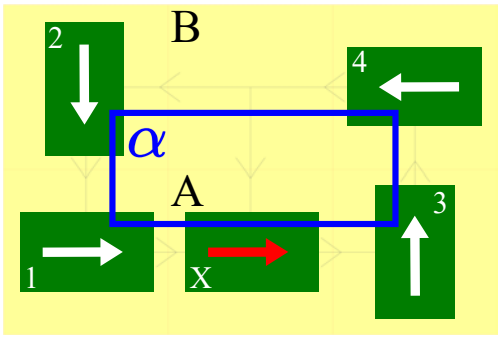
Lemma 2 *An AGV which is located in a directed cycle can be moved to any location in the cycle, if that cycle includes an empty tile, movement of an AGV to the empty tile is not restricted and AGVs outside the circuit do not move.*

Given an AGV X which is located in a cycle with an empty tile, AGV X can move in the direction of the cycle if the successor tile is empty. By Lemma 1, it is possible to empty the successor tile of AGV X. If that is done, AGV X can move one step in the direction of the cycle. The procedure can be repeated indefinitely. By repeating the procedure, AGV X can move sequentially one step further in the direction of the cycle, and thus move to any location within the cycle.

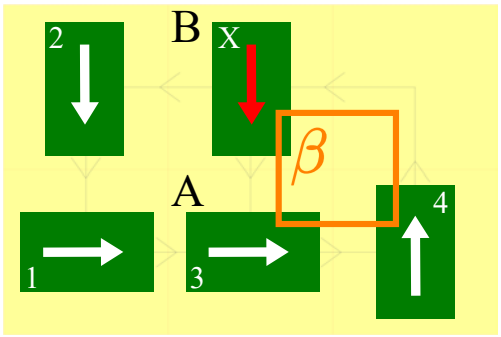
Lemma 1 and 2 show that AGVs and an empty space can be moved around in a cycle. With this insight, Proposition 1 shows that each AGV can reach each tile in the layout.

Proposition 1 *One empty driveable tile in a strongly connected directed layout is necessary and sufficient to make sure each AGV can reach its destination. Requirements are that a directed cycle can be formed between each tile pair, AGV movement to empty tiles is not restricted other than by layout constraints and the paths of each AGV in the system can be altered.*

Assume an AGV X is currently located at tile A and has destination tile B. By assumption, there is a cycle α including tile B and the empty tile in the layout, see Figure 4.3a. Lemma 1 shows that the empty tile can be located at tile B. So, without loss of generality, it can be assumed that tile B is the empty tile. By assumption, there also exists a directed cycle β in which tiles A and B are located. There is an empty tile in cycle β, since tile B is part of cycle β. Thus, Lemma 2 tells that AGV X can reach its destination, see Figure 4.3b. The previous procedure can be repeated sequentially for each AGV in the layout. Therefore, each AGV in the layout can reach its destination.



(a) Empty tile at destination via cycle α



(b) AGV X at destination via cycle β

Figure 4.3: AGV X can move to destination tile B once the empty tile is located at tile B.

Furthermore, it is easily concluded that one empty driveable tile in the layout is a necessary condition. No AGV has an empty tile in front if there are no empty driveable tiles in the layout. So, no AGV can move and thus reach its destination if there are no empty tiles in the layout.

Now that is known that each AGV can reach its destination, Corollary 1 extends the knowledge to the recoverability of a deadlock.

Corollary 1 *One empty driveable tile in a finite strongly connected directed layout is necessary and sufficient to recover from a deadlock. Requirements are that a directed cycle can be formed between each tile pair, AGV movement to empty tiles is not restricted other than by layout constraints and the paths of each AGV in the system can be altered.*

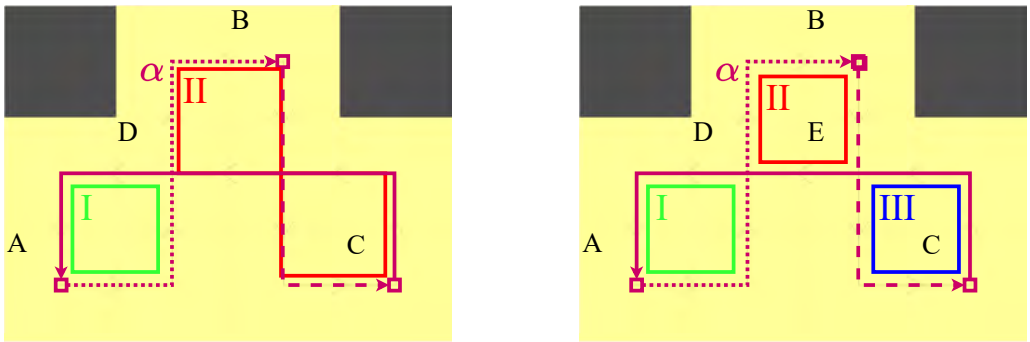
Given the mentioned requirements, each AGV is always able to reach its destination, see Proposition 1. This is consequently also the case if the system starts in deadlock. Thus, a deadlock can be recovered. Some or all AGVs that started in deadlock cannot reach their destination if this would not be the case.

4.2 Deadlock recoverability in a layout with non-cycles

If a cycle can be formed between each tile pair, Proposition 1 of Section 4.1 holds. The proposition of this section is for finite strongly connected directed layouts in which some tile pairs can only be connected by non-cycles in stead of cycles. A circuit is a non-cycle, if the circuit contains a tile, other than the start/end tile, that occurs more than once in the path. Such a path is a circuit, since it ends at its begin tile, but not a cycle. Showing recoverability gets more complex if not all tile pairs can from a cycle. Lemma 3 and 4 are therefore introduced below. Lemma 3 shows that a circuit can be split into a sequence of cycles. After that, Lemma 4 shows that two, instead of one, empty tiles are needed in the circuit for an AGV to follow a non-cycle circuit.

Lemma 3 *Directed circuit α can be fully defined by a sequence of directed cycles with a common tile. Requirement is that circuit α is in a finite strongly connected directed layout.*

Since circuit α is part of a strongly connected layout there is always a path between two tiles in that layout. Thus, circuit α can always be constructed by concatenating finite directed paths between tiles. Assume path between tiles A, B and C from circuit α, see Figure 4.4a. These paths are finite since the layout is finite. Circuit α either is a cycle or it is not. If circuit α is a cycle, the lemma can be concluded. If circuit α is not a cycle, this means that the directed circuit intersects itself. As a consequence at least one tile, other than the begin/end tile A, occurs more than once in the circuit. A list of all tiles where circuit α intersects itself can be made. This is done by counting how often a tile occurs in the circuit. Tiles, other than the begin/end tile A, are added to the list if they occur more than once. Since circuit α is not a cycle, there is at least one tile in this list. The circuit is split into subcircuits I and II that have a common tile, to decompose circuit α, see Figure 4.4a. Tile D is the common tile. The split is done by dividing circuit α into three consecutive parts. Part one is directed path P1 which starts at tile A and ends at the first occurrence of tile D. Part two is directed circuit II. Circuit II is a subcircuit of circuit α, starting and ending at tile D. Lastly, part three is directed path P2 from the last occurrence of tile D to the last tile in circuit α, tile A. Paths P1 and P2 can be concatenated to form directed subcircuit I, since the end tile of P1 is the same as the begin tile of P2 and vice versa. The circuits I and II are split at tile D, so the two subcircuits have a tile in common: tile D. Thus, each non-cycle circuit can be split into two subcircuits with a common tile.



(a) First split (b) Split of circuit II

Figure 4.4: Split of circuit α into a sequence of cycles.

Subcircuits I and II are cycles or not. If the subcircuits are cycles, the circuit is reduced to a combination of cycles. If a subcircuit, say circuit II, is not a cycle, circuit II can be split again into two subcircuits. The split is done with the same procedure as the split of circuit α. The procedure of splitting circuits into two subcircuits is repeated until each subcircuit is a cycle, see Figure 4.4b. This procedure is finite, since there is a finite number of intersection points. There

is a finite number of intersection points, since circuit α is composed of finite paths.

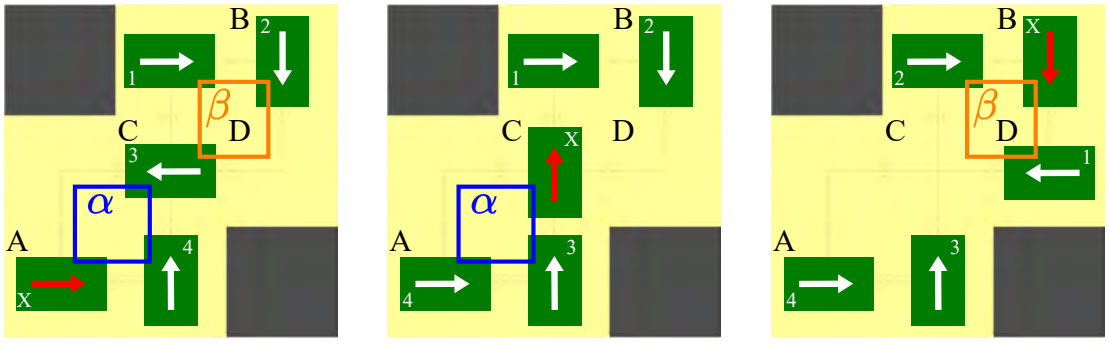
By the above procedure, each pair of subcircuits has a common tile, namely the tile on which the subcircuits are split. This means that in the end, each directed cycle has a common tile with at least one other cycle that is a subcircuit of the original circuit α. Thus, each directed circuit that is not already a cycle, can be written as a sequence of directed cycles. The circuit α can be followed by using segments of the found directed cycles. This can be done, since the segment sets of all cycles combined will be the same as the original segment set of circuit α.

To conclude, Lemma 3 shows that a non-cycle circuit can be decomposed in a sequence of cycles. Next, Lemma 4 shows how and under which conditions an AGV can follow a sequence of cycles.

Lemma 4 *AGV X which is part of cycle α can be located at any tile in cycle β, given that unique cycles α and β have at least one common tile and two empty tiles are present in cycle α and β combined.*

Cycle α and β contain at least two tiles, since no cycle can be made with one tile only. Therefore, the empty tiles are either both located in one cycle or there is one empty tile located in both cycles. If both empty tiles are located in one cycle, say cycle α one of the empty tiles can be moved to a common tile by Lemma 1. Once one empty tile is located on the common tile, this empty tile is now also part of cycle β. Now, each cycle contains an empty tile. So, without loss of generality, it is assumed that each cycle contains one empty tile, see Figure 4.5a.

To goal is to move AGV X from cycle α to any tile in cycle β. AGV X is located in cycle α either on a common tile or not. If AGV X is located on a common tile, it can move to any location in cycle β by Lemma 2. If AGV X is not on a common tile, the empty tile in cycle β needs to be located on a tile that is not a common tile, by the use of Lemma 1. There is at least one non-common tile in cycle β, since the cycles α and β are unique. Since AGV X is not located on a common tile, it will not move during the procedure of relocating the empty space in cycle β. If the empty tile of cycle α did move during this procedure, it can be moved back to a common tile by Lemma 1. There is now one empty tile in cycle α and one empty tile in cycle β, with the latter not on a common tile, see tile D in Figure 4.5a. AGV X can be located at a common tile of cycle α and β by using Lemma 2 to move AGV X in cycle α. The result is visible in Figure 4.5b. Once AGV X is located at a common tile it is part of cycle β. Since tile D is not part of cycle α, at least one empty tile remains part of cycle β after locating AGV X on the common tile. By Lemma 2, AGV X is able to move to any location in cycle β, see Figure 4.5c.



(a) Initial situation with one empty tile in each cycle.

(b) AGV X located at common tile C by moving along cycle α

(c) AGV X located at destination tile B by moving along cycle β

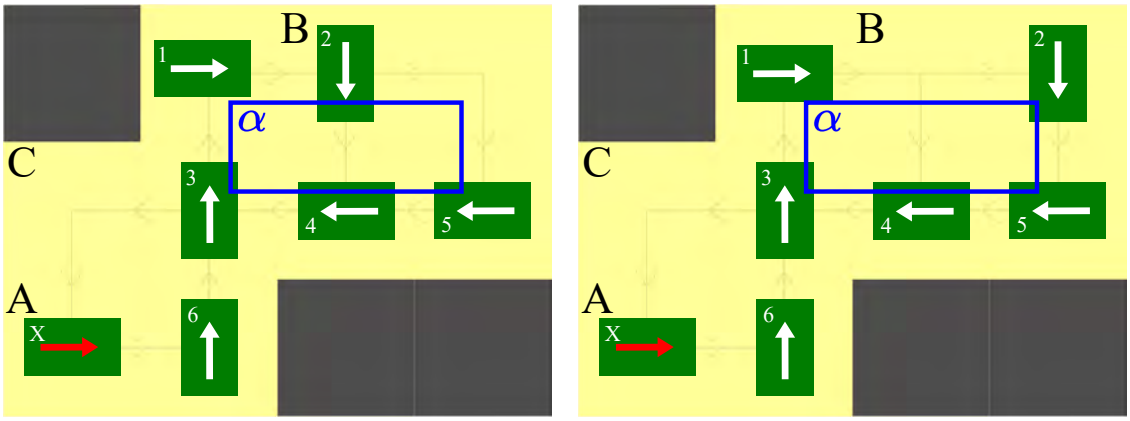
Figure 4.5: Cycles α and β both containing an empty tile and sharing a common tile C.

Lemma 4 shows that an AGV can follow a non-cycle circuit if there are two empty tiles location

in that circuit. Consequently, Proposition 2 shows that each AGV can reach its destination in a layout, even if there are some tile pairs for which only a non-cycle can be formed.

Proposition 2 *Two empty driveable tiles in the layout are sufficient to make sure each AGV can reach its destination. Requirements are a finite strongly connected directed layout, the movement of AGVs to empty tiles is not restricted other than by layout constraints and the paths of each AGV in the system can be altered.*

Assume an AGV X is currently located at tile A and has destination tile B, see Figure 4.6. Lemma 1 shows that the destination tile B can be emptied. So, without loss of generality, it can be assumed that one empty tile is located at tile B, see Figure 4.6.



(a) Starting situation (b) Empty tile located at destination tile B

Figure 4.6: Destination tile B can be emptied by following circuit α.

There is one other empty tile in the layout, located at tile C. A circuit, called circuit β, is constructed by concatenating the directed paths between tile A, B and C. This circuit contains AGV X and two empty tiles, see Figure 4.7. Lemma 3 shows that circuit β can be decomposed in a sequence of cycles, S, which are all connected through common tiles. This sequence consists of cycles I and II in the example in Figure 4.7. Lemma 1 tells that the empty tiles can move to any location within circuit β. Thus, the empty tiles can be located in the current cycle (I) of AGV X and the next cycle (II) of sequence S.

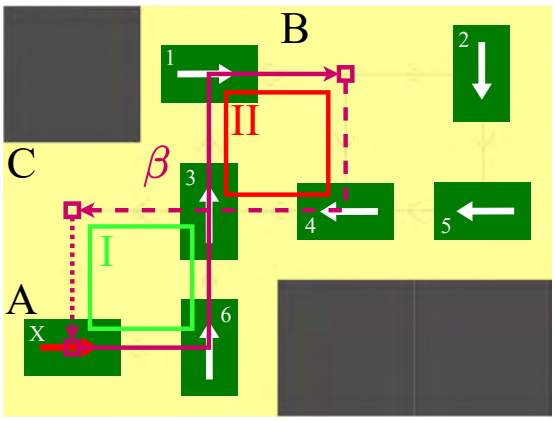
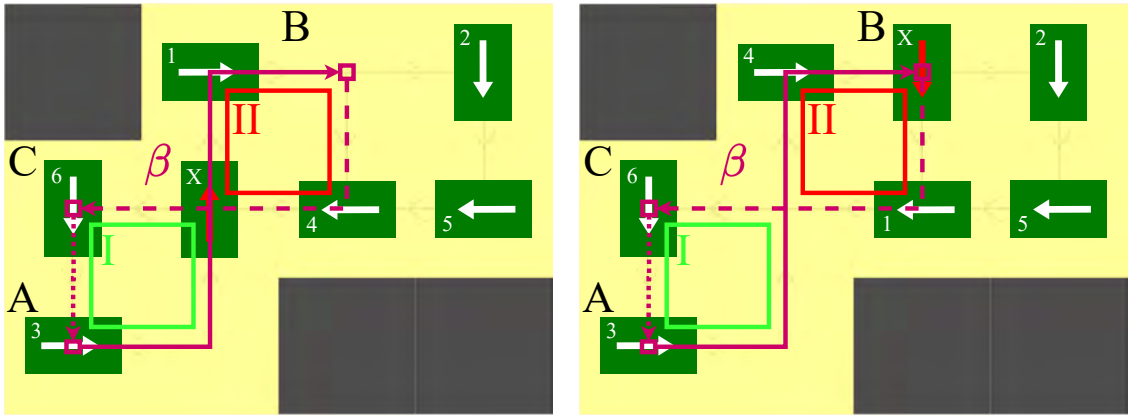


Figure 4.7: Circuit β decomposed into cycles I and II.

By Lemma 4, AGV X can proceed to cycle II of sequence S and reach its destination, see Figure 4.8. If sequence S would consist of more cycles, the empty tiles are consecutively relocated to the

current cycle of AGV X and the next cycle of sequence S. In this way, by using Lemma 4, AGV X can follow circuit β. Once AGV X is in the same cycle as destination tile B it can reach its destination by Lemma 2. The above procedure can be repeated sequentially for each AGV in the layout. Therefore, each AGV in the layout can reach its destination.



(a) AGV X travelled cycle I.

(b) AGV X travelled cycle II.

Figure 4.8: AGV X can reach destination tile B by following circuit β. This is done by following sequence S consisting of the decomposed cycles of circuit β.

Thus, Proposition 2 describes an algorithm for any AGV to reach its destination. Next, Corollary 2 continues on the proposition to show the recoverability of a deadlock.

Corollary 2 *Two empty driveable tiles in a finite strongly connected directed layout is sufficient to recover from a deadlock. Requirements are that the movement of AGVs to empty tiles is not restricted other than by layout constraints and the paths of each AGV in the system can be altered.*

Given the mentioned assumptions each AGV is always able to reach its destination, see Proposition 2. This is consequently also the case if the system starts in deadlock. Thus, a deadlock can be recovered. If this would not be the case, some AGVs that started in deadlock cannot reach their destination.

To conclude, Proposition 1 and 2 show algorithms to locate AGVs at their destination. Proposition 1 shows that at least one empty tile is needed to resolve a deadlock, if cycles can be formed between each tile pair in the layout. Two empty tiles in the layout are needed if some circuits between tile pairs are non-cycles. Keeping the requirements in mind, a deadlock can always be recovered by the presented algorithms. However, the algorithms are inefficient and would require large adaptations to the current AGV controls. Therefore, another deadlock recovery algorithm is found to actually implement in the Grid-Based Control model. The implemented deadlock recovery algorithm is elaborated in Chapter 5.

5 Recovery algorithm

Section 2.2 and Chapter 3 describe the Grid-Based Control model which is used during the project and the deadlocks that can occur. Furthermore, it is shown that deadlocks that occur in the model, can always be recovered if there are two empty tiles in the layout, using the algorithms of Chapter 4. But, the algorithms of Chapter 4 are very inefficient as one AGV at a time is driven to its destination. Therefore, there is a need for another recovery algorithm more suited for the Grid-Based Control model. An algorithm is found that fits better to the modular structure of the Grid-Based Control model. The algorithm is created after a literature study which is presented in Section 5.1. The resulting structure of the implemented algorithm is discussed in Section 5.2. An important feature of the algorithm is the use of prohibited segments. This feature is explained in Section 5.3. In some extreme cases, AGVs cannot reach their destination with the use of the recovery algorithm. These edge cases are discussed in Section 5.4. Lastly, the algorithm consists of multiple independent steps. The content of some independent steps are varied and the variations are presented in Sections 5.5, 5.6 and 5.7.

5.1 Literature

The phenomenon of deadlocks is not new. Deadlocks and deadlock handling techniques are studied since the 1960s as parallel computing emerges [12]. A deadlock can occur if parallel computations need to access the same memory. Starting from 1990, deadlocks in (physical) manufacturing systems are studied due to the rise of industrial automation [16–20]. The research of deadlock handling also expands to path planning and execution of AGVs, around the year 2000. The developed deadlock handling strategies for AGV systems only focus on deadlock prevention or deadlock avoidance [14]. No research is done on deadlock recovery of AGV systems as it is believed to result in lower performance as opposed to preventing and avoiding deadlocks. Deadlock recovery techniques applied to non-AGV systems are investigated to find a suitable deadlock recovery approach. Much inspiration comes from deadlock recovery strategies in automated manufacturing plants. These systems are physical systems, just as AGV systems. This means that a deadlock can only be recovered by breaking the circular wait condition of Coffman, see Section 3.1. At least one of the involved processes needs to re-plan their use of resources to break the circular waiting condition. There are different ways to make sure diverting is possible. Multiple deadlock recovery approaches based on breaking the circular wait condition are discussed below.

Dedicated deadlock infrastructure
Many researchers propose dedicated deadlock infrastructure to be able to break the circular wait condition. They use graph theory to detect a deadlock in manufacturing systems [17–20]. The deadlock is then resolved by placing one of the involved jobs in a deadlock buffer. This buffer cannot be used during normal operation which makes sure that diverting from the original planning is always possible. Yeh [21] also shows a graph-based real-time deadlock recovery approach for automated manufacturing systems, but assumes a buffer at every resource instead of one central buffer.

Apart from dedicated buffers, the circular wait condition can also be broken by rerouting over dedicated deadlock lanes. Lankes et al. [22] use this method in a Network on a Chip. When a router gets in a deadlock, one of the packages at the input side of the router is delivered through a deadlock channel. Moorthy et al. [15] propose the use of a dedicated deadlock lane in the context of zone-controlled AGVs. In both methods, the deadlock lane can be accessed from all other resources. The deadlock lane thus provides a way to reroute in case of a deadlock, so the circular wait condition can always be broken. Thirdly, Lehman et al. [23] study deadlock recovery methods for ship container terminals. A deadlock might consist of an AGV that needs to be unloaded by a crane, but the crane has a container that needs to be unloaded on the AGV. In that case, a dedicated deadlock AGV arrives to provide a space to unload.

Change purpose of resource to buffer
Next to using dedicated infrastructure and hardware to resolve deadlocks, one can also temporarily use resources as buffers. To have a resource to divert to, Wu and Zhou [24] discuss a deadlock resolution approach of a robotic manufacturing cell. In their method, robots are treated both as material handling devices and as temporary buffers in case of a deadlock. Zhou and Zhou [25] propose a deadlock recovery approach in a wafer fabrication plant that uses railed Overhead Hoist Vehicles to transport wafers. Part of the recovery method consists of using storage for finished wafers as temporary buffers to divert a wafer to.

Rerouting without dedicated infrastructure
Deadlocks can also be recovered by rerouting processes, without dedicated infrastructure or the use of buffers. Im et al. [26] explain a deadlock recovery method for material handling devices in a wafer fabrication plant. The plant is divided into segments. Each segment can have intersections with other segments. In the case of a deadlock, paths are evaluated of all deadlocked devices that are at an intersection. If a device is at an intersection and can divert, the path of that device is changed to break the circular wait condition. Fransen et al. [1] present a dynamic path planning approach for grid-based AGV systems. The path of an AGV is determined based on a graph representation of the grid, including weights on the nodes. The weights of nodes can change over time. The path of AGVs might change if the weights of the nodes have changed. Therefore, a deadlock might resolve if one of the involved AGVs gets a new path and the circular condition of the deadlock is broken. Lehmann et al. [23] studied deadlock recovery in a container terminal. Multiple deadlock scenarios are studied. Some of the deadlocks are resolved by replanning the order in which containers are loaded and unloaded by the crane. The container is chosen with the least (time) impact on the original schedule.

It is clear that one or more paths of involved AGVs need to be replanned to resolve a deadlock. This corresponds with breaking the circular wait condition of the involved AGVs. In manufacturing systems, it is common to use dedicated deadlock resources to be able to guarantee a redirecting option. In other approaches, no dedicated deadlock resources are used, but processes are altered based on the current system state. Deadlock recovery for the Grid-Based Controls model purely based on additional controls is preferred, as it does not cost any additional space and/or AGVs. The general deadlock resolving approach of Im et al. [26] is used. Literature shows that there should always be a resource to divert to, in this case, a tile. If no involved AGV can divert directly, the deadlock recovery algorithm should create a diverting option. When there is a diverting option (created), there should be some criteria to select which AGV(s) is/are going to divert. These criteria could be on different topics, such as the location of the deadlock in the grid or the remaining path of the deadlocked AGVs. Lastly, the path of an AGV must be replanned in such way that it chooses a deadlock-free path. The identified recovery steps are implemented in the algorithm structure which is explained in the next section.

5.2 Algorithm structure

In the current Grid-Based Controls model a deadlock detection algorithm is implemented based on the work of Van Weert [4]. This algorithm runs at a fixed time interval to detect any deadlocks. If a deadlock is detected this algorithm provides a list of involved AGVs. The moment the detection algorithm finds one or multiple deadlocks, the recovery algorithm is started. It is clear from Section 5.1 that the circular condition between involved AGVs needs to be broken to resolve a deadlock. This is done by giving at least one of the involved AGVs a different path. Providing a recovery path to one or more involved AGVs to resolve the deadlock(s) is the goal of the deadlock recovery algorithm. The algorithm succeeds in its task if the system is deadlock-free when the algorithm is finished. This section describes the found algorithm structure to do so.

The algorithm consists of multiple independent steps with fixed in- and output. The algorithm is split up based on the general actions which need to be taken to resolve a deadlock. By creating

independent sub-algorithms, each sub-algorithm can be improved separately without influencing the other sub-algorithms. Figure 5.1 shows the global structure of the algorithm in a flowchart and is elaborated below the figure. The variants of the general algorithm which are tested are described in Section 5.5, 5.6 and 5.7.

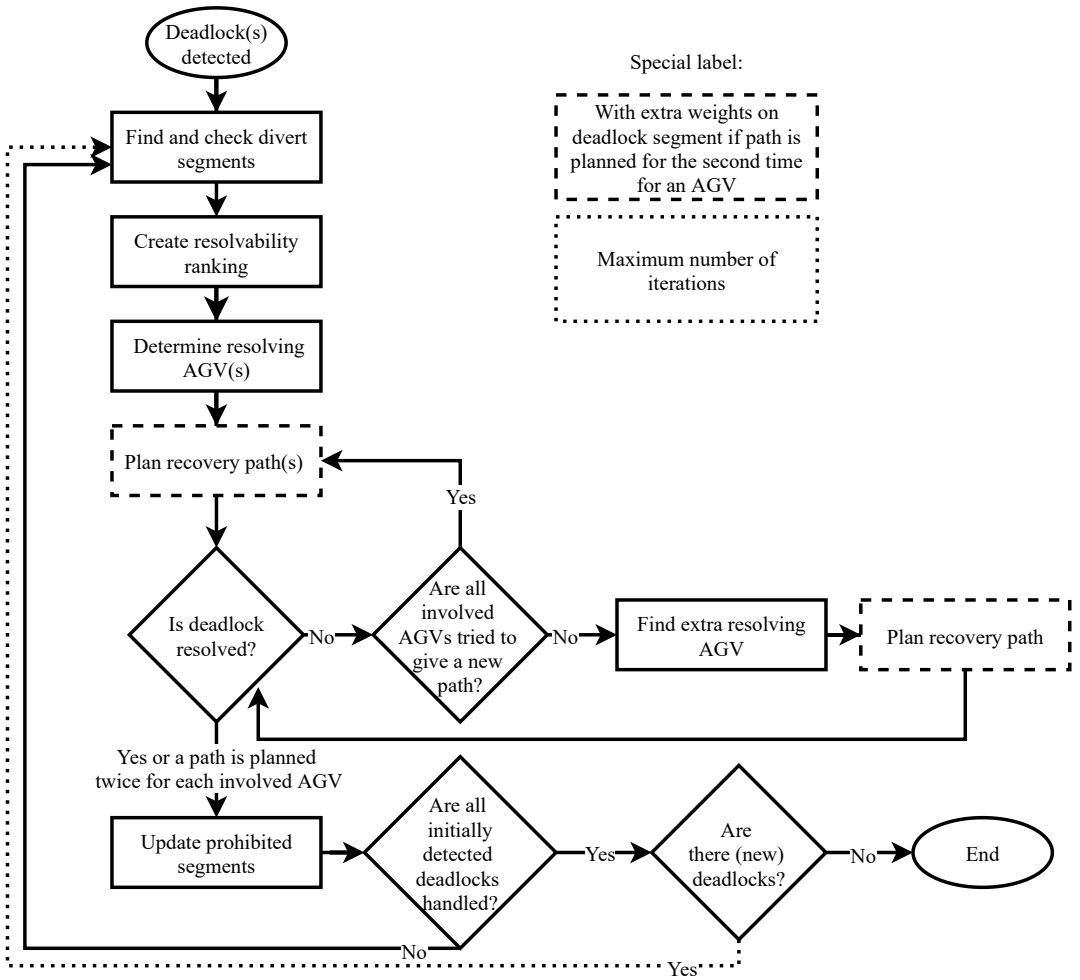


Figure 5.1: Flowchart of the general structure of the deadlock recovery algorithm.

The first main action of the algorithm is to determine which AGV(s) get(s) appointed as resolving AGV(s). A resolving AGV is an AGV for which a recovery path is searched to break the circular wait condition of the deadlock. The task of finding resolving AGV(s) is split into two parts. First, a ranking of involved AGVs is made. The ranking reflects which AGV is most suited to be resolving AGV. This ranking is called the resolvability ranking. All involved AGVs are placed in this ranking from highest to lowest resolvability score. Different methods can be used to create the resolvability ranking and are elaborated in Section 5.5. Secondly, a list of the actual resolving AGVs is determined by another function. This is done by picking some AGVs from the top of the ranking. The exact number of resolving AGVs can be varied, see Section 5.6. Thus, the output of the first part of the algorithm is a list with resolving AGV(s).

Next, the resolving AGV(s) get(s) a recovery path. For this step a path planning function based on Fransen [1] is used. In this method, the path with the lowest costs is chosen, based on the sum of three costs: node weights, segment weights and turn penalties. The node weights are dynamic

weights per tile based on how busy that tile is. The more an AGV stands still on a tile, the higher the dynamic weight. The segment weights represent the time it takes to travel from tile to tile. Lastly, the turn penalties take into account the extra time it takes to turn. Additionally, a temporary extra weight can be added during path planning. This temporary weight is used to penalise a path that would immediately result in a deadlock. An extra weight is added to the first segment of that path if a path would immediately result in a deadlock, after which path planning is run again. This procedure is repeated until either a deadlock-free path is found or a maximum number of segments is checked. If no deadlock-free path can be found, the initial optimal path is chosen.

Lastly, once a new path is tried to plan for all resolving AGVs, it is checked if the deadlock is resolved. This is done by running the deadlock detection algorithm again. The deadlock recovery algorithm is started again if the system is not deadlock-free. It can be the case that the resolving AGV(s) did not resolve the deadlock. This might, for instance, happen because there are no other outgoing segments from the current tile of the resolving AGV. An extra involved AGV has to be appointed as resolving AGV if the deadlock is not resolved. The two different methods to find an extra resolving AGV are described in Section 5.7. The methods provide one extra resolving AGV per time the function is called. Paths of consecutive extra resolving AGVs are altered until either the deadlock is resolved or a recovery path is tried to find for all involved AGVs.

The deadlock recovery algorithm always terminates. The algorithm terminates because of two measures. The first is that deadlock recovery of a specific deadlock is terminated if the deadlock cannot be resolved. This can happen in very specific cases related to path planning, as is explained in Section 5.4. Next to that, there is a maximum number of consecutive iterations of the deadlock recovery algorithm. The limit on the number of checks prevents an infinite loop.

5.3 Prohibited segments

The goal of the deadlock recovery algorithm is to deliver a deadlock-free system. However, it might happen that a deadlock cannot be resolved without introducing a new deadlock. The use of prohibited segments makes sure that the system becomes deadlock-free, even if new deadlocks need to be introduced. Prohibited segments are also used to prevent infinite recovery loops. This section explains how prohibited segments are applied.

First, the notion of deadlock and divert segments is shortly introduced before the feature of prohibited segments is explained. Each involved AGV has only one deadlock segment. A deadlock

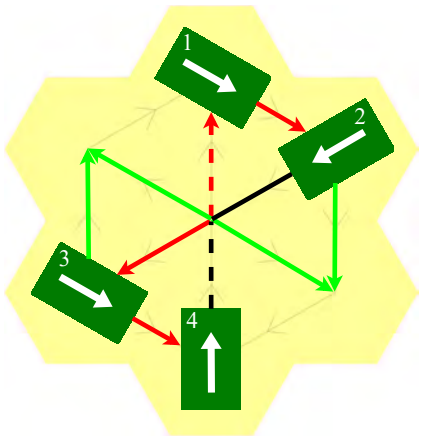
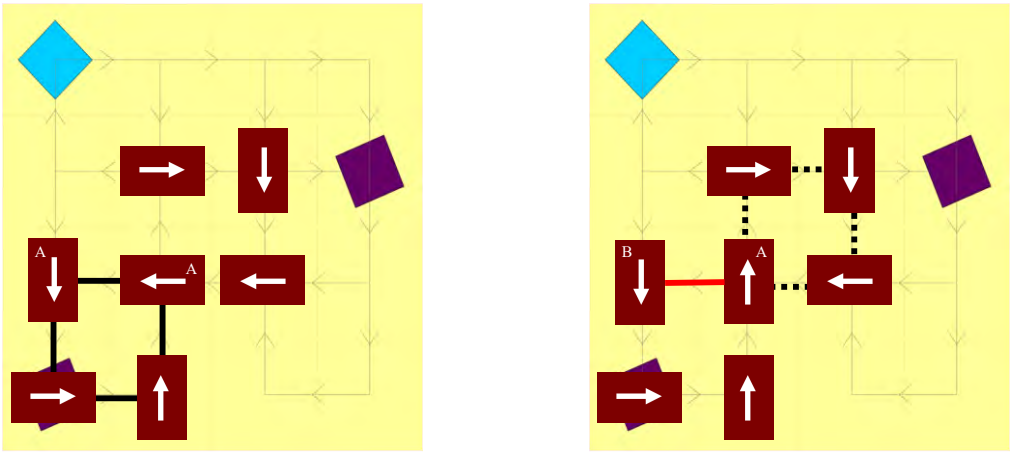


Figure 5.2: A multicycle deadlock with deadlock segments (red) and divert segments (green). Segments to empty tiles are skipped as deadlock segment.

segment of an AGV is the segment leading to the next involved AGV in the deadlock. Figure 5.2 shows the deadlock segments of an example deadlock in red. The deadlock segment of AGV 4 is the dotted red arrow and note that the empty tile is skipped. Divert segments are possible outward segments that might be used in a deadlock resolving path. As an example, the divert segments of Figure 5.2 are coloured green. All divert segments of a deadlock can be found by listing all direct outward segments of the deadlock cycle.

The first use of prohibited segments is to force recovery of a deadlock. Figure 5.3 shows an example of a situation in which resolving one deadlock creates a new one. AGV A is the only AGV that can resolve the deadlock. But, the path planner will not choose a different path, since the alternative path also leads to a deadlock. Prohibited segments are used in this case. A temporary extra weight is assigned to that segment if a segment is prohibited. The weight is higher than the penalty for a standard deadlock. In this way, a path with a new deadlock is preferred over a path including the original deadlock. Prohibited segments are only used if there are no deadlock-free alternatives. In Figure 5.3b the segment between AGV A and B is made prohibited, so AGV A is forced to take a path without that segment. Creating the new deadlock gives the opportunity to make the system deadlock-free. This opportunity would not be there if the original deadlock of Figure 5.3a is kept intact.



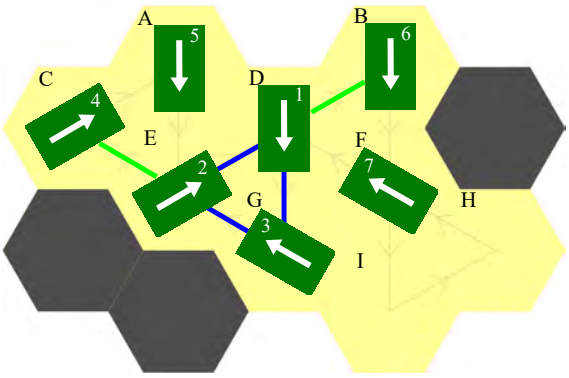
(a) Initial deadlock (b) Newly introduced deadlock

Figure 5.3: Example in which resolving a deadlock (solid line) results in another deadlock (dotted line) due to a prohibited segment (red).

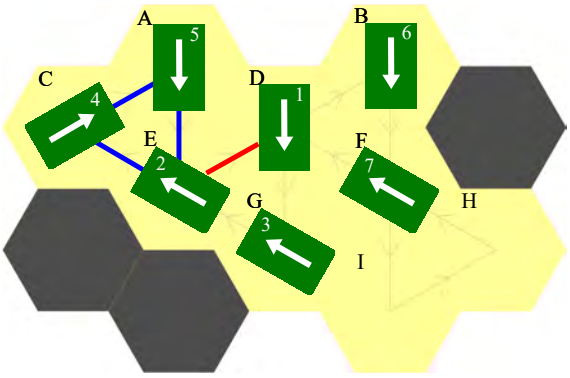
Iteratively resolving newly introduced deadlocks creates the risk of an infinite recovery loop. Such a situation occurs when AGV A in the example of Figure 5.3b gets a new path during the recovery of the new deadlock (dotted line). When AGV A gets a new path the original deadlock (solid line) is restored. The second use of prohibited segments is to prevent such an infinite recovery loop. After a deadlock is resolved, the deadlock segment of resolving AGV becomes prohibited. The segment becomes prohibited no matter if prohibited segments were needed to resolve the deadlock. The previous deadlock will not be restored if a new deadlock is introduced. The previous deadlock is not restored, because the path resulting in the original deadlock would contain a prohibited segment. When the system is deadlock-free at the end of a recovery call, the list with prohibited segments is reset.

In one situation a prohibited segment becomes unprohibited before the end of a deadlock recovery call. Before a deadlock is resolved, the divert segments of the considered deadlock are listed. It might happen that all divert segments of a new deadlock are prohibited. An example is shown in

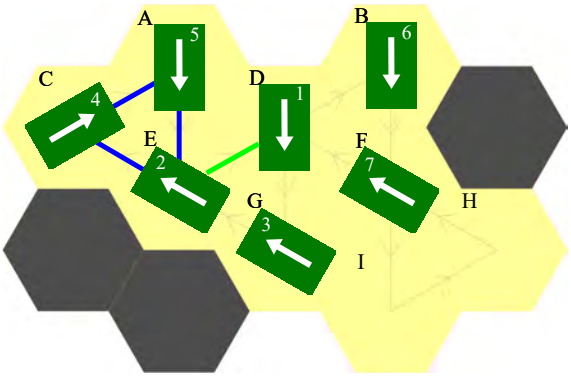
Figure 5.4b as the result of resolving the deadlock of Figure 5.4a.



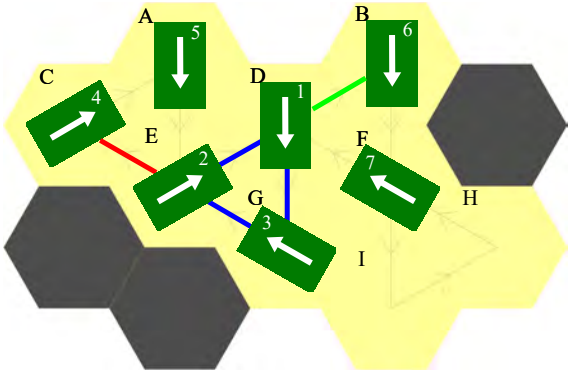
(a) Initial deadlock (AGV 1,2,3), two unprohibited divert segments.



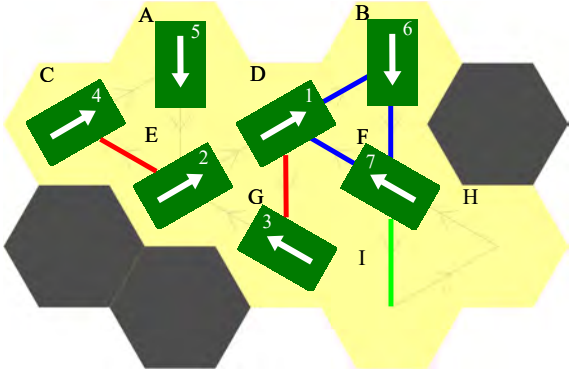
(b) New deadlock (AGV 2,4,5) introduced. Segment ED becomes prohibited.



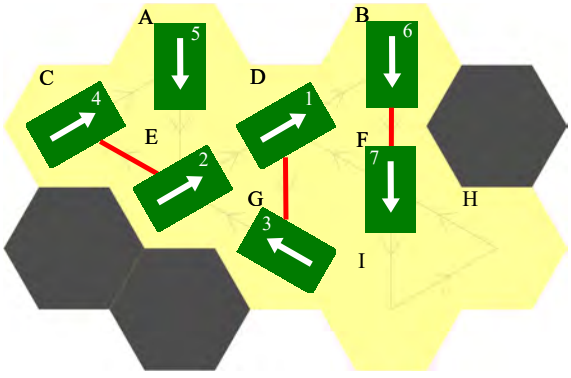
(c) Segment ED becomes unprohibited as all divert segments were prohibited



(d) Original deadlock (AGV 1,2,3) is restored. Segment CE becomes prohibited, only one unprohibited divert segment left.



(e) New deadlock (AGV 1,6,7) introduced. Segment DG becomes prohibited.



(f) System is deadlock free.

Figure 5.4: The use of prohibited segments (red) making sure a deadlock free solution is found. Unprohibited divert segments are marked green.

A divert segment needs to be made unprohibited, to resolve the new deadlock and find a deadlock-free solution. The divert segment that is made unprohibited, is chosen based on a list in which all newly created deadlocks are stored. Deadlocks are stored including the deadlock segment of the resolving AGV of the previous deadlock, that created the new deadlock. The deadlock is looked up in the list and the corresponding segment is made unprohibited if all divert segments of a

deadlock are prohibited, see Figure 5.4c. The previous deadlock gets restored by this action, but with one important difference. One extra divert segment of the deadlock is now prohibited, see Figure 5.4d. So, when the original deadlock is resolved for the second time, another AGV will be resolving AGV, see Figure 5.4e.

The use of prohibited segments creates a very important property of the recovery algorithm. It makes sure that the system is, in principle, always deadlock-free after a call of the deadlock recovery algorithm. The algorithm introduces a depth-first search to an AGV that can divert deadlock-free. By the procedure, each divert segment is systematically visited to find an AGV that can divert with a deadlock-free path. A divert segment becomes prohibited in the end if using that segment does not yield a deadlock-free solution. It has the result that the corresponding branch of AGVs is not visited again.

The use of prohibited segments prevents recovery loops. Another method, to prevent loops, is used in an earlier version of the recovery algorithm using prohibited AGVs. An AGV is marked prohibited if it resolves a deadlock. The prohibited AGV cannot be chosen as resolving AGV during the recovery of a new deadlock, in the same call of recovery algorithm. The use of prohibited AGVs works properly if a tile has a maximum of two outward segments. But, for instance, hexagonal tiles can have more than two outward segments. Imagine an AGV is on a tile with three outward segments. The AGV is in a deadlock with a path that uses the first outward segment. The deadlock is resolved by giving that AGV a new path, including the second outward segment. Marking the AGV as prohibited, excludes the use of the third outward segment, although the third segment might be necessary to resolve a newly created deadlock. Prohibited segments are in the end used to prevent this problem.

Despite the use of prohibited segments there are some situations in practice in which the system is not deadlock-free at the end of the recovery algorithm. This is due to implementation flaws. The limitations are further discussed in Section 5.4.

5.4 Reachability guarantee

As described in Section 5.3, the use of prohibited segments and sequentially resolving deadlocks makes sure the system is deadlock-free at the end of the deadlock recovery algorithm. Nonetheless, there are two characteristics in the practical implementation of the algorithm that might cause that the system does not become deadlock-free. The use of the deadlock recovery algorithm also introduces a risk of livelocks. The three characterises discussed in this section have the consequence that no absolute guarantee can be given that each AGV reaches its destination.

The first characteristic relates to path planning. The use of prohibited segments should enforce that for each divert segment a path is planned, if needed. However, it might happen that a divert segment only ends up in the recovery path if the path contains a loop. Figure 5.5 shows a situation in which not all divert segments are used in a recovery path. The deadlock, shown in blue, has two divert segments. The use of both divert segments results in a new deadlock. To force progress, one of the deadlock segments, segment DE, becomes prohibited. This should enforce that AGV 2 gets a different path. However, AGV 2 always has to use segment DE to reach destination tile I. The use of prohibited segment DE is very expensive, but adding the intended divert segment CE to the path always results in even higher costs. Thus, the unprohibited divert segment CE is not used in a recovery path. This problem can be solved by planning paths via a third tile, tile C in the example. This solution is not implemented as the flaw did not give any situation in which the system could not be deadlock-free. The layouts used at Vanderlande do not contain areas that can only be reached by using one specific segment which reduces the risk that the system cannot become deadlock-free. Only AGVs that are one tile away from their destination experience this path plan characteristic.

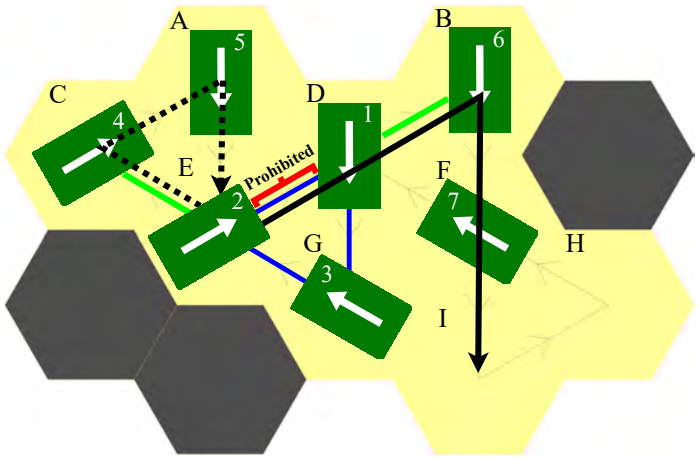
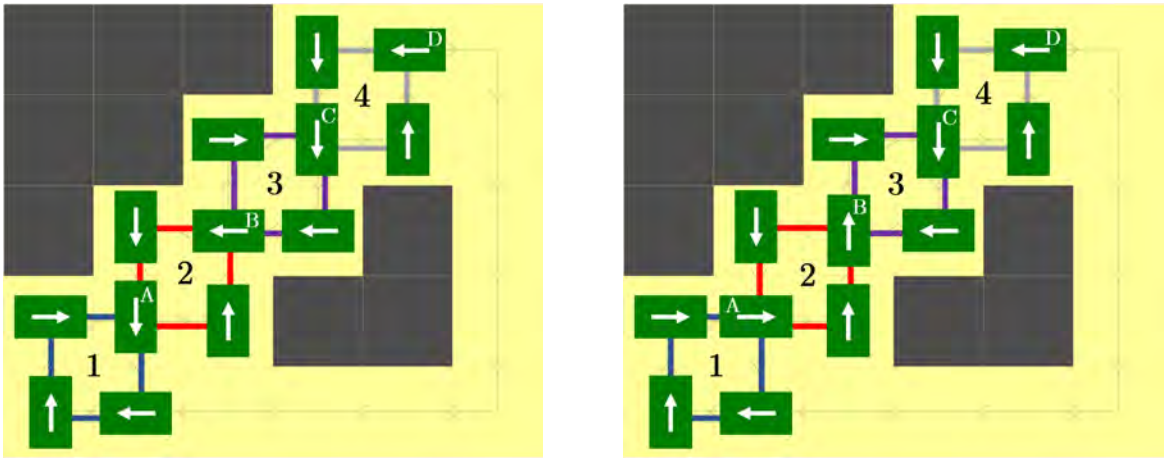


Figure 5.5: Deadlock (blue) with two divert segments (green). Segment CE is only added to the path of AGV 2 (solid line) if a loop if added (dotted line). There is no path to destination tile I without the prohibited segment DE.

Secondly, new deadlocks might be introduced in the process of finding an AGV that can get a deadlock-free recovery path. The number of extra created deadlocks is finite, but can be large in dense layouts. The consequence is that deadlock recovery takes a long time. Routine tasks are paused and processes can come to a stop if the central controller is occupied with deadlock recovery. Sending way-points to AGVs that are not involved in the deadlock is one of these routine tasks. There is a maximum number of times that newly created deadlocks are resolved within one deadlock recovery call, to prevent overloading the central controller. The unresolved deadlocks are detected and resolved with the next deadlock detection action. However, ending the deadlock recovery algorithm before the system is deadlock-free, can create an infinite recovery loop. This edge case is illustrated by the example in Figure 5.6.



(a) Begin situation

(b) Situation after recovery algorithm ended

Figure 5.6: Example of an infinite recovery loop. The coloured boxes represent (possible) deadlocks. The arrows on AGVs indicate the driving direction.

Assume that the recovery algorithm is configured to at most resolve one newly created deadlock. Figure 5.6a shows the beginning situation with Deadlock 1 in the bottom left corner of the layout. Deadlock1 can only be resolved by rerouting AGV A. This recovery action introduces Deadlock 2. The previous deadlock segment of AGV A becomes prohibited. Therefore, Deadlock 2 can only

be resolved by rerouting AGV B. Now Deadlock 3 is introduced, but the maximum number of recovery iterations is reached. So, Deadlock 3 is not recovered during the first call of deadlock recovery. All AGVs remain in position as shown in Figure 5.6b. Deadlock 3 is detected with the next detection cycle, and recovery is started for this deadlock. Deadlock 3 can be recovered by rerouting AGV B or C. An infinite recovery loop occurs if AGV B is chosen as resolving AGV. Deadlock 2 is restored if AGV B is given a new path. AGV A will be the recovery AGV for Deadlock 2, since the previous deadlock segment of AGV B becomes prohibited. At the end of this second deadlock recovery call, the system ends up in exactly the same state as started.

AGVs do not move and reach their destination if a deadlock recovery loop occurs. AGVs also do not reach their destinations if a livelock happens. In that case, an AGV moves but does not make any progress. A livelock might happen if an AGV is appointed as resolving AGV at exactly the same location as it was a resolving AGV before. There is a chance that cyclic behaviour is introduced causing the AGV to never reach its destination. The chance that a livelock occurs can be decreased by introducing randomness in the deadlock recovery algorithm. This might be done by using a random resolvability ranking once in a while, or flipping the created ranking upside down once in a random number of recovery calls.

The edge cases discussed in this section prevent AGVs from reaching their destination, either because a deadlock can not be recovered, an infinite recovery loop happens or a livelock occurs. The edge cases are caused by the algorithm structure. The structure including the use of prohibited segments is fixed. However, some parts of the algorithm can be changed. The parts of the algorithm that can change are next discussed in Section 5.5, 5.6 and 5.7.

5.5 Resolvability ranking

Creating the resolvability ranking is one of the independent functions in the deadlock recovery algorithm. The resolvability ranking of involved AGVs can be made based on many different system parameters and calculations. This section discusses the different methods there are found to create a resolvability ranking. The resolvability ranking methods which are presented are distilled from studying deadlock situations and manually trying to resolve them.

Longest remaining path
For this ranking method, the length of the remaining path is decisive. The more remaining tiles an AGV needs to travel to its destination, the higher it ends up in the resolvability ranking. The idea behind this method is that AGVs with a long remaining path, generally have a destination far away. By choosing such an AGV as resolving AGV, the AGV is sent away from the congested area in which a deadlock typically occurs. Secondly, rerouting an AGV which has a long remaining path probably has a smaller influence on the remaining duration of the path as opposed to an AGV which is almost at its destination. Lastly, this method is tested since it requires no costly computations. The length of the remaining path is easily extracted from the model. A possible downside of this method is that the consequence for the resolving AGV(s) is not known and taken into account.

Most used tile
In general, a certain global flow can be identified within a layout. Such a flow can, for instance, be found around a pick-up point, where AGVs have to arrive from a specific direction. In such a situation, there is an inbound flow of unloaded AGVs and an outbound flow of loaded AGVs. A deadlock can occur if a loaded AGV X wants to cross the flow of unloaded AGVs. AGV X is essentially blocking the entire flow of AGVs to and from the pick-up point. Therefore, it would make sense to remove this blockage and thus appoint AGV X as resolving AGV. The Most Used Tile method tries to identify which AGV is blocking the most other AGVs.
To identify the 'blocking' AGV, the Most Used Tile method first creates a list with the tiles on which involved AGVs are located. These tiles are called involved tiles. Secondly, it is checked how often an involved tile occurs in all remaining paths of involved AGVs. The resolvability ranking

is made based on the occurrences of the involved tiles. The AGV that is located on a tile with the most occurrences ends on top of the ranking. So, the idea of this method is to restore the flow in the system by removing AGVs that hinder the most other involved AGVs. A downside of the method might again be that the consequence for the resolving AGV is not considered.

Last in queue
It is clear that areas around handling points are vulnerable for congestion and therefore deadlocks. A deadlock around a handling point is often created when the inbound and outbound flow of that handling point cross. Since multiple AGVs are travelling to the same handling point, multiple AGVs have the same destination. The Last In Queue method tries to find virtual queues for destinations to relocate them. In that way, room is created to let other AGVs pass. Of all AGVs travelling to the same handling point, it makes sense to choose the AGV with the longest remaining path to the destination as resolving AGV. This AGV has relatively the most time to divert before it would arrive at the handling point.
First the destination of each AGV is checked and counted to make the ranking. This creates a ranking of destinations, in which the destination that occurs the most comes on top. The place in the ranking is randomly divided if destinations have equal occurrence. Next, for each unique destination a sub-ranking is created. This sub-ranking ranks AGVs with the same destination from the longest remaining path to shortest remaining path. The full resolvability ranking is made by concatenating the sub-rankings of each destination. In the resulting resolvability ranking, the involved AGV with the longest remaining path going to the most popular destination, ends up first in the ranking.

Least extra cost path
The above ranking methods all use the current path of involved AGVs to determine the resolvability ranking. This means that the calculation of possible recovery paths is not needed. It eases computations, however the consequence for the resolving AGV(s) is not taken into account. This might result in recovery paths which are, for instance, much longer than the original path or go through a very congested area.
The Least Extra Cost Path resolvability ranking method uses the relative extra cost of the recovery path as a ranking value. The use of costs of a recovery path prevents an unfavourable recovery path. The AGV with relatively the least extra cost for its recovery path ends on top of the ranking. The cost of the current remaining path is first calculated. This is done based on node weights, segment weights and turn penalties which makes up the costs for path planning, see Fransen et al. [1]. The path planner tries to plan a new deadlock-free path after the costs of the current path are calculated. The costs of the new path are compared to the old path. The involved AGV with the least increase or even a decrease in path costs ends up on top in the resolvability ranking. An involved AGV is placed at the bottom of the ranking if it cannot get a deadlock-free path.

5.6 Number of resolving AGVs

A second parameter that can be adjusted in the deadlock recovery algorithm is the number of involved AGVs to give a recovery path. A deadlock is resolved by letting one or more AGVs divert from their original path. One AGV that diverts is enough to resolve the deadlock. However, giving multiple AGVs a new path might reduce congestion sooner. Reducing congestion improves the flow in the area of the deadlock and decreases the chance of direct new subsequent deadlocks. Another possible gain of labelling multiple AGVs as resolving AGVs is that it saves computation time. No extra recovery AGV has to be found if the deadlock is resolved after assigning new paths to multiple resolving AGVs. The number of resolving AGVs is determined by a percentage of the number of involved AGVs. The percentage can be varied in the settings of the algorithm. It is made sure that there is always at least one resolving AGV.

5.7 Extra resolving AGV

The last independent step in the recovery algorithm that can be varied is finding an extra resolving AGV. A check is performed to see if the deadlock is resolved, after the path planner did try to plan a recovery path for each initial resolving AGV. It can be the case that the deadlock is intact after planning recovery paths. This can happen if, for instance, the layout prevents the resolving AGV from diverting. Consequently, the path of a different involved AGV has to be replanned. To find this extra resolving AGV, two methods are found.

Next in path
The Next In Path method is introduced to clear the way for the initial resolving AGVs. The initial resolving AGVs are on top of the resolvability ranking. Thus, by some measure, it is expected that these AGVs are most suitable to give a recovery path. If the resolving AGV(s) did not resolve the deadlock, they are apparently blocked by other AGVs or layout constraints. The Next In Path method continues the path of a resolving AGV until the next involved AGV in the deadlock if found. The AGV encountered in the path is chosen as an extra resolving AGV for which a recovery path is planned. The idea is to remove the blocking. The path of the extra AGV is continued to, again, find an extra resolving AGV if the extra AGV did not resolve the deadlock.

Next in ranking
The second method to find extra resolving AGVs uses the resolvability ranking. The ranking tells how suited an involved AGV is to give a recovery path. So, the resolvability ranking should provide the best candidate of the AGVs which are previously not chosen as resolving AGV. Therefore, the Next In Ranking method picks the first AGV in the ranking for which no recovery path is searched yet.

This chapter shows the general structure of the implemented deadlock recovery algorithm. Parts of the algorithm are elaborated for which different variants are designed. Next to that, the use of prohibited segments is discussed, as well as three edge cases in which AGVs do not reach their destination. Exploratory simulations are performed for all algorithm variants. The simulations give insight into the effectiveness of each variant and the importance of specific design choices. The set-up of these exploratory simulations and the results are discussed in Chapter 6.

6 Exploratory simulations

The goal of the deadlock recovery algorithm is to deliver a deadlock-free system. As described in Section 5.2, there are three parts of the algorithm for which different variants are made. Besides variants of the deadlock recovery algorithm, the length of the deadlock detection interval is also expected to influence the system performance. It is expected that each deadlock can be recovered, no matter which variant of the deadlock recovery algorithm is used, due to the algorithm structure. The system performance and computation time are hence the relevant differences between algorithm variants. The goal of simulations is to find the algorithm variant that gives the best system performance and investigate the computation time of the variants. Congestion and hinder is the main reason that there is a limit on the maximum performance of an AGV system. Algorithm variants that are the best in lowering congestion, should therefore give the best system performance.

Previous Sections 5.5, 5.6, and 5.7 describe why variants of the recovery algorithm can perform better than others in specific situations. But, predicting which and how deadlocks exactly occur is impossible due to the randomness in the AGV system and interaction with other AGV control algorithms. Thus, it is difficult to predict which algorithm variants provides the best system performance over the long run, purely based on their design. Simulating algorithm variants and evaluation of the results is therefore key to understand the most important contributions to the best performance. Simulating and testing all the different settings on full-scale layouts takes too much time. Therefore, exploratory simulations are done on a smaller layout. The exploratory simulations are used to get a feeling for the effectiveness of each algorithm variant and how it combines with the other control algorithms. A small selection of algorithm variants is chosen to simulate on full-scale layouts, based on the finding of the exploratory simulations. The layout used for the exploratory simulations can be seen in Figure 6.1. Each different layout creates a different AGV flow in the system. That is why the location and size of deadlocks is dependent on the layout. The test layout is made to represent as much of existing layouts as possible by the combination of layout characteristics. The bottom part resembles a layout that is characterized by many intersections and multiple drop-off tiles per drop-off location. The top part of the layout relates to layouts with big loops and the costs of an alternative path is high.

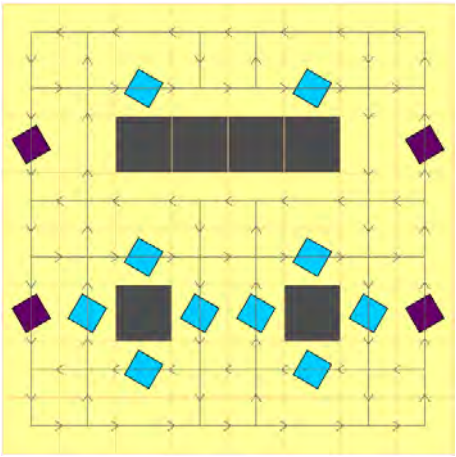


Figure 6.1: Layout which is used for the exploratory simulations.

In the remainder of this chapter, first the performance measures are explained in Section 6.1. In Section 6.2 the test parameters are elaborated. The system behaviour for the different algorithm variants are discussed in Section 6.4 and 6.5. The effect of changing the deadlock detection interval is shown in Section 6.6. Section 6.7 shows the computation time of the algorithm. Lastly, a conclusion on the exploratory simulations is draw in Section 6.8.

6.1 Performance measure

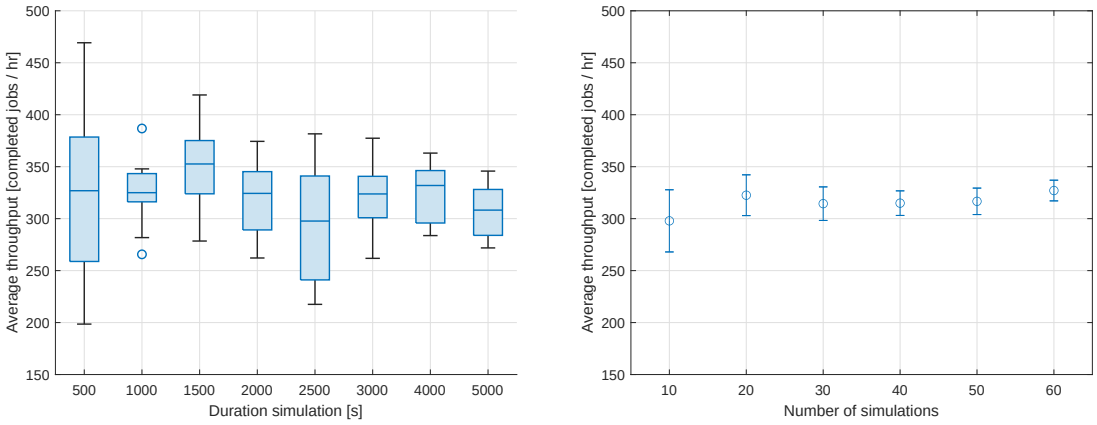
The algorithm variants are compared by the results of simulations. There are three main performance measures used to evaluate the variants.

- Throughput [jobs / hr]: the number of completed jobs per hour. A measure for the achieved capacity of an AGV system.
- Computation time [s]: the computation time it takes to complete a (sub-)algorithm.
- Resolving AGV ratio [-]: the ratio of recovery actions for which the initially found resolving AGV(s) did resolve the deadlock and no extra resolving AGV needed to be found. This measure gives insight into how often a suitable involved AGV ends on top of the resolvability ranking.

Next to the throughput, computation time and ratio some other metrics are logged. The number of deadlocks that occurred during deadlock and deadlock size are two of them. It is also logged at which point in the algorithm a deadlock is resolved. The extra metrics are used to understand the performance of the algorithm variants and their interaction with other control algorithms.

6.2 Simulation set-up

A proper duration of each simulation and the number of simulations need to determined to find reliable results for the performance measures of Section 6.1. First, the length of a single run is checked. Sets of ten simulations are started with different simulation lengths to compare the mean throughput. The goal of these simulations is to check for start-up behaviour at the beginning of the simulation. The simulations are done with 30 AGVs without deadlock avoidance. A random resolvability ranking is used as this is a baseline to compare the other recovery variants. Figure 6.2a shows the results of the simulations with different lengths. Ten runs are not enough to say if the data is distributed according to a normal distribution. Thus, the results are shown in boxplots instead of confidence intervals. Boxplots show the full range of data points divided into four quartiles, including the median of the data between the second and third quartile. The boxplots in Figure 6.2a for relatively short runs (500s/1000s) do not show a noticeable offset in throughput. It is concluded that start-up behaviour does not influence the mean throughput considerably for runs longer than 500s. On the other hand, increasing the length of the simulation decreases the



(a) Simulation duration

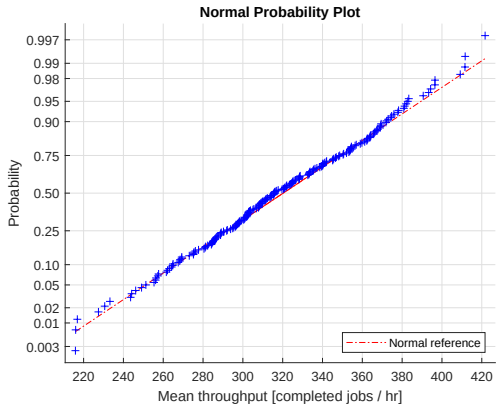
(b) Number of simulations

Figure 6.2: Figures on simulation duration and number of simulation for the exploratory simulations.

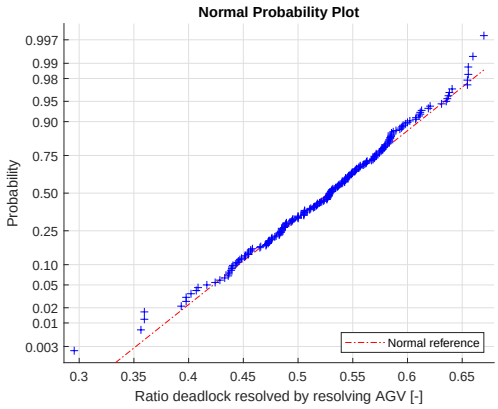
spread of the mean throughput. Therefore, a simulation length of 2000s is chosen for the the simulations.

Next, the number of simulations per setting is determined. The grouped mean throughput per combination of different number of runs is shown in Figure 6.2b by means of confidence intervals. Contrary to boxplots, confidence intervals do not show the real range of the used data. A confidence interval shows an estimated range for the true value of a parameter. The confidence intervals in Figure 6.2b on the mean throughput are created with student-t distribution and are relatively small. The number of runs per simulation is set on 30 runs, as this is expected to have a proper balance between run time and accuracy.

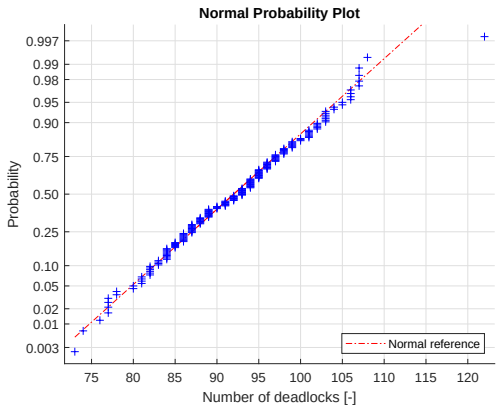
Figure 6.2b shows confidence intervals, since the mean throughput is assumed to be normally distributed. This is assumed after performing a Lilliefors test on normality and visually inspecting the mean throughput of the 210 simulations in a normal probability plot shown in Figure 6.3a. The data of the considered performance measure lies on the straight reference line if the data is normally distributed. Normality is checked in the same way for the two other performance measures: the number of deadlocks during a simulation and the resolving AGV ratio. Figure 6.3b and 6.3c show the normality plots of the latter two measures. It is assumed that these measures are also normally distributed. The spread of the performance measures in the rest of the report is therefore visualised using 95%-confidence intervals.



(a) Mean throughput



(b) Mean resolving AGV resolving the deadlock ratio



(c) Mean number of deadlocks

Figure 6.3: Normal probability plot comparing the performance measures based on 210 runs to a normal distribution.

6.3 Implementation flaws

During exploratory testing of the recovery algorithm, two unexpected flaws are discovered. Deadlocks might not be resolved as a consequence of the flaws. The flaws originate from reusing of the existing path plan function. The two flaws and their fixes are elaborated on below.

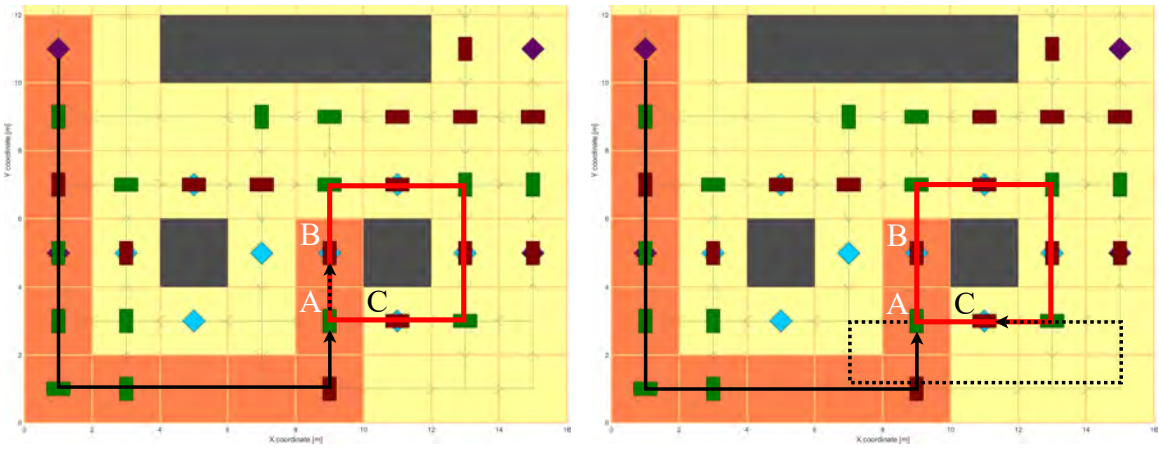
Implementation flaw, deadlock avoidance during path planning
The deadlock recovery algorithm is implemented in the existing Grid-Based Control model. To prevent duplicate code, path planning during deadlock recovery is executed by an existing path planning function. This path planning function is originally built to periodically reroute paths of AGVs as a part of the dynamic path planning of Fransen [1]. The path planner searches a deadlock-free path from the current tile of the AGV. The newly found path is ignored if it leads to a deadlock. The deadlock check during path planning is executed by the deadlock avoidance algorithm. The use of deadlock avoidance is troublesome when the reroute function is used for the deadlock recovery.

The problem is illustrated by the following example. Assume the model uses monocycle deadlock avoidance. This makes sure that no monocycle deadlocks occur during simulation. However, monocycle avoidance can introduce two-cycle deadlocks as stated in Section 3.3. Therefore, the deadlocks that deadlock recovery has to recover are two-cycle deadlocks. During deadlock recovery, the path planning function is called for an involved AGV. It is checked if the newly calculated path creates a deadlock. However, this check is done by deadlock avoidance, which is on monocycle level. Thus, the two-cycle deadlock, for which the recovery algorithm is called, is not detected. This can lead to the situation in which the original path of an AGV, and thus the deadlock is kept intact, despite there is a path available that resolves the deadlock.

The deadlock that needs to be recovered can only be found with the deadlock detection algorithm, not the deadlock avoidance algorithm. Thus, the deadlock check during path planning is altered. The deadlock detection algorithm is used whenever path planning is called during deadlock recovery. With the use of deadlock detection, the original path, which causes the deadlock, is not chosen again if there is a deadlock-free alternative path. The deadlock avoidance algorithm is still used to predict deadlocks when path planning is called in the standard path planning. This ensures that paths are not discarded unnecessarily.

Implementation flaw, loop detection during path planning
The use of the existing path planning function introduces another implementation flaw. Paths are checked for loops after they are planned. Loops are not allowed due to possible bookkeeping errors within the simulation. A new path is thrown away if a loop is detected. By this check, a suitable and needed recovery path might be disregarded. It can even occur that all recovery paths create a loop, and the deadlock is not recovered at all. An example of a loop in a recovery path is shown in Figure 6.4. The figure shows a deadlock in which the AGV located at tile A is assigned as resolving AGV. The AGV is only one tile away from its destination tile B. Figure 6.4b shows the found deadlock-free recovery path. The AGV travels to the same drop-off chute, but at another location: tile C. The new path creates a loop with the already traversed path of the AGV. Consequently, the deadlock-free path is disregarded and the original path is restored. This has the result that the deadlock is not recovered. This flaw is fixed by adapting the controls so it could handle with loops in paths.

The Grid-Based Controls model is adapted to fix the unexpected implementation flaws. The exploratory simulations are performed with the fixed model. The results of the exploratory simulations are discussed in the following sections. Lastly, a conclusion on which algorithm variants are used in full-scale simulations is drawn in Section 6.8.



(a) Original path (b) Newly calculated path

Figure 6.4: Example of a disregarded recovery path for AGV at tile A due to a loop in the new path. Red line: contour of the deadlock, solid black line: already traversed path, dotted black line: remaining path.

6.4 Ranking and extra resolving AGV methods

Two central parts of the recovery algorithm are the ranking method and the method determining the extra resolving AGV. Each different combination of these methods creates a unique recovery algorithm variant. There are five different ranking algorithms: Random, Longest Remaining Path, Most Used Tile, Last In Queue and, finally, Least Extra Costs Path. The Random ranking is added as a reference. Furthermore, there are two different methods used to find an extra resolving AGV: Follow Path and Next In Ranking. Simulations are performed to investigate all possible combinations of these two parts of the algorithm. The simulations are done without deadlock avoidance and with a deadlock detection interval of 20 seconds. Only one AGV per deadlock is appointed as resolving AGV. The small layout of Figure 6.1 is used. The simulations are executed for different numbers of AGVs to explore the effect of grid density on the effect of deadlock recovery.

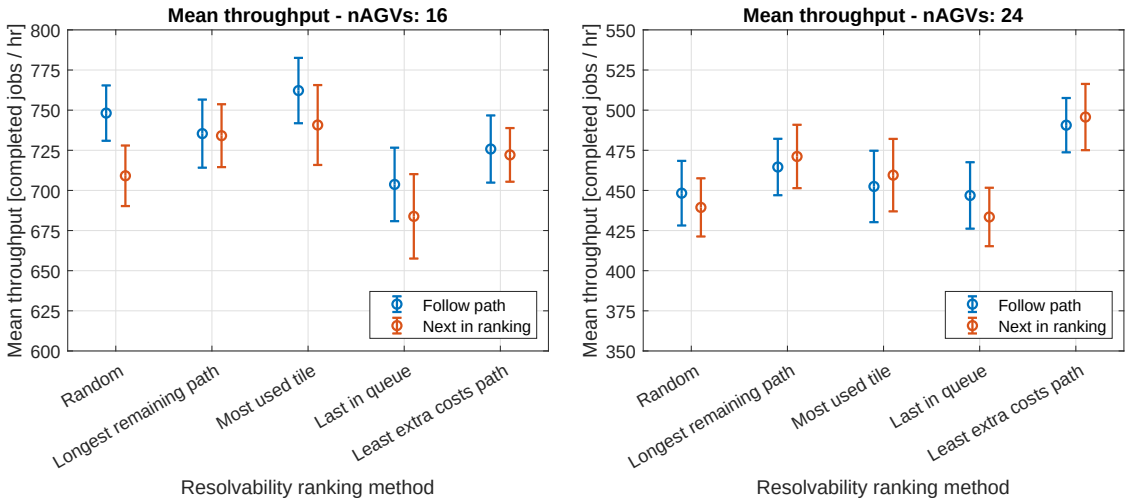
As stated before, it is expected that recovery algorithms that lower congestion the most, give the best performance. Each algorithm variant has some properties that can lower congestion, resulting in a better performance compared to other variants. However, simulations have to tell which properties are more valuable than others. The Longest Remaining Path and Least Extra Costs Path ranking methods both appoint an AGV that has to travel far, thus potentially decreasing local congestion. But, the Least Extra Costs Path ranking is expected to yield better system performance as it takes into account the costs of the alternative path. The better performance could come at the expense of higher computation costs. The Last In Queue ranking method is designed to work well around busy handling points. It aims to make way for AGVs that need to pass by a handling point, but do not need to visit it. This ranking is predicted to work well if handling points are the bottleneck and there is enough room to divert to a less dense area. Next, the Most Used Tile ranking creates a resolving AGV that hinders the most other involved AGVs. So, that ranking should give the best results when deadlocks occur in a dominant AGV flow.

There are two methods designed to find an extra resolving AGV: Follow Path and Next In Ranking. The Next In Ranking method just takes the next involved AGV of the ranking. This method should perform best if all involved AGVs are roughly equally sensible choices. The Follow Path method aims to remove the blockage of the initial resolving AGV. Thus, making it more likely that the initial resolving AGV can start driving. The Follow Path method is therefore predicted to perform better if the consequences per AGV differ more.

The system performance is discussed in this section. After that. the effect of using different resolvability ranking methods on the computation time is reviewed in Section 6.7.

Throughput

The mean throughput is the most important performance measure. Figure 6.5 shows the results for the mean throughput on the small test layout. In general Figure 6.5a and 6.5b show that there is no favourable method to find an extra resolving AGV if the initial resolving AGV could not resolve the deadlock. The confidence intervals of the Follow Path and Next In Ranking methods lie very close to each other. This indicates that there is no clear better method.



(a) 16 AGVs (28% density) (b) 24 AGVS (41% density)

Figure 6.5: Mean throughput for different resolvability ranking methods. The combinations with a specific method to find an extra resolving AGV are visualised in separate colors.

The only big difference in extra resolving AGV method can be seen in Figure 6.5a when the Random resolvability ranking is used. Figure 6.6 shows a very typical deadlock in the test layout which explains the difference. The deadlock occurs when unloaded AGVs have a path to the pick-up locations at tiles A and F, and a loaded AGV (tile D) wants to cross the paths of the unloaded AGVs.

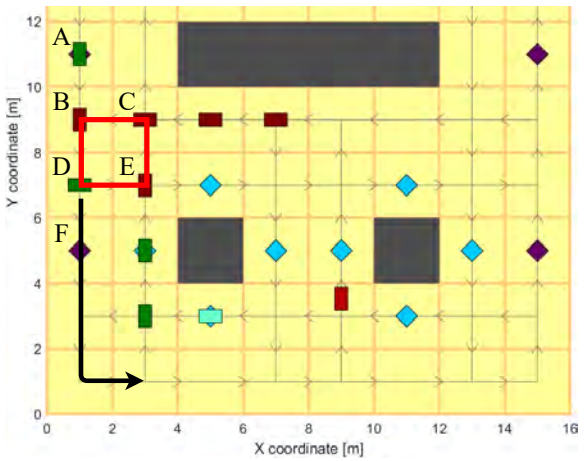


Figure 6.6: Dominant deadlock situation (marked by red box) in which loaded AGV at tile D can decrease congestion in the area of the deadlock.

The extra resolving AGV method is only called if the AGV at tile B ends up on top of the random ranking. The deadlock can be resolved if an AGV at tile C, D or E is chosen as resolving AGV and the method to find an extra AGV is not called. In the example, AGV D is given a new path if an extra resolving AGV is chosen with the Follow Path method. This AGV can easily be sent away from the congested area with a new black path as in Figure 6.6. The area of the deadlock becomes less dense when the AGV at tile D is given a new path. This has a positive effect on the throughput. With the Next In Ranking algorithm, the AGVs at tile C or E are also chosen regularly, since the extra AGV is based on the random ranking. The AGVs at tile C and E can divert, however, the AGVs will return quickly to the congested area as their destination is one of the pick-up points. The deadlock is thus resolved, but congestion is not decreased, which results in lower average throughput.

Secondly, the ranking methods are compared. The performance of the Most Used Tile and Last In Queue ranking algorithms in the simulations with 16 AGVs can also be clarified by the above explanation. The Most Used Tile ranking algorithm looks for the AGV that is blocking the most other AGVs. Given the dominant deadlock location, the AGV at tile D is often chosen as resolving AGV. Choosing the AGV at tile D decreases the congestion in the area where the deadlock happened and thus has a positive effect on the throughput. The decent performance of the Most Used Tile ranking is as expected, since there is a clear flow of AGVs around a pick-up point. The Last In Queue method performs worse than expected, since it does not outperform the Random ranking. The Last In Queue method finds involved AGVs which have the same destination and chooses the AGV that is last in the virtual queue to that destination. In the used layout, AGVs at tile C or E are often chosen as resolving AGV with this ranking method. Again, the deadlock resolves but congestion is not decreased by choosing these AGVs. This explains the poorer performance over the other ranking methods. Furthermore, the Longest Remaining Path and Least Extra Costs Path rankings also do not perform better than the Random Ranking. This probably comes from the fact that the remaining path lengths are not very distinct due to the small layout. The Longest Remaining Path ranking is thought to perform better in bigger layouts.

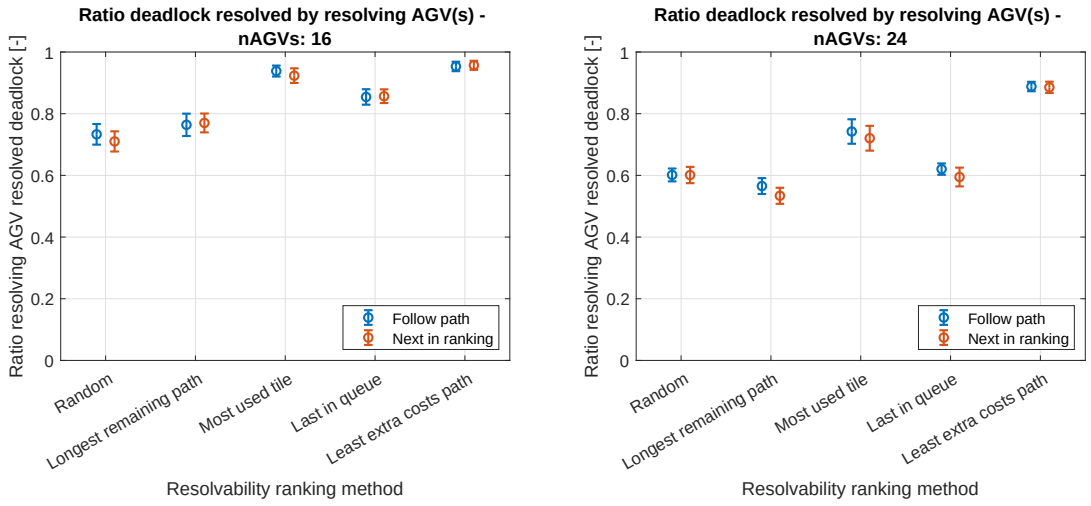
The mean throughput of the first four ranking methods for the simulations with 24 AGVs is comparable, see Figure 6.5b. This is an unexpected result, as it means that the Longest Remaining Path, Most Used Tile and Last In Queue rankings do not perform better than randomly appointing a resolving AGV. The Least Extra Costs Path ranking however outperforms the Random, Most Used Tile and Last In Queue ranking methods. The performance is better, because the ranking copes the best with congestion. The simulations are done with a very dense layout with a density of 41%. AGVs hinder each other more than in the simulations with 16 AGVs (28% density). The path planning algorithm that is used to calculate the extra costs of a recovery path, also takes into account how busy the path currently is. This means that an AGV on top of the Least Extra Costs Path ranking can relative easy move away from the congested area, which is beneficial for the throughput. Other rankings appoint a resolving AGV, but do not take into account how busy the recovery path is. As the layout is very dense, there is a big chance that the resolving AGV is waiting for other AGVs and thus does not decrease the congestion quickly.

Ratio initial resolving AGV resolving the deadlock
Next, it is counted how often the initially appointed resolving AGV(s) actually resolves the deadlock. This number is converted into a ratio of the total number of deadlocks during simulation. The ratio tells how well the first AGV in the ranking can divert with a deadlock-free path. In the case a deadlock is not resolved by giving the original resolving AGV(s) another path, an extra resolving AGV is found.

The used layout consists of many crossings, thus many tiles in the layout have two outwards segments. A resolving AGV on a tile with two outward segments can divert. The alternative path is likely to be deadlock-free if the system is not too dense, since the outward segments are probably empty. As an example, Figure 6.6 shows free outward segments for AGVs C, D and E.

Most of the tiles with only one outward segment are handling points. The rankings that regularly appoint resolving AGVs on handling points are therefore expected to score lower on the ratio.

The value of the ratio per algorithm variant as the result of the simulations is shown in Figure 6.7. The simulations that are performed with 16 AGVs show a better ratio than the simulations with 24 AGVs. It makes sense that the initial resolving AGV can divert in fewer cases as density increases. The Most Used Tile and Least Extra Costs Path ranking methods have the highest ratio for both densities. The Least Extra Costs Path ranking method has a check to see if the recovery path is different from the current path. If the recovery path is different, it is quite certain that the resolving AGV can resolve the deadlock. It is therefore as expected that this ranking method has a high ratio. The Most Used Tile method identifies the AGV that is blocking the most other involved AGVs. The AGV on top of this ranking is thus blocking the most prevailing flow in the system. The AGV at tile D in Figure 6.6 is an example of such AGV. In the time that the AGV at tile D is standing still, the AGVs following the flow (black arrow) can continue and thus empty the way for the resolving AGV. With this in mind, the AGV that blocks the flow can likely divert and thus the ratio of this method is high. The ratio of the Longest Remaining Path ranking is comparable to that of the Random ranking. The relatively small difference in remaining path length is probably the cause of this fact.



(a) 16 AGVs (28% density) (b) 24 AGVS (41% density)

Figure 6.7: Average ratio of resolving AGV(s) resolving the deadlock for different resolvability ranking methods. The combinations with a specific method to find an extra resolving AGV are visualised in separate colors.

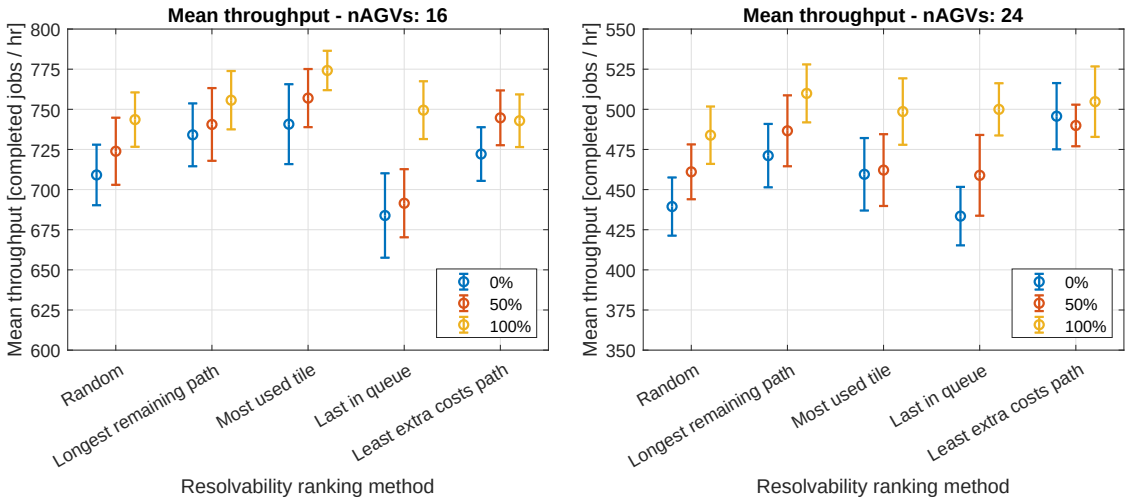
The exploratory simulations show that using different ranking methods do show significant effects on the achieved throughput. On the other hand, the method to find an extra resolving AGV does not seem to matter. The following sections also discuss the effect of the number of resolving AGV and the length of the detection interval. A full conclusion on the exploratory simulations is given in Section 6.8.

6.5 Number of resolving AGVs

Next to the resolvability ranking and extra resolving AGV methods, different algorithm variants are made, based on varying the percentage of initial resolving AGVs. Only one AGV needs to get an alternative path to resolve a deadlock. However, replanning paths for more AGVs, could distribute AGVs better over the layout and decrease congestion. On the other hand, diverting more AGVs than necessary can decrease performance, since AGVs get a longer path. Simulations

with a varying number of resolving AGVs should give insight into whether the possible benefit outweighs the possible drawback.

The result on the resolving AGV ratio is elaborated below. The simulations are only run with Next In Ranking as an extra resolving AGV method, since Section 6.4 shows that the differences with the Follow Path method are very small. The simulations are performed for 0, 50 and 100 percent of the number of involved AGVs set as resolving AGVs. At least one AGV is chosen as resolving AGV, thus the setting 0% gives one resolving AGV. Over 95% of all deadlocks during simulation have four involved AGVs. This means that the setting 50% gives two resolving AGVs. The deadlock detection interval is again set at 20 seconds.



(a) 16 AGVs (28% density)

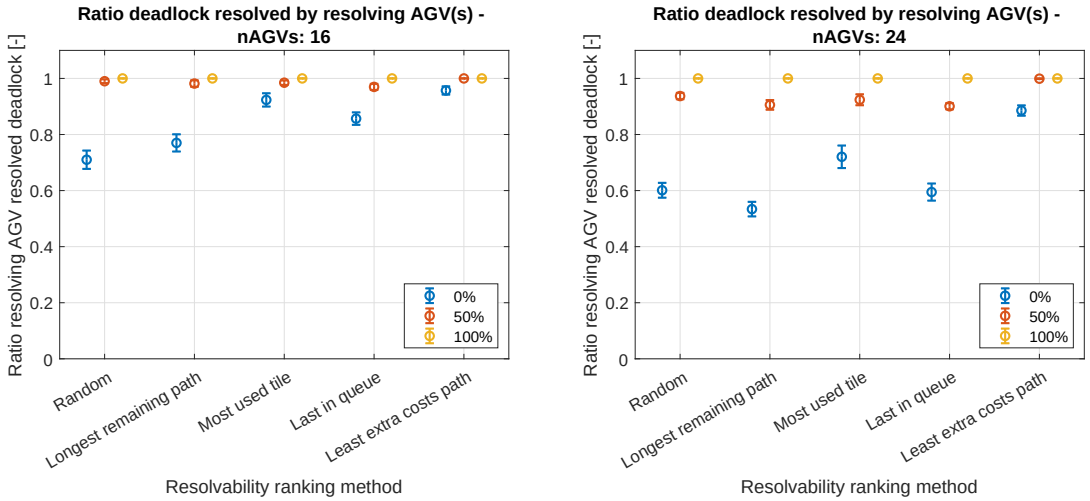
(b) 24 AGVS (41% density)

Figure 6.8: Mean throughput for different resolvability ranking methods. The combinations with certain percentages of resolving AGVs are visualised in separate colors.

The mean throughput shows a clear positive response to increasing the number of resolving AGVs, see Figure 6.8. This response comes from the forced reevaluation of the paths of involved AGVs. Even if the first resolving AGV actually resolves the deadlock, a new path is planned for the other resolving AGVs. The paths of the other resolving AGVs do not change to resolve a deadlock, since the first resolving AGV already resolved the deadlock. However, the paths of the other resolving AGVs can change because another path with lower costs is found. The costs are partly based on how busy a path is expected to be. In other words, planning a recovery path for more AGVs than necessary can decrease congestion of the area of the deadlock, thus having a positive influence on throughput.

Figure 6.5a gives some clue that the Most Used Tile ranking yields higher throughput than some of the other ranking methods in the least dense layout. Figure 6.8a shows that the Most Used Tile ranking performs better than the Random, Last In Queue and Least Extra Costs Path ranking methods if all involved AGVs are set as resolving AGV. The difference seems strange at first glance, since all involved AGVs get a new path. However, the resolving AGVs get a new path sequentially based on the ranking. The reason that throughput is not equal if all paths are replanned, is because of the order in which the paths are replanned. For the high-density simulations, the Least Extra Costs Path ranking outperforms the other ranking if only the first AGV in the ranking is assigned as resolving AGV, see Figure 6.8b. However, the advantage over other ranking methods vanishes if all involved AGVs get a new path. The distinctive advantage per rankings diminishes since AGVs hinder each other a lot with high density.

Lastly, Figure 6.9 shows the effect of increasing the percentage of resolving AGVs on the ratio with which resolving AGVs do resolve the deadlock. The result is as expected. The ratio becomes one when all AGVs are given a new path. In other words, there is almost always one involved AGV that can get a deadlock-free recovery path.



(a) 16 AGVs (28% density)

(b) 24 AGVS (41% density)

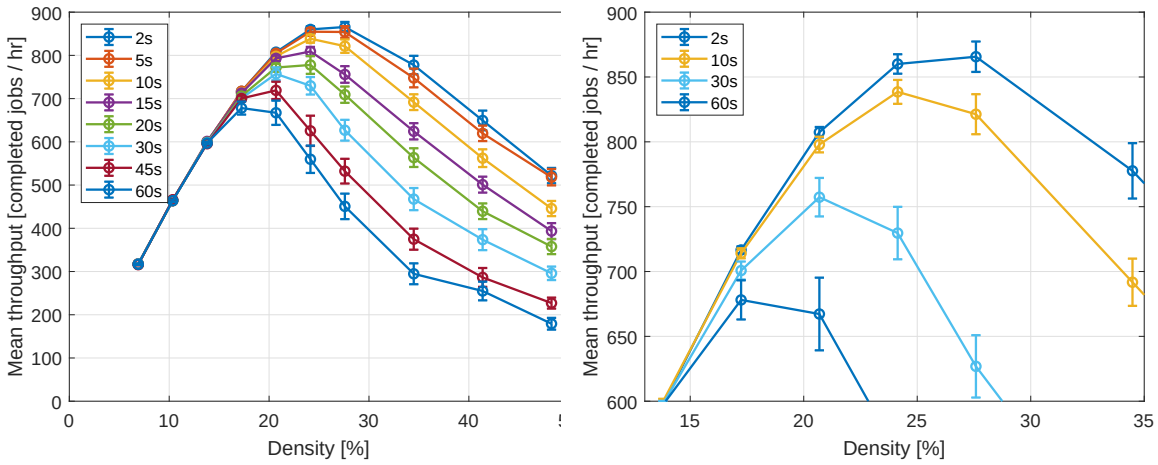
Figure 6.9: Average ratio of resolving AGV(s) resolving the deadlock for different resolvability ranking methods. The combinations with certain percentages of resolving AGVs are visualised in separate colors.

It is clear from Figure 6.8 that increasing the percentage of resolving AGVs has a positive influence on the throughput. However, this effect is due to more rerouting and not specifically due to a better recovery approach. The optimal path can change if dynamic weights are used. Thus, planning a path for more AGVs during deadlock recovery, gives more AGVs a better path. However, in practice, periodic rerouting is used. This means that approximately every 10 seconds the path of each AGV is adjusted if a better path is found. Periodic rerouting is disabled during the above simulations to isolate the effect of the deadlock recovery algorithm. The simulations are also performed including periodic rerouting. The increase in throughput, as in Figure 6.8, is not visible if periodic rerouting is enabled. This fact confirms that the increase in throughput for more resolving AGV comes from more rerouting.

6.6 Detection interval

The deadlock detection interval is the last system parameter that is varied. The interval is a value not directly related to the deadlock recovery algorithm. However, it might largely influence the performance of the system. Congestion is limited if deadlocks are detected earlier, and thus recovered earlier. The best throughput should be achieved with the smallest detection interval.

Figure 6.10 shows the average throughput for increasingly dense simulations for different deadlock detection intervals.The average throughput first increases. AGVs do not hinder each other at all, and thus no deadlocks occur for low densities. Starting from a density of 17,5%, AGVs start to hinder each other so the throughput does not increase as fast, and deadlocks start to form. A longer detection interval has only effect if there are deadlocks. So, changing the interval only has an effect on throughput starting from 17,5% density.



(a) Full result

(b) Zoomed with few detection intervals

Figure 6.10: Throughput versus the density of the layout and the detection interval.

Congestion increases with increasing density to a point that throughput decreases. As expected, the maximum achievable throughput increases if the deadlock detection interval increases. The number of connected AGVs is limited, if a deadlock is detected early after formation. Thus, the added waiting time is restricted. As expected, figure 6.10b clearly shows that the maximum throughput increases for shorter deadlock detection intervals. The figure also shows that the density at which the maximum throughput is achieved increases. As deadlocks are detected earlier, hinder is minimized and more AGVs can be placed in a layout before the caused extra hinder of an additional AGV decreases throughput.

Section 6.4, 6.5 and 6.6 elaborate on the system performance for different deadlock recovery algorithm variants. Next, the computation time of different ranking methods is discussed. After that, a conclusion on the algorithm variants is given in Section 6.8 based on the exploratory simulations.

6.7 Computation time

To be able to properly compare the ranking algorithms, the computation time needs to be taken into account as well. It is expected that the Random ranking is the quickest, because creating a random ranking is a simple operation. The Least Extra Costs Path ranking needs to calculate the alternative path of each involved AGV to make up the ranking. Consequently, this ranking is deemed to be the most computationally expensive.

The computation time of the ranking algorithm and the total recovery algorithm are measured. 50% of the involved AGVs is appointed as resolving AGV for the simulations. The results on computation time are stated in Table 6.1.

Table 6.1: Mean computation time per ranking method based on approximately 3000 samples.

	Ranking method	**Mean total recovery computation time** [s]	**Mean ranking computation time** [s]
1	Random	3,87e-2	7,17e-6 (0,019%)
2	Longest remaining path	4,40e-2	4,09e-5 (0.093%)
3	Most used tile	4,04e-2	8,12e-5 (0.20%)
4	Last in queue	4,26e-2	6,45e-5 (0.15%)
5	Least extra costs path	7,49e-2	4,39e-2 (58,61%)

The Random ranking is the quickest as expected. Rankings 2, 3 and 4 only use prior available information to determine the ranking. This has the result that the computation time to create the ranking is very low compared to the total recovery computation time. The Least Extra Costs Path ranking method calculates a new path for every involved AGV. The average computation time for this ranking method is more than half of the total recovery computation time, since calculating new paths is relatively expensive.

6.8 Conclusion exploratory results

Exploratory simulations are performed to find improvements of the algorithm. The small scale simulations help tremendously in finding and fixing minor bugs. Two more substantial path planning flaws are described in Section 6.3. The relative short simulation time of the exploratory simulations makes the development of the algorithm, as presented in Section 5.2 a lot easier. For instance, the use of prohibited segments, instead of AGVs, is an improvement implemented as a result of exploratory simulations. The exploratory simulations also give a feeling for the effectiveness of deadlock recovery variants. A choice is made on which recovery variants are tested on full-scale layouts, based on the exploratory results. These full-scale simulations are discussed in the next Chapter 7. The results in Sections 6.4, 6.5 and 6.7 are the basis for the following conclusions on algorithm variants.

The simulations give the insight that the effectiveness of the deadlock recovery algorithm is linked to the extent to which congestion is decreased as a result of the recovery actions. As resolvability ranking, only the Random, Longest Remaining Path and Most Used Tile methods are tested on full-scale layouts. The Random ranking acts as a baseline. The Longest Remaining Path and Most Used Tile ranking methods have good potential in decreasing the congestion in the area of the deadlock, and are therefore prone to result in the highest throughput. The Last In Queue ranking method is not chosen, since that ranking method shows the worst throughput performance. This ranking method selects resolving AGVs that do resolve the deadlock, but do not effectively reduce the congestion around a handling point. The Least Extra Costs Path ranking method is also not chosen. The ranking shows comparable results to the chosen ranking methods, but at the expense of longer computations. Especially the Longest Remaining Path ranking is thought to perform equally to the Least Extra Costs Path ranking in larger layouts.

As a method to find an extra resolving AGV, the Next In Ranking method is chosen for the full-scale simulations. The exploratory simulations show that there is no significant difference in throughput between the two extra resolving AGV methods. Therefore, the simplest method is chosen. Furthermore, it is clear from Figure 6.8 that increasing the percentage of resolving AGVs has a positive influence on the throughput. However, this effect is due to more rerouting and not specifically due to a better recovery approach. In practice, periodic rerouting is used, so there is no need to force a reroute action by the deadlock recovery algorithm. Hence, for full-scale simulations, only one resolving AGV is appointed. Moreover, the deadlock detection interval has a huge influence on the performance of the AGV system when deadlock recovery is the deadlock handling method. The deadlock detection interval is kept at 20 seconds. The full-scale simulations of Chapter 7 are about the relative performance of the deadlock recovery variants, so it is not a problem that better overall performance can be obtained by lowering the deadlock detection interval. After the most promising deadlock recovery algorithm is found in the following Section 7.2, the performance of deadlock recovery is compared to deadlock avoidance in Section 7.3. In these simulations, the effect of the deadlock detection interval is taken into account.

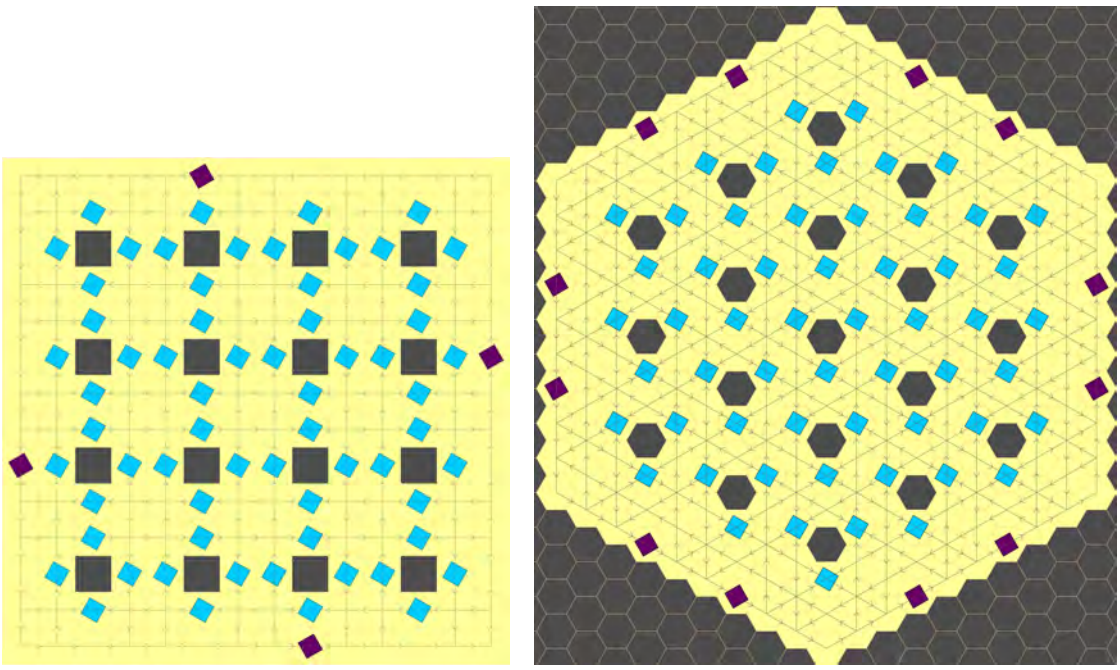
7 Full-scale simulations

The goal of this project is to develop a deadlock recovery algorithm that yields as high as possible throughput. The exploratory simulations of Chapter 6 give insight into how the deadlock recovery algorithm interacts with the other AGV controls. It is clear that congestion control is an important factor in the performance of the recovery algorithm. This chapter describes the results of full-scale simulations in order to conclude to most suitable deadlock recovery algorithm variant. The full-scale simulation layouts are larger than the layout used for the exploratory tests, to resemble real-life situations better. Only a selection of most promising algorithm variants is tested, based on the results of the exploratory simulations, see Section 6.8. The variants of the deadlock recovery algorithm only use different resolvability ranking methods.

The use of different ranking algorithms is tested with simulation described in Section 7.1. The results are discussed in Section 7.2. The deadlock recovery variant that yields the best throughput is compared to deadlock avoidance in Section 7.3. A conclusion on computation time of the recovery algorithm can be drawn based on these results. Lastly, Section 7.4 discusses the reachability guarantees of Section 5.4 with the use of the deadlock recovery algorithm during simulations.

7.1 Simulation set-up

The full-scale simulations are done on two different layouts: GridSorter4 and BeeHive4. These layouts have many intersection points and are therefore prone to have deadlocks, see Figure 7.1. Both layouts have pick-up stations around the edges and chutes spread across the layout to drop off luggage or packages. Each chute has multiple drop-off locations around it. GridSorter4 has square tiles and BeeHive4 has hexagonal tiles. The use of hexagonal tiles can give different types of deadlocks and paths compared to a layout with square tiles.

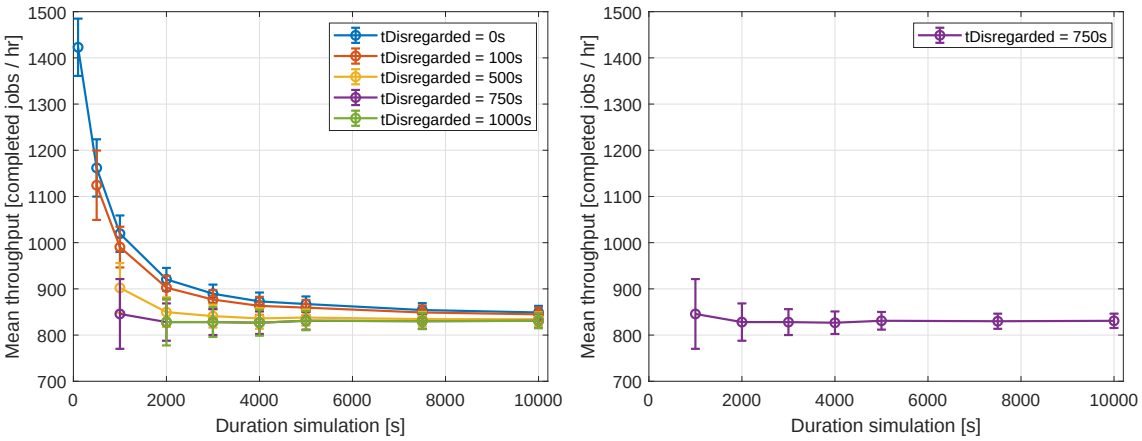


(a) GridSorter4 (b) BeeHive4

Figure 7.1: Layouts used for full-scale simulations

There are some additional parameters to set for each simulation. The density is set to 25%. This means that 25% of the driveable tiles of the layout are occupied by an AGV. This number is higher than 15-20% which is typically used, to make sure enough deadlocks occur, to be able to compare the deadlock recovery algorithm variants. Furthermore, the deadlock detection interval is set to 20 seconds. This interval is a bit shorter than the default value of 30 seconds to compensate for the higher density. The exact influence of the detection interval is discussed in Section 7.3.

The length of the simulation is varied to investigate the effect on the throughput and the presence of a start-up effect. The effect is tested on the GridSorter4 layout with the random resolvability ranking setting. The throughput is assumed to be normally distributed and thus visualised using 95%-confidence intervals, based on the result in Section 6.2. Figure 7.2a shows the effect on throughput for different lengths of simulations.



(a) Different lengths of disregarded periods

(b) Disregarded start-up period of 750s

Figure 7.2: Mean throughput for different lengths of simulations while disregarding a start-up period. Data point are connected by lines for visibility only.

Different colours represent throughput values for which different start-up periods are disregarded. The line for *tDisregarded = 0* displays the result without disregarding any start-up period, and shows that there is a clear start-up behaviour. This behaviour happens since all AGVs have a random initialisation location in the layout. As a result, AGVs do not hinder each other that much at the beginning, which increases the throughput. As the simulation proceeds, queues build up around the pick-up locations, creating steady-state behaviour. The increased throughput at the beginning of the simulation increases the average throughput. Disregarding the start-up period makes sure the steady-state throughput is obtained quicker. The obtained throughput does not change anymore between disregarding the results of the first 750 or 1000 seconds, see Figure 7.2a. Hence, the first 750 seconds is considered as the start-up period. Figure 7.2b shows the throughput for different simulation durations, for which the first 750 seconds is disregarded. A duration of 4000s is chosen for the simulations. The duration is a balance between decreasing the 95%-confidence intervals and simulation time. All metrics in the following sections are thus based on the results from 750 till 4000s of each simulation.

Secondly, the number of simulations per setting is set to 30, based on Figure 7.3. This number is a balance between precision and simulation time.

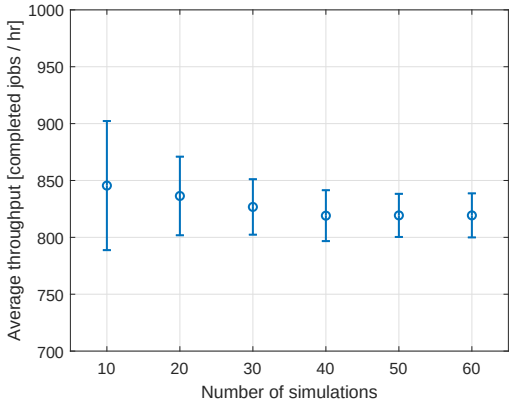


Figure 7.3: Mean throughput for grouping varying number of simulations with disregarding the first 750 seconds.

7.2 Results

This section discusses the obtained results for two different layouts. The only difference in deadlock recovery algorithm is made in the resolvability ranking method. Contrary to the conclusion of the exploratory simulations, four, instead of three, ranking algorithms are tested on the full-scale layouts. The exploratory results of Chapter 6 show that it is important that the resolving AGV quickly travels away from the deadlock area. Resolving AGVs in the BeeHive4 layout are hindered more during their travel away from the deadlock area than in the layout for the exploratory simulations. Therefore, the Least Extra Costs Path ranking is also included in the full-scale simulations. This ranking method also considers congestion of the alternative path, which could result in better throughput compared to the other ranking methods. Furthermore, the Longest Remaining Path ranking should perform better than the Random ranking. This is expected, due to bigger differences in remaining path length compared to the exploratory simulations. The Most Used Tile ranking is designed to restore a dominant flow that is blocked by a deadlock. This ranking should give the best results if there are many deadlocks around handling points.

GridSorter4

Figure 7.4 shows the mean throughput in the GridSorter4 layout. The results for the Random

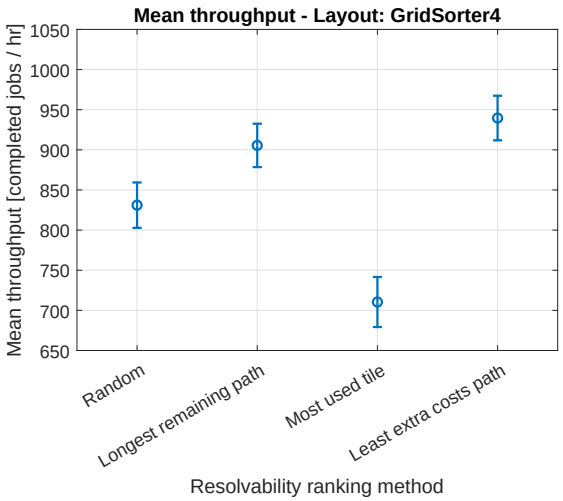


Figure 7.4: Mean throughput during a simulation on the GridSorter4 layout.

ranking are a baseline for the other ranking methods. The Most Used Tile ranking is the most promising ranking method based on the exploratory simulations. However, Figure 7.4 clearly shows that the Most Used Tile ranking underperforms the Random ranking for the GridSorter4 layout, despite many deadlocks happen around pick-up points. This can be explained by Figure 7.5, which shows a very common deadlock.

The deadlock occurs around a pick-up location where streams of unloaded and loaded AGVs cross. The Most Used Tile ranking looks for the AGV that is on a tile over which the most other involved AGVs have to travel. AGV at tile D ends almost always on top of the ranking. However, tile D is the pick-up location, so that AGV cannot divert. The second AGV in the ranking is often located at tile A. The AGV at tile A can divert, but the AGV is unloaded and thus quickly returns to the congested area around the pick-up point. The Most Used Tile ranking cannot predict if an AGV can and will swiftly move away from the congested area, which is the most important property to improve the throughput. The Most Used Tile ranking shows exactly the behaviour for which the Last In Queue ranking method is dropped after the simulations on the small layout.

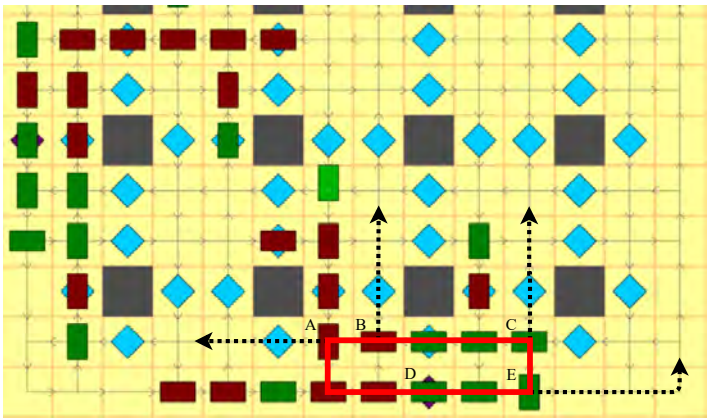


Figure 7.5: Common deadlock (in red) around a pick-up point in the GridSorter4 layout. The dotted arrow indicate diverting options to resolve the deadlock.

The Longest Remaining Path and Least Extra Costs Path ranking algorithms perform significantly better than the Random ranking. This is not the case in the small exploratory tests. The difference can be explained by the size difference of the layouts. The GridSorter4 layout has 3,1 times as many tiles as the small layout. The differences in the remaining path length of involved AGVs are bigger and thus is the ranking more distinct. Secondly, the distances within the layout are bigger, so sending away the AGV with the long remaining path has a bigger effect.

The Least Extra Costs Path ranking method seems to perform better than the Longest Remaining Path method. However, this conclusion cannot be drawn, as the confidence intervals overlap too much. The two ranking methods perform alike, since resolving AGVs are not hindered a lot by other AGVs during the first steps of their recovery path. The tiles along the dotted arrows in Figure 7.5 are almost always empty. This means that a resolving AGV can easily move away from the deadlock area. The Least Extra Costs Path ranking takes into account the congestion along the recovery path, but this does not have a very big advantage compared to the Longest Remaining Path ranking.

Secondly, Figure 7.6 shows the resolving AGV ratio for the GridSorter4 layout. The Least Extra Costs Path ranking method has the maximum ratio, as expected. Paths are planned during the creation of the ranking. It is taken into account if an AGV can get a deadlock-free path. The Longest Remaining Path and Most Used Tile ranking score worst on this performance measure. Deadlocks often occur around a pick-up point, see Figure 7.5. The resolving AGV appointed by

the two rankings is often located at the pick-up point (tile D) and cannot divert. Thus, another AGV has to be found to resolve the deadlock.

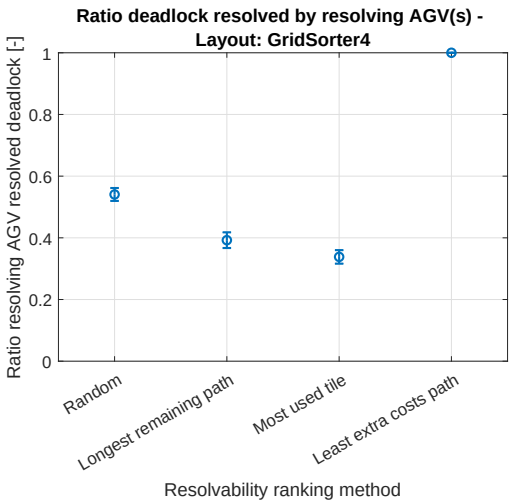


Figure 7.6: Mean ratio of the resolving AGV actually resolving the deadlock for the GridSorter4 layout based on a set of 30 simulations per ranking method.

BeeHive4

Now the performance of the deadlock recovery algorithm variants is discussed for the simulations on the hexagonal BeeHive4 layout. The results shown below are based on sets of 45 instead of 30 simulations. The number of simulations per set is increased since the BeeHive4 layout is 40% bigger than GridSorter4. This has the result that the average throughput fluctuates more between simulations.

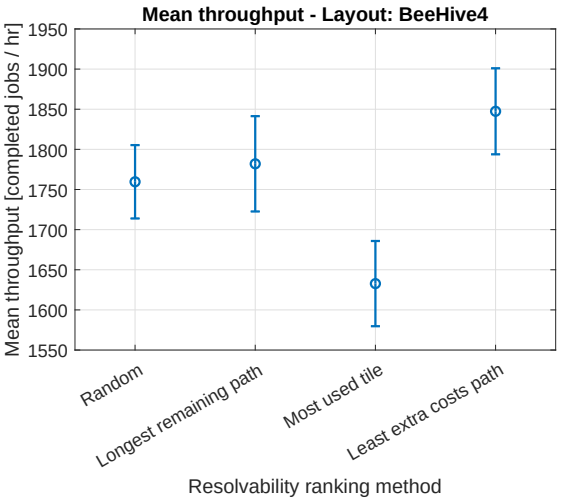


Figure 7.7: Mean throughput during a simulation with the BeeHive4 layout.

The mean throughput of the Most Used Tile ranking is lower than the Random ranking for the same reason as with the GridSorter4 layout. The resolving AGV following from that ranking does not drive away from the congested deadlock area. The mean throughput value of the Longest Remaining Path ranking is a bit surprising at first sight, as it is not significantly better than the Random ranking. The result can be explained by two observations. The first is that the BeeHive4

has twelve pick-ups points. As a result, pick-up points are less congested, so more deadlocks occur in the middle of the layout. The path length of involved AGVs of deadlocks in the middle of the layout is less distinct. Thus, a ranking by the Longest Remaining Path method approaches a randomly generated ranking for deadlocks in the center of the layout. The second observation is that resolving AGVs in the BeeHive layout cannot drive away from the congested area as easy as in the GridSorter4 layout. Figure 7.5 shows that the recovery AGV can often immediately drive a few tiles away from the deadlock in the GridSorter4 layout. On the other hand, an AGV with a recovery path in the BeeHive4 layout is generally more obstructed, see Figure 7.8.

Since resolving AGVs are hindered by other AGVs, the Least Extra Costs Path ranking performs the best. The extra costs of a recovery path consist of extra tiles to drive, turn penalties and tile weights. The latter is a measure of how congested the path is. AGVs that can divert through a lower congested area are thus preferred by the ranking method. Furthermore, the ranking is based on the relative extra costs compared to the current path. AGVs with a long current path, consequently have lower relative extra costs and are thus preferred by the ranking method. In short, the ranking provides a resolving AGV that wants to get away from the congested area (long remaining path) and can easily do so (low congestion on the path). This has the result that congestion is lowered in the area of the deadlock and because of that, the throughput increases.

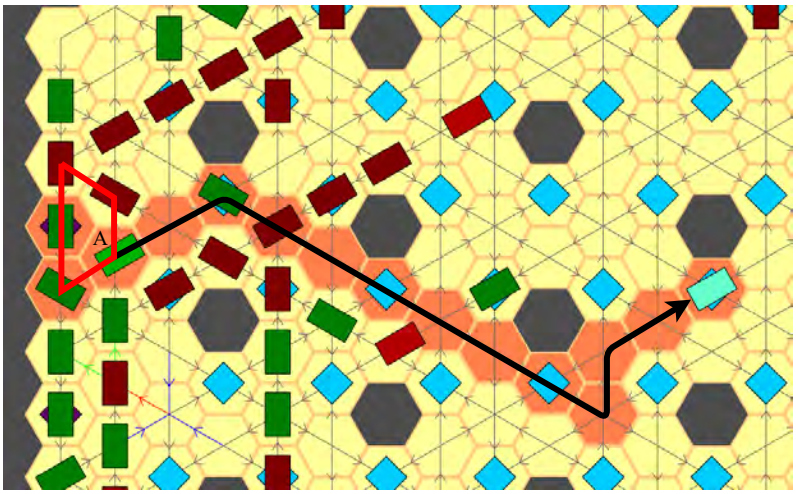
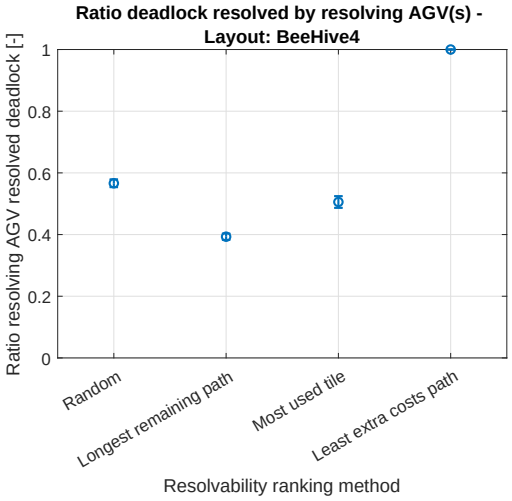


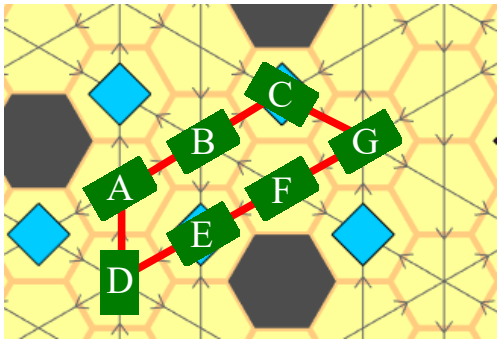
Figure 7.8: Congested recovery path of resolving AGV at tile A in the BeeHive4 layout.

Next, the resolving AGV ratios of the BeeHive4 simulations are shown in Figure 7.9a. The Random, Longest Remaining Path and Least Extra Costs Path ranking methods show very similar ratios to the ratios of the GridSorter4 simulations.

The Most Used Tile ranking has a resolving AGV ratio of about 0.5, where the same ranking only scored about 0.35 in the GridSorter simulations. This improvement can be explained by the BeeHive layout, see Figure 7.9b. The Most Used Tile ranking finds the AGV that is on a tile, over which the most other involved AGVs have to travel. In the example, AGV C and E are on a drop-off location, that only has one outward segment. Each AGV that needs to travel to the tiles of AGV C or E, always also needs to go over tiles G and D respectively. Tiles of AGV G and D will therefore always be visited at least one time more than the tiles of AGV C and E. Thus, AGV C and E will never end on top of the Most Used tile ranking. The probability that a resolving AGV cannot divert with the Most Used Tile ranking decreases, and thus improves the ratio shown in Figure 7.9a.



(a) Ratios



(b) Example deadlock in which AGVs without diverting option end up lower in the Most Used Tile ranking.

Figure 7.9: Mean ratio of the resolving AGV actually resolving the deadlock for the BeeHive4 layout based on a set of 45 simulations per ranking method.

Conclusion on throughput

The Least Extra Costs path ranking performs the best in the BeeHive4 layout and shares the best throughput values with the Longest Remaining Path ranking in the GridSorter4 layout. The ranking method can be crowned as the best ranking, based on system performance in the tested layouts. Furthermore, it is important to decrease local congestion around a deadlock as quickly as possible to obtain the highest throughput. The Least Extra Costs Path ranking is believed to deal the best with congestion in general, since it is the only ranking that takes into account the alternative paths. Because of that reason, the ranking is expected to also give the best system performance in other layouts than GridSorter4 and BeeHive4.

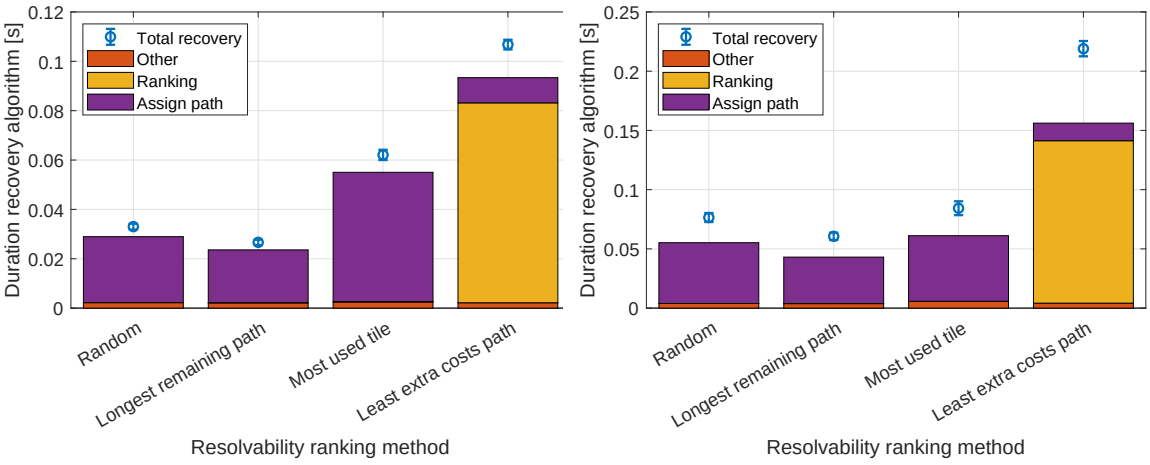
Computation time

A suitable deadlock recovery algorithm should not only yield the highest throughput. The computation time of a recovery action is also an important consideration. The Least Extra Costs Path ranking method calculates a path for each involved AGV. It is therefore expected that the deadlock recovery algorithm using this ranking algorithm has the longest computation time. The Most Used Tile ranking has the worst resolving AGV ratio in the GridSorter4 layout. This means that often an extra resolving AGV has to be appointed and more paths need to be planned. For this reason, using the Most Used Tile ranking is thought to be more expensive than using the Longest Remaining Path. For the same reason, using the Longest Remaining Path ranking is predicted to take longer than the Most Used Tile ranking in the BeeHive4 layout.

Figure 7.10 shows the average duration of one recovery call for the different ranking methods in both layouts. The time it takes to recover a deadlock is split into three parts. The 95%-confidence intervals of the individual parts are omitted for clarity as the size of the intervals is below 5% of the mean. The ranking part shows the time it takes to create the resolvability ranking. The Assign path part is the part of the algorithm for which the resolvability ranking is the input. This part of the algorithm stops when a deadlock is resolved. Path planning for extra resolving AGVs is also included in the Assign path part. Lastly, there is a small remaining portion of the computation time that is devoted to logging and general actions.

Figure 7.10 shows that the average duration of one call of the recovery algorithm (data point) is not equal to the average computation time needed to recover one deadlock (bar). This is the case, because generally multiple deadlocks are recovered within one deadlock recovery call. On average

1.15 deadlocks are recovered per recovery call in the GridSorter4 layout and 1,41 deadlocks per recovery call in the BeeHive4 layout. Multiplying these factors with the average time needed to recover one deadlock yields the average duration of one deadlock recovery call.



(a) GridSorter4 (b) BeeHive4

Figure 7.10: The average duration of one recovery call (data points) and the duration of the recovery of one deadlock (stacked bars). Note the difference in y-axis range.

The time it takes to create the resolvability ranking is negligibly small for the first three ranking algorithms. However, creating the Least Extra Costs Path ranking is by far the longest action in a recovery call when this ranking is used. It is as expected that creating the Least Extra Costs Path ranking takes the longest computation time. Furthermore, the computation time of the Assign path part of the recovery algorithm shows large differences between ranking methods. The expectation that the resolving AGV ratio explains the difference in duration of Assign path is only partly true.

Two related other values are found, that explain the duration of the Assign path part and thus the differences. The Assign path part stops when a deadlock-free path is found for an involved AGV. Thus, the number of recovery paths planned and the time it takes to plan one path determines the duration of the Assign path part. Table 7.1 shows both values. When using the Least Extra Costs Path ranking, the first AGV in the resolvability ranking can almost always divert. That is why only one path needs to be planned in the Assign path part of the algorithm. If the initial resolving AGV could not get a deadlock-free path, an extra resolving AGV is picked by looking to the next AGV in the resolvability ranking. In general, the consecutive AGVs in the resolvability ranking for the Longest Remaining Path and Most Used Tile ranking cannot divert as often as a random extra resolving AGV. That is why more paths need to be planned for these rankings.

The average duration of one path plan action differs between used ranking methods. This is due to the used path planning algorithm: A* shortest path search. This path planning algorithm uses a heuristic to estimate the travel costs between two tiles. The better the heuristic estimates the real costs, the faster the algorithm is. The heuristic uses the Euclidean distance between the tiles as an estimate for the costs. The involved AGVs on top of the Longest Remaining Path and Least Extra Costs Path rankings have relative long paths. This means that the recovery path is likely to still be in the direction of the destination. In this case, the Euclidean distance is a good estimate for the cheapest path. However, the AGV on top of the Most Used Tile ranking is often an AGV that has to travel away from its destination to divert deadlock-free. In these cases, the Euclidean distance is a worse estimator of the cheapest path compared to a recovery path in the direction

of the destination. Thus, it takes longer to find the cheapest deadlock-free path.

Table 7.1: The average number of times a path is planned for an involved AGV before the deadlock is resolved during Assign path and the duration of one path plan action. Ranking: 1. Random, 2. Longest remaining path, 3. Most used tile, 4. Least extra costs path.

	GridSorter4				BeeHive4			
Ranking	1	2	3	4	1	2	3	4
Number of paths planned to resolve one deadlock [-]	1.95	2.27	3.21	1.00	1.73	1.85	1.99	1.01
Average duration to plan one path [s]	0.014	0.010	0.016	0.010	0.030	0.021	0.029	0.015

Figure 7.10 shows that a recovery call with the Least Extra Costs Path ranking takes approximately two to four times as long as the other a recovery action with the other ranking methods. However, no conclusion can be drawn yet whether this computation time is too long. To put the computation time of the recovery algorithm with the Least Extra Costs Path ranking in perspective, the computation time of detection and recovery versus deadlock avoidance is compared in Section 7.3.

7.3 Deadlock avoidance versus detection and recovery

There are multiple ways to deal with deadlocks as described in Section 3.2. Deadlock avoidance is previously used as the main deadlock handling method. The deadlock avoidance algorithm can be adjusted to avoid different types of deadlock, from monocycle cycle deadlocks up to inevitable thee-cycle deadlocks. The most strict deadlock avoidance variant is always used as there was no deadlock recovery algorithm available. New combinations of deadlock handling techniques can be formed with the addition of deadlock recovery. The use of a different deadlock handling technique results in different throughput effects.

To illustrate the differences, assume a set of AGVs encounter each other in a way that a deadlock is created. Deadlock avoidance prevents the deadlock by letting AGVs wait for a little moment before a deadlock arises. The additional waiting avoids the deadlock, but can introduce additional hinder to consecutive AGV. Additional hinder can result in throughput loss compared to a system without deadlock avoidance. Such hinder is prevented if only deadlock detection and recovery is used. In theory, the throughput can increase if only deadlock detection and recovery is used. However, letting deadlocks happen, creates a period in which involved AGVs are standing still. In the period between deadlock creation and recovery, none of the involved AGVs is moving. This waiting has a negative influence on the throughput. The use of deadlock recovery can thus improve and deteriorate throughput compared to using deadlock avoidance. Therefore, the performance of three different combinations of deadlock handling techniques is compared in this section.

Three deadlock handling variants are compared. Deadlock detection and recovery is combined with: no deadlock avoidance, monocycle avoidance and three-cycle avoidance (referred to as multicycle avoidance). The goal is to analyse the effect of the deadlock handling technique on throughput and computation time for increasing density and varying the deadlock detection interval. Deadlock recovery performs the best with the shortest deadlock detection interval. Vis [14] expected nonetheless that only using deadlock detection and recovery as deadlock handling technique gives the worst system performance. The simulations should also conclude if the computationally expensive Least Extra Costs Path ranking can be used. Deadlock avoidance is a relative cheap action, but is executed before every tile reservation. Recovering a deadlock is very expensive compared to one deadlock avoidance action, however it happens not that often. The contrast make it hard to predict which deadlock handling technique is overall computationally less expensive.

Simulations are completed for a range of densities. Besides grid density, the deadlock detection interval is also varied between 5, 15 and 60 seconds. It is expected that throughput is lower if the deadlock detection interval is bigger because deadlocks are not resolved as quickly. The layout used for the comparison is GridSorter5, see Figure 7.11. The GridSorter5 layout is identical to GridSorter4 apart from four additional pick-up points. Pick-up points are added in GridSorter5 to make sure the bottleneck for throughput is the interaction between AGVs, not the pick-up points.

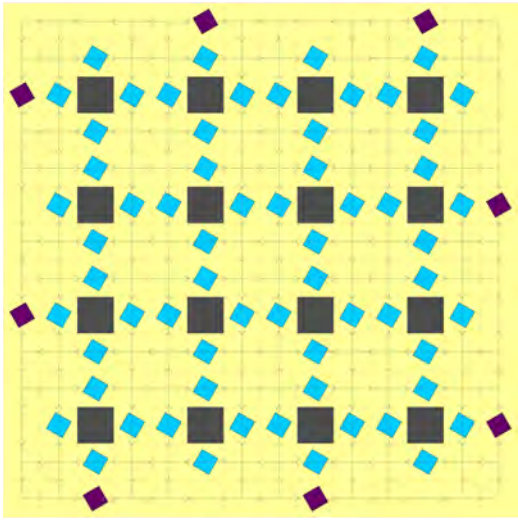
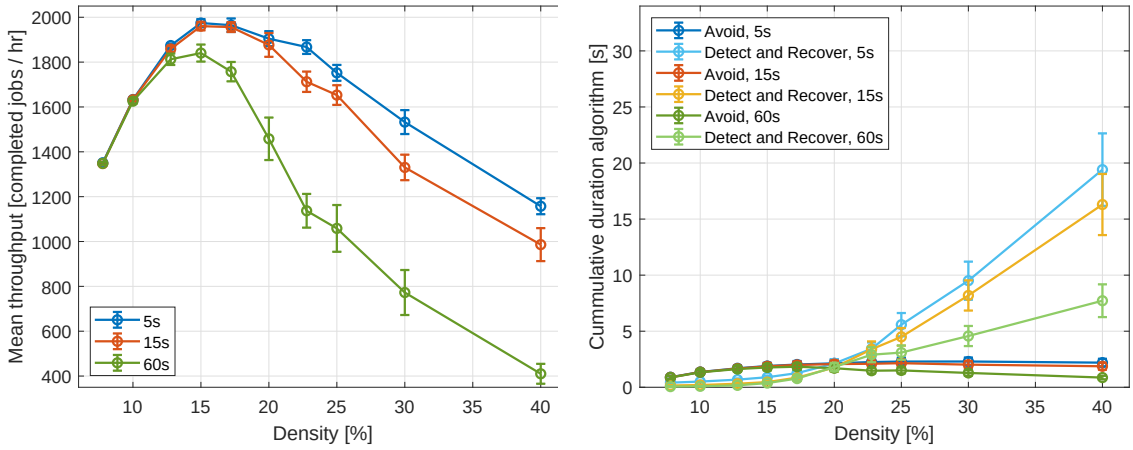


Figure 7.11: GridSorter5 layout used for comparison of deadlock handling methods.

Figures 7.12, 7.14 and 7.15 show the two main performance indicators of the three deadlock handling combinations: average throughput and cumulative computation time.

Only deadlock detection and recovery

The results when only deadlock detection and recovery is used, are shown in Figure 7.12.



(a) Mean throughput

(b) Computation time

Figure 7.12: Performance of the system with deadlock recovery as the only deadlock handling method for three different deadlock detection intervals.

First, Figure 7.12a shows that throughput first increases for increasing density, as AGVs do almost

not hinder each other. Starting from approximately 12,5% density the first hinder and deadlocks occur. Figure 7.13 shows the number of deadlocks during a simulation to confirm this. The throughput changes for different deadlock detection intervals, starting as the first deadlocks occur with a density of 12,5% density. The quicker deadlocks are detected, the fewer AGVs are standing still and the higher the throughput. Not only the involved AGVs are standing still longer if deadlocks are detected relatively late (detection interval of 60 seconds), but there will also be more connected AGVs. This increases congestion in the area of the deadlock, which has a negative influence on the throughput, even after the deadlock is resolved.

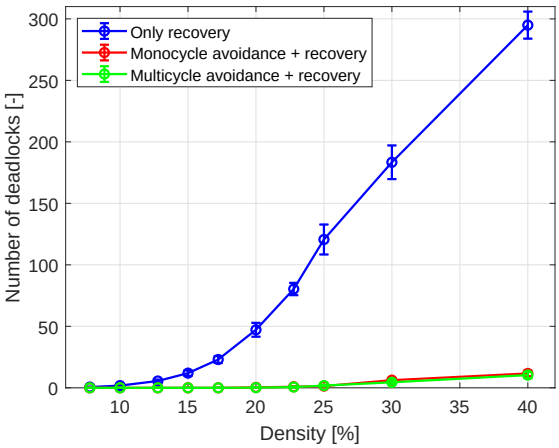
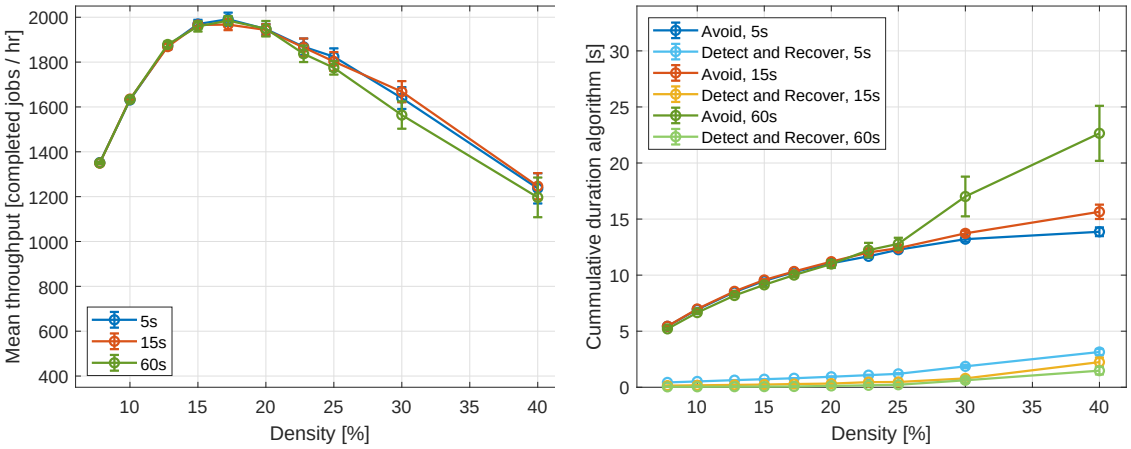


Figure 7.13: The average number of deadlocks during a simulation for the three different deadlock handling combinations with a deadlock detection interval of five seconds.

Secondly, Figure 7.12b shows the cumulative computation time during a simulation of, on one hand, deadlock avoidance, and on the other hand deadlock detection plus recovery. The computation time of deadlock avoidance is not zero, despite deadlock avoidance is disabled. The computation time is not zero, since the avoidance algorithm is still called. But, it just gives the prediction that no deadlock will occur. One avoidance step takes as little as about 50 microseconds, but happens around 40.000 times per simulation. This creates a cumulative computation time of a couple of seconds. The total computation time of deadlock detection and recovery relates mostly to the number of deadlocks that occur and thus the number of recovery actions. The cumulative computation time of deadlock detection is only about 1.5 seconds, for a deadlock detection interval of five seconds and a density of 40%. The number of deadlocks increases as the detection interval decreases. Deadlocks are more quickly resolved, thus the involved AGVs can earlier introduce new deadlocks.

Deadlock detection and recovery plus monocycle deadlock avoidance
Figure 7.14 shows the performance indicators of simulations in which monocycle deadlock avoidance is combined with deadlock detection and recovery. The number of deadlocks decreases a lot with enabled deadlock avoidance, see Figure 7.13. This explains why the influence of the deadlock detection interval on throughput in Figure 7.14a is very small.

The low average number of deadlocks also explains why the computation time of deadlock detection and recovery in Figure 7.14b is low. The cumulative computation time of deadlock avoidance steadily increases for increasing density, since more AGVs mean more deadlock avoidance actions. The computation time of deadlock avoidance for 30% and 40% density stands out. The computation time of deadlock avoidance, in combination with a 60-second detection interval, jumps up. For these densities, the first deadlocks start to emerge. These deadlocks are multicycle deadlocks. Deadlock avoidance is only called if there is an empty tile in front of the considered AGV.

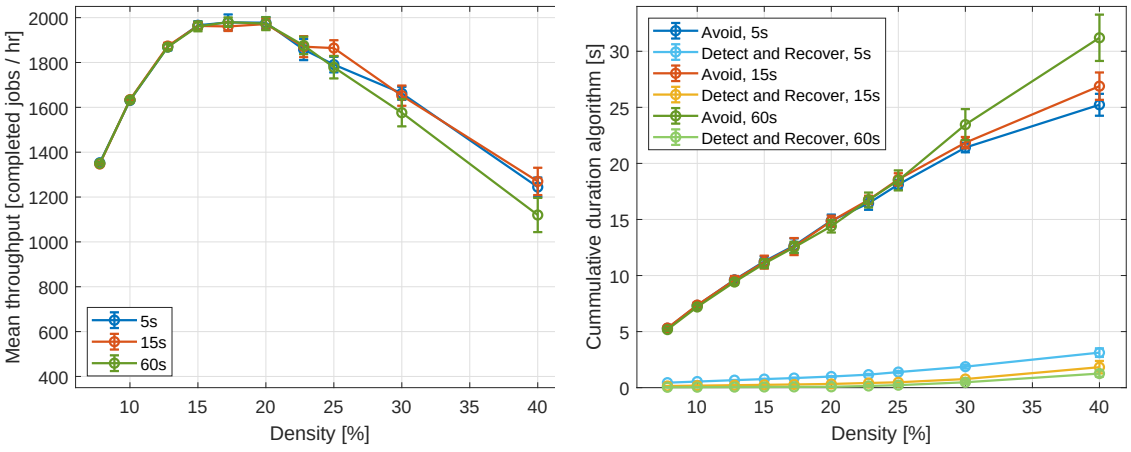


(a) Mean throughput (b) Computation time

Figure 7.14: Performance of the system with deadlock recovery in combination with monocycle deadlock avoidance as deadlock handling methods for three different deadlock detection intervals.

Multicycle deadlocks include an empty tile, and thus deadlock avoidance is called for some AGVs that are involved in a deadlock. The avoidance algorithm is only terminated when all involved AGVs are checked. It takes quite long until a deadlock is resolved with a detection interval of 60 seconds. Thus, deadlock avoidance is called often for involved AGVs that are stuck in a deadlock. This significantly increases the duration of a deadlock avoidance call.

Deadlock detection and recovery plus multicycle deadlock avoidance
The strictest deadlock avoidance algorithm is multicycle avoidance. The use of multicycle avoidance in combination with deadlock detection and recovery shows comparable throughput results as monocycle avoidance, see 7.15a. This can be explained by the number of deadlocks during simulation. The cumulative computation time of the deadlock handling methods also shows the same trend in Figure 7.15b. However, the avoidance algorithm is more complex, so the cumulative computation time for deadlock avoidance shows a steeper increase.

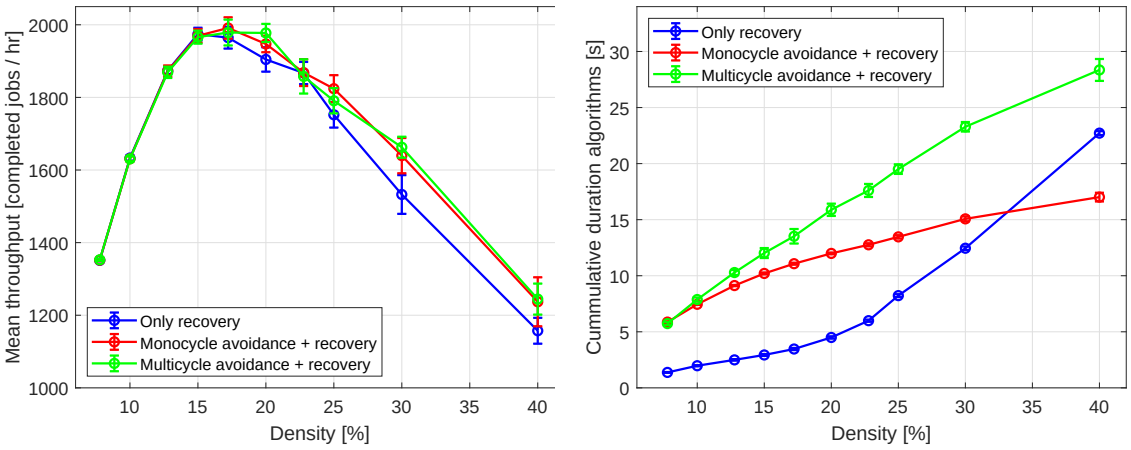


(a) Mean throughput (b) Computation time

Figure 7.15: Performance of the system with deadlock recovery in combination with multicycle deadlock avoidance as deadlock handling methods for three different deadlock detection intervals.

Combined results

The throughput and cumulative computation time of the three deadlock handling methods are combined in Figure 7.16 to easily compare the results. The figure shows the results for a deadlock detection interval of five seconds, as this setting gave the best results when deadlock avoidance is turned off.



(a) Mean throughput

(b) Mean cumulative computation time of all deadlock handling algorithms (avoidance + detection + recovery)

Figure 7.16: Comparison of the performance of the system between the three deadlock handling variants for a five second deadlock detection interval.

Surprisingly, the throughput, when only deadlock recovery is used, comes very close to the throughput when using deadlock avoidance. Noteworthy, there is no significant throughput difference around the peak of 17.5% grid density. However, for densities starting from 25% the claim of Vis [14] seems true that deadlock handling including deadlock avoidance performs a bit better than pure deadlock detection and recovery. On the other hand, pure deadlock detection and recovery with the Least Extra Costs Path ranking is significantly computationally less expensive than deadlock avoidance. The most interesting range of densities is around the maximum throughput. In this density range, pure deadlock detection and recovery takes approximately four times less time than monocycle avoidance in combination with detection and recovery. The benefit of pure deadlock detection and recovery can even be increased, if the avoidance algorithm is not uselessly called if deadlock avoidance is turned off.

In general, the use of deadlock detection and recovery in combination with or without deadlock avoidance can result in almost the same behaviour. One precondition is that deadlocks are quickly detected. As stated at the beginning of this section, disabling deadlock avoidance potentially increases throughput. Figure 7.16 shows that disabling deadlock avoidance nevertheless does not clearly increase throughput. This result is explained by the number of deadlocks that occur. Figure 7.13 shows that about fifty deadlocks occur if deadlock avoidance is disabled. This means that in simulations with deadlock avoidance, only fifty tile reservations are prohibited by deadlock avoidance. Fifty tile reservations are only $\pm$ 0.1% of all reservations. Thus, the possible throughput gain of disabling deadlock avoidance is very small. The small throughput gain by disabling avoidance is cancelled by the extra waiting time to detect a deadlock.

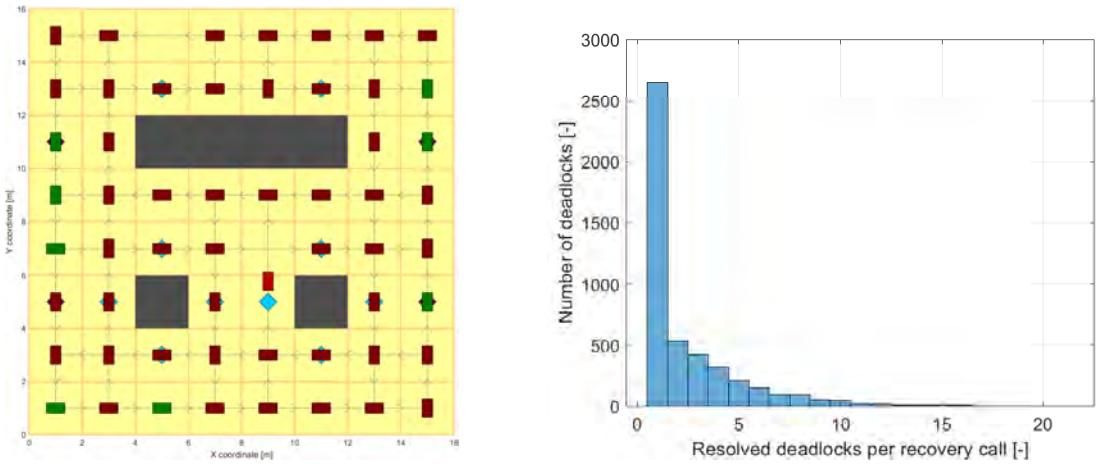
However, only using deadlock detection and recovery can in theory perform better deadlock handling that includes deadlock avoidance. An avoidance action is almost the same as a deadlock detection action. Deadlock avoidance detects a deadlock just before it occurs. Deadlock detection

detects a deadlock just after a deadlock occurred. Assume the deadlock detection interval is very small and deadlocks are detected very quickly. Then, the waiting time for involved AGVs between deadlock occurrence and recovery is negligibly small. The only difference between deadlock avoidance and deadlock recovery that is left is path planning. An AGV has to wait if a deadlock is avoided, but its path remains the same. But, the path of an involved AGV changes if a deadlock is recovered. Deadlock recovery can force that an AGV gets a recovery path that is beneficial for the throughput. A recovery path might lead an AGV through a less congested area if, for instance, by default the shortest path is chosen. The less congested path is not chosen if the deadlock is avoided, since deadlock avoidance does not change paths. So, deadlock recovery can improve throughput compared to using deadlock avoidance. On the other hand, throughput will suffer from planning a recovery path if the paths of AGVs are generally close to the optimal path. In the Grid-Based Controls model, periodic rerouting happens based on a dynamic path planning algorithm. So, the paths of AGVs are expected to be close to their optimal path in the Grid-Based Control model. It is therefore unlikely that disabling deadlock avoidance yields a throughput increase.

To conclude, using short deadlock detection intervals is crucial to minimise throughput loss due to deadlocks. Only using deadlock detection and recovery as a deadlock handling technique can equal the throughput performance of deadlock handling in combination with deadlock avoidance. However, it is unlikely that throughput can be improved by disabling deadlock avoidance. Using the simplest deadlock avoidance algorithm, instead of the most expensive, is sufficient with the addition of deadlock recovery. Only using deadlock detection and recovery as a deadlock handling technique is the best choice if layouts are insensitive to deadlocks or computation time needs to be trimmed.

7.4 Reachability

Section 5.4 discusses edge cases of the deadlock recovery algorithm. The edge cases prevent AGVs from reaching their destination, either because a deadlock could not be recovered, an infinite recovery loop happens or a livelock occurs. During none of the extensive simulations used for this chapter, a deadlock could not be recovered or an infinite recovery loop occurred. The typical layouts and density used by Vanderlande make it unlikely that these edge cases occur. To get a feeling if the above edge cases do occur at all, a stress test is performed with the small layout of the exploratory simulations.



(a) Layout used for stress tests with only two empty tiles (97% density)

(b) Histogram of number of recovery iterations per recovery call before system is deadlock-free

Figure 7.17: Stress test to test edge cases.

In the layout only two tiles are not filled with an AGV, creating a density of 97%, see Figure 7.17a. The test is done using the Random resolvability ranking. The simulation duration was 24 hours. At the end of the simulation, all deadlocks could be recovered. Each AGV did between 6 and 12 drop-offs during the 24 hour simulation period. This gives confidence that a livelock did not occur. The fact that no deadlock recovery loop occurred can be seen in Figure 7.17b. It shows the number of deadlocks that needed to be recovered before the system was deadlock-free. The recovery of by far the most deadlocks did not result in new deadlock, even with the density of 97%. The maximum number of iterations needed within one deadlock recovery to make the system deadlock-free call is 21.

The results of this section indicate that there are no reachability issues with the implemented deadlock recovery algorithm. Apart from reachability, this chapter discusses the result of full-scale simulations of some variants of the deadlock recovery algorithm. The goal of the full-scale simulations is to appoint the best performing deadlock recovery algorithm variant. Section 7.2 shows that the Least Extra Costs Path ranking performs the best. Assigning a resolving AGV based on this ranking lowers the congestion the most, which gives the highest throughput. The throughput performance of this ranking comes at the cost of the highest computation time. However, Section 7.3 shows that computation time of deadlock handling can be lowered compared to using deadlock avoidance, despite using the most expensive deadlock recovery method. Furthermore, the throughput, when handling deadlocks without deadlock avoidance, can equal the throughput of deadlock handling including deadlock avoidance. Next to the conclusion on the full-scale simulations, the overall conclusion of the project is given in the following Chapter 8.

8 Conclusion

The development of a deadlock recovery algorithm for grid-based AGV systems is presented in this report. Such an algorithm is needed as the current deadlock handling technique cannot avoid all deadlocks. A lot of research is done on deadlock prevention and avoidance, however, deadlock recovery for physical systems is an underexposed research area. Literature shows that a deadlock in an AGV system can only be recovered by breaking the circular wait condition of a deadlock. The path of at least one involved AGV needs to be changed to do so. Dedicated deadlock infrastructure, like buffers, can be used to make sure an AGV can always divert directly. However, such dedicated infrastructure is not chosen because of the use of space. Instead, it is shown under which conditions all deadlocks can be recovered without the use of dedicated deadlock infrastructure. To do so, an algorithm is presented including system requirements that let AGVs always reach their destination.

The actual implemented and tested deadlock recovery algorithm in the used Grid-Based Controls model is different. The implemented algorithm is based on a modular framework in which each sub-algorithm can be adjusted without influencing the consecutive calculations. The framework is established after identifying the necessary recovery steps. The recovery steps are found by analysing the formation of deadlocks in a range of settings and layouts. The four main parts of the framework are: creating a resolvability ranking of involved AGVs, appointing AGV(s) as resolving AGV(s), planning a recovery path and, finally, finding an extra resolving AGV if the initial resolving AGV(s) could not resolve the deadlock. In the first main part, a resolvability ranking is created. The ranking is based on which AGV is expected to be the most suitable resolving AGV. Multiple ranking methods are tested. It is shown that algorithm variants that lower congestion around a deadlock the most, result in the highest system throughput. Of all tested ranking algorithms, the Least Extra Costs Path ranking is crowned as the best ranking. To create this ranking, the recovery paths are computed for each involved AGV. Based on the expected recovery path, the relative extra costs of the recovery path compared to the current path is calculated. The AGV with the least extra costs ends on top of the resolvability ranking. The additional costs are partially based on the expected congestion on the recovery path. Next to that, the additional costs of a recovery path tend to be low for AGVs with a long remaining path. Thus, the resulting ranking generally provides a resolving AGV that wants to get away from the congested area (long remaining path) and can easily do so (low congestion on the path). It is found that calculating a path for each involved AGV, to construct the resolvability ranking, is relative computationally expensive. The other tested ranking algorithms only use already available system information. The result is that a recovery action with the Least Extra Costs Path ranking can be four times as expensive as using other tested rankings.

Secondly, resolving AGVs are appointed based on the resolvability ranking. The AGVs at the top of the ranking are selected to be given a recovery path. It can be concluded that increasing the number of resolving AGVs is generally not useful. It is only useful if a path planner is used with dynamic costs and no or very infrequent rerouting happens. In these cases, a path with lower costs might be found by the forced reroute action of deadlock recovery. The current standard path planning algorithm is used to plan recovery paths. No new path planner is developed to keep code maintenance easy. Next, an extra resolving AGV needs to be found if the initially appointed resolving AGV could not resolve the deadlock. A method based on the path of the initial resolving AGV and a method based on the resolvability ranking are compared. The two methods did not show significant differences in system throughput. Thus, the simplest method based on the resolvability ranking is chosen in the final deadlock recovery algorithm.

To make sure the system is deadlock-free at the end of the deadlock recovery algorithm, prohibited segments are applied. Prohibited segments receive a high weight during path planning, by which a different path is forced. The application of prohibited segments creates a depth-first search over AGVs in the system to find a deadlock-free solution. The deadlock recovery algorithm introduces

three possible edge cases in which AGV cannot reach their destination. But, the layouts used within Vanderlande are very insusceptible to these edge cases.

With the implementation of the final deadlock recovery algorithm two deadlock handling techniques can be used: deadlock avoidance and, deadlock detection and recovery. The performance of different deadlock handling combinations is compared. It is shown that the system throughput is greatly dependent on the length of the deadlock detection interval, when only deadlock detection and recovery is used. Only using deadlock detection and recovery as a deadlock handling technique can equal the throughput of using deadlock handling with deadlock avoidance. However, it is unlikely that throughput can be improved by disabling deadlock avoidance. A benefit of the added recovery algorithm is the significant decrease in computation time for deadlock handling. Simulations show that deadlock avoidance can be scaled down to the least expensive variant. Only using deadlock detection and recovery as a deadlock handling technique is the best choice if layouts are insensitive to deadlocks or computation time needs to be trimmed.

In conclusion, by using the developed deadlock recovery algorithm it can be guaranteed that the system becomes deadlock-free. Using the algorithm does not only prevent the system from stalling in case of a deadlock, but can also reduce the overall computation time of deadlock handling.

8.1 Recommendations

It is shown that lowering congestion in deadlock areas increases the throughput of a system. The effect indicates that throughput can be increased if general congestion control methods are improved. Methods that predict congestion during path execution could be developed for traffic control. Jobs assignment can also be improved to prevent congestion around pick-up points.

During this project a deadlock recovery algorithm is developed and implemented in the existing Grid-Based Controls model of Vanderlande. A balance is found between, on one side, ease of implementation and maintainability of the algorithm and, on the other side, being able to guarantee deadlocks can be recovered and no livelocks occur. By the current implementation, no absolute guarantee can be given on deadlock recoverability and the prevention of livelocks. It is not believed that this problematic behaviour will occur, given the controls and typical layouts currently used within Vanderlande. However, if some deadlocks cannot be resolved, path planning can be adjusted to guarantee all deadlocks are recovered. The needed changes are described in Section 5.4. And, further research can be put into preventing livelocks if they do occur after all. For instance, by introducing randomness in the deadlock recovery algorithm.

Furthermore, all AGV control algorithms are geared towards preventing and avoiding deadlocks, since no deadlock recovery algorithm was in place. Now a deadlock recovery algorithm is implemented, the costs of experiencing a deadlock are lowered dramatically. Therefore, an analysis can be done to see which current AGV controls can be improved. As an example, the deadlock avoidance action can be executed in parallel to speed up the computations. Currently, the tile reservations of all AGVs are checked sequentially. An extra tile reservation is allowed if the reservation is not expected to lead to a deadlock. The tile reservations are updated after each AGV is checked, to be able to accurately predict a deadlock. The avoidance check could be performed in parallel for each AGV, after which the tile reservations are updated once. This can speed up the calculations, at the risk of introducing deadlocks.

Lastly, a short deadlock detection interval is important to achieve the best throughput. Deadlock detection currently happens at fixed intervals during which all AGVs are checked. However, ways could be found to shorten the deadlock detection interval without increasing the computation time. Generally, a lot of AGVs are simply making progress when deadlock detection is run. An improvement could be that deadlock detection is only performed for AGVs that did not reserve a new tile within a certain period, which could be a clue that an AGV is involved in a deadlock.

References

[1] K. Fransen, J. van Eekelen, A. Pogromsky, M. Boon, and I. Adan, "A dynamic path planning approach for dense, large, grid-based automated guided vehicle systems," *Computers & Operations Research*, vol. 123, p. 105046, 2020.

[2] Vanderlande.com, "About Vanderlande," 2021, accessed on 2021-12-06, https://www.vanderlande.com/about-vanderlande.

[3] ——, "Parcel," 2021, accessed on 2021-12-06, https://www.vanderlande.com/parcel.

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

[5] Addverb, "Zippy — sorting robot for intelligent sortation from Addverb," 2020, accessed on 2021-12-06, https://www.youtube.com/watch?v=gDe-ErKLoks&ab_channel=AddverbTechnologies.

[6] G. Bartelet, "How autonomous vehicles can add value to the baggage handling process at airports," Vanderlande, Whitepaper, Apr. 2020.

[7] E. van Meijl, "FLEET bag value case analysis," Vanderlande, Presentation, Jan. 2020.

[8] BastianSolutions, "Optimize storage space with this easy-to-expand, automated goods-to-person shuttle system," 2019, accessed on 2021-12-06, https://www.bastiansolutions.com/solutions/technology/goods-to-person/vanderlande-adapto-shuttle-system/.

[9] R. Curley, "Robots to aid passengers checking in luggage at Dallas fort worth," july 2019, accessed on 2021-12-06, https://www.businesstraveller.com/business-travel/2019/07/14/robots-to-aid-passengers-checking-in-luggage-at-dallas-fort-worth.

[10] J. van Eekelen, "AgvSorter model," Vanderlande, 2021, accessed on 2022-01-03, https://devcolla.vanderlande.com/display/SIMTOOLS/AgvSorter+model.

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

[12] E. G. Coffman, M. Elphick, and A. Shoshani, "System deadlocks," *ACM Computing Surveys (CSUR)*, vol. 3, no. 2, pp. 67–78, 1971.

[13] Z. W. Li, N. Q. Wu, and M. Zhou, "Deadlock control of automated manufacturing systems based on petri nets — a literature review," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 4, pp. 437–462, 2011.

[14] I. F. Vis, "Survey of research in the design and control of automated guided vehicle systems," *European Journal of Operational Research*, vol. 170, no. 3, pp. 677–709, 2006.

[15] R. L. Moorthy, W. Hock-Guan, N. Wing-Cheong, and T. Chung-Piaw, "Cyclic deadlock prediction and avoidance for zone-controlled agv system," *International Journal of Production Economics*, vol. 83, no. 3, pp. 309–324, 2003.

[16] N. Viswanadham, Y. Narahari, and T. L. Johnson, "Deadlock prevention and deadlock avoidance in flexible manufacturing systems using petri net models," *IEEE Transactions on Robotics & Automation Magazine*, vol. 6, no. 6, pp. 713–723, 1990.

[17] Y. T. Leung and G.-J. Sheen, "Resolving deadlocks in flexible manufacturing cells," *Journal of manufacturing systems*, vol. 12, no. 4, pp. 291–304, 1993.

[18] R. A. Wysk, N.-S. Yang, and S. Joshi, "Resolution of deadlocks in flexible manufacturing systems: avoidance and recovery approaches," *Journal of manufacturing systems*, vol. 13, no. 2, pp. 128–138, 1994.

[19] H. Cho, T. Kumaran, and R. A. Wysk, "Graph-theoretic deadlock detection and resolution for flexible manufacturing systems," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 413–421, 1995.

[20] M. Fanti, "Digraph-theoretic approach for deadlock detection and recovery in flexible production systems," *Stud. Inf. Control*, vol. 5, no. 4, pp. 373–383, 1996.

[21] W. Yeh, "Real-time deadlock detection and recovery for automated manufacturing systems," *The International Journal of Advanced Manufacturing Technology*, vol. 20, no. 10, pp. 780–786, 2002.

[22] A. Lankes, T. Wild, A. Herkersdorf, S. Sonntag, and H. Reinig, "Comparison of deadlock recovery and avoidance mechanisms to approach message dependent deadlocks in on-chip networks," in *2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*. IEEE, 2010, pp. 17–24.

[23] M. Lehmann, M. Grunow, and H.-O. Günther, "Deadlock handling for real-time control of agvs at automated container terminals," in *Container Terminals and Cargo Systems*. Springer, 2007, pp. 215–241.

[24] N. Wu and M. Zhou, "Resource-oriented petri net for deadlock resolution in automated manufacturing systems with robots," in *2006 IEEE International Conference on Systems, Man and Cybernetics*, vol. 1. IEEE, 2006, pp. 74–79.

[25] Q. Zhou and B.-H. Zhou, "A deadlock recovery strategy for unified automated material handling systems in 300 mm wafer fabrications," *Computers in Industry*, vol. 75, pp. 1–12, 2016.

[26] K. Im, K. Kim, Y. Moon, T. Park, and S. Lee, "The deadlock detection and resolution method for a unified transport system," *International journal of production research*, vol. 48, no. 15, pp. 4423–4435, 2010.