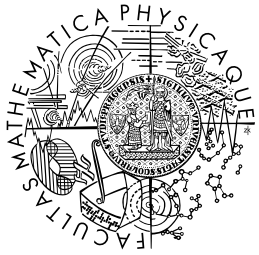


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Peter Kochelka

Planning for container warehouses

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis: Prof. RNDr. Roman Barták, Ph.D.

Study programme: Computer Science with specialisation in Artificial Intelligence

Prague 2025

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor prof. Barták for his valuable guidance and support during my work on this thesis.

Title: Planning for container warehouses

Author: Peter Kochelka

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Prof. RNDr. Roman Barták, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: This thesis focuses on the design of techniques for planning the sequence of actions used in a warehouse, where cranes are employed to move containers between cargo trucks and storage locations (in the style of the Blocks World Problem). The thesis proposes a decentralized and prioritized local planning approach to find the paths of container trucks. It is then combined with multiple proposed stacking policies of varying complexity and experimentally compared.

Keywords: planning, warehouse, container, multi-agent, decentralized

Název práce: Plánování skladů kontejnerů

Autor: Peter Kochelka

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Prof. RNDr. Roman Barták, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Práce se zabývá návrhem technik pro plánování posloupnosti akcí, kterými se ve skladu pomocí jeřábů přesouvají kontejnery mezi nákladními vozy a úložnými místy (ve stylu Blocks World Problem a Hanojských věží). Práce navrhuje decentralizovaný a prioritní lokální přístup k plánování cest kontejnerových vozidel. Tento přístup následně kombinuje s několika navrženými strategiemi různé složitosti pro ukládání kontejnerů a experimentálně je srovná.

Klíčová slova: plánování, sklad, kontejner, multi-agentní

Contents

Introduction

As world trade has grown rapidly over the past century, the number of containers shipped worldwide every day has also increased significantly. The performance of port warehouses is one of the crucial factors in optimizing the efficiency of cargo ships by reducing the time spent in ports.

Since warehouse terminals are being automated, the quality of operation planning becomes a key factor. Operators usually strive to achieve the highest possible speed of importing or exporting a given set of containers.

Container terminals typically operate several different types of agents. These agents include automated cranes, which are responsible for storing containers by stacking them on top of each other in multiple stacks of restricted height, as well as for retrieving containers from these stacks. Agents usually also include container trucks that bring containers into the warehouse or export them. Although there might be other agents as well, they are typically just variants of these two main types.

When planning paths for container trucks, we have to take into account that assigning containers to stacks is typically done in real time due to better scalability. As a result, the assigned stacks are not known in advance. This thesis therefore proposes a straightforward online planning approach for quickly finding container truck paths combined with a decentralized, prioritized, and local collision and deadlock avoidance protocol. This combined approach, although likely suboptimal, is flexible and works well with online container assignment. It is also robust against unexpected truck delays, which makes it suitable for real-life applications. The proposed approach then serves as a framework for experimentally comparing the performance of multiple proposed stacking policies of varying complexity.

The rest of the thesis is organized as follows. In chapter 1, the problem is stated more precisely. The warehouse environment and an instance of the problem are described, along with the necessary properties of a valid solution. The problem is divided into two main subproblems, the *Container stacking problem* and the *Multi-Agent Pathfinding* problem in chapter 2. The works and problems related to both subproblems, as well as their combination, are then examined, along with existing techniques. An approach for obtaining solutions, including its simplifications and assumptions, is proposed and analyzed in chapter 3. Its variants are then experimentally compared and analyzed in chapter 4. Attached is a custom-built simulation environment that implements the proposed solving approach. Attachment A.1 contains a description of its architecture. Attachment A.2 then provides an overview of its basic usage and functionalities.

1 Problem definition

This chapter provides formal definitions of the problem, the warehouse environment, and the agents, with their respective responsibilities and actions, as well as the imposed constraints, assumptions, and simplifications. It concludes with a definition of what can be considered a valid solution and how we can determine its quality.

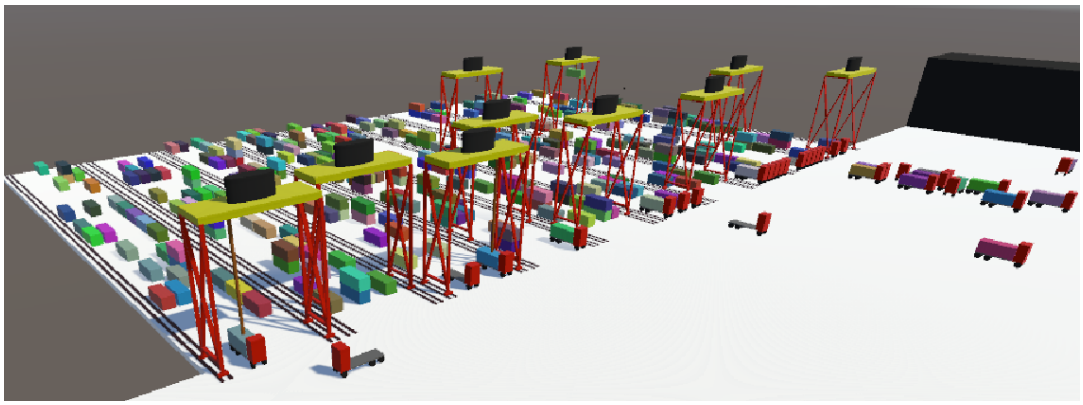


Figure 1.1 A warehouse during its operation

1.1 Environment

The problem environment is a container warehouse (see Figure 1.1). The containers it can store are standardized and all have the same dimensions. The warehouse floor is divided into two adjacent rectangular sections oriented parallel to the north-south and east-west axes (see Figure 1.2).

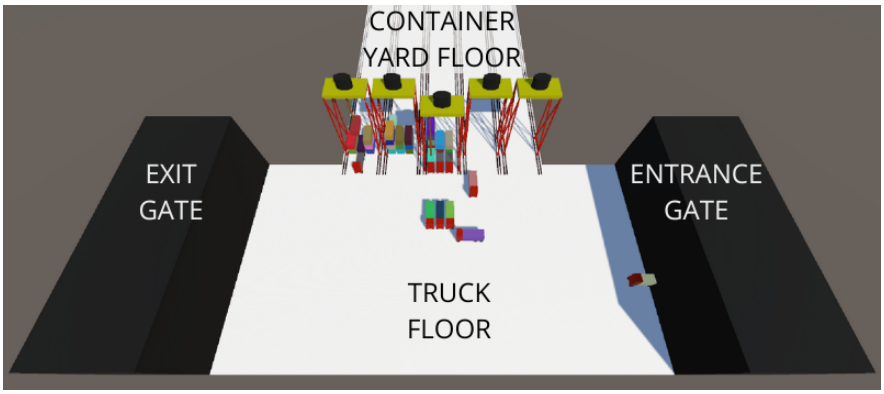


Figure 1.2 A view of a warehouse from above

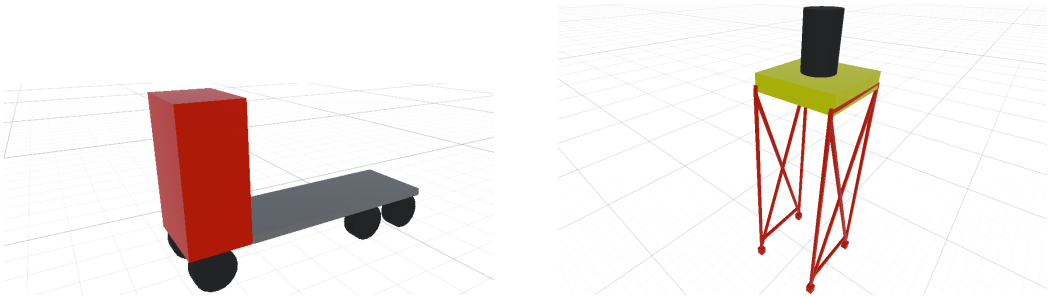
The southern section is the *truck floor*. It is large enough, so the trucks can perform all their movements safely without ever leaving it. There are two gates in this section. The gate located at the eastern end is the only entrance for incoming trucks, while the gate at the western end is the only exit for outgoing trucks.

The northern section is the floor of the *container yard*. It consists of rows of stack slots (on which the container stacks are built) that run perpendicular to the

truck floor. Along equally sized groups of rows are laid crane tracks whose length restricts the movement of the cranes. The tracks extend a little to the truck floor, to allow cranes to offload and load trucks when they reverse into this extended space (as seen in Figure 1.2).

1.2 Agents

The warehouse environment includes two distinct types of agents that cooperate in container transport, namely gantry cranes, responsible for container loading and unloading, and trucks, which transport containers between cranes and gates (see Figure 1.3).



(a) Container truck **(b)** Gantry crane

Figure 1.3 Warehouse agents

Container trucks are standardized, all of the same length, height, width, and speed. They enter the warehouse sequentially through the eastern entrance gate. They import or export exactly one container and then leave through the western gate. To import or export a container, a truck must rotate and reverse into the loading area of a crane, which then loads or unloads one of its containers. The truck then has to rotate again to get to the exit gate. The warehouse can operate multiple trucks at the same time as long as they never collide and never form a deadlock.

Gantry cranes are also standardized. Every crane is assigned to a unique, non-overlapping, and equally sized group of rows in a way that all rows are covered. Every crane is of the same height (which restricts how tall the stacks can be) and moves along tracks of the same length (which restrict the axis and the range of its movement). A crane is responsible for loading and unloading containers from trucks and then storing or retrieving them from the stacks in its rows. It can move only along the north-south axis and only on its crane tracks. The most important part of a crane is its cable. It can move between the rows of its crane, and when the crane is not moving, it can also stretch or retract. In the process, it picks up, transports, or releases the containers. The cable can access only the containers that are on top of the stacks for which the crane is responsible. It can put a container it holds only on an empty waiting truck or on top of a stack as long as it is not too tall. The maximum height of a stack is restricted by the height of the crane, as it requires some space to move a container above its stacks. To access a container that is not on top of a stack, it first has to move the containers stacked on top of that container to different stacks. We will refer to this process

as an in-warehouse reshuffle. A crane can also reorganize its stacks independently of the actions of other agents.

1.3 Problem specification

An individual problem is specified by the dimensions of the environment, a container sequence, and a truck sequence.

The dimensions of the environment specify how many cranes are available in the warehouse, how many rows a crane operates, how many stacks are in a row, and the maximum number of containers a stack can hold. They also specify how many container trucks are allowed to operate at once within the warehouse and how many paths they can use for their movement.

The flow of containers into the warehouse follows a predefined sequence that the warehouse receives in advance. Specifically, it receives a sequence of container IDs in the order in which the trucks will bring them through the eastern gate to be stored in the warehouse. The priority of export through the western gate of the containers stored in the warehouse is then determined by their IDs, with lower IDs indicating higher priority.

In a manner similar to the container flow, the flow of trucks into the warehouse follows a predefined sequence that the warehouse receives in advance. The sequence of trucks specifies whether the next truck will be importing or exporting a container. An importing truck will bring the next container from the container sequence to a crane assigned at the entrance gate, wait until it is unloaded, and then leave through the exit gate. An exporting truck will enter the warehouse empty, and at the entrance gate it will be assigned the container with the highest priority (the lowest ID) from among those in the warehouse. It will move into the loading area of the crane responsible for the corresponding stack, wait until it is loaded, and then leave through the exit gate.

1.4 Solutions

A valid solution to the problem is a sequence of actions that results in the entire container sequence specified in Section 1.3 being successfully imported and exported by trucks that follow the import/ export sequence specified in Section 1.3 without any collisions. The actions required to import or export a single container include truck movement from the entrance gate to a crane (along with the necessary rotation), container pick-up by the crane, movement of the crane with the container, release of the container, and truck movement from the crane to the exit gate. In case of retrieving a container that has other containers stacked on top of it, in-warehouse reshuffles have to be performed before accessing it.

We can compare the quality of valid solutions. There are four main parameters on which we will compare them:

- The duration of execution, or the time it takes for all the required actions to finish execution. This is typically called makespan
- The number of in-warehouse container reshuffles

- The total distance covered by individual agents, or equivalently the cost of the solution in terms of the amount of fuel burned
- The proportion of time spent waiting, measured separately for all the agent types

For each parameter, the lower the value, the higher the quality of the solution. There is no clear relationship between the individual parameters as different metrics fit different situations. In this thesis, we will focus on the first two metrics.

2 Background and literature review

Planning for container warehouses, along with many similar problems, has already been studied extensively. This chapter provides a brief overview of selected related works. The main aspects of planning for container warehouses are Multi-Agent Pathfinding for container trucks and cranes, organization of containers into stacks, and the interaction between the two.

2.1 Container stacking

Container stacking problem is about determining how to algorithmically allocate containers to free and supported places within warehouse stacks to make them easily accessible by the cranes. The optimality of the solution has direct real-world consequences for warehouse operators. Some well-known examples of similar problems are the Towers of Hanoi and the Block World Problem.

The Towers of Hanoi puzzle starts with a number of various-diameter disks stacked on one of the three given rods ordered from the largest at the bottom to the smallest at the top. The goal is to move all of them to another rod, one at a time, while moving them only between the rods and never putting a disk on top of a smaller one. The container warehouse might get into a similar situation if we want to access the bottom container of a full stack, and thus we have to transfer all the containers stacked on top of it to other stacks.

Similarly, in *the Block World Problem*, we have a set of blocks of the same size and a flat surface. We want to reach a goal configuration starting from the initial one, by moving one block at a time. The blocks can be put on the table or on top of another block. When stacking containers, additional constraints include a limited number of stacks of limited height.

The *Container stacking problem* has been successfully modeled as an AI planning problem by Salido et al. [1]. They proposed a domain-specific heuristic to minimize container distance from crane truck loading area and balance stack heights. It has succeeded in significantly decreasing the planning time while also producing plans that require less crane movement compared to domain-independent heuristics. This approach would likely work well with a centralized, offline Multi-Agent Pathfinding algorithm for the movement of the trucks. Given enough time, it would be able to find optimal solutions, but it would likely be computationally expensive and scale poorly with the size of the warehouse.

Several stacking policies for online search algorithms, with a particular focus on automated container terminals, have been proposed and compared in simulations by Park et al. [2] as well as Dekker et al. [3]. Dekker et al. have studied terminals that handle containers of different sizes, making stacking more difficult. Containers were also prioritized according to a category they belonged to instead of a unique ID, so their order was more relaxed. However, their focus on crane workload is adapted to our setting later in this thesis as the Minimize crane workload policy (see section 3.3.6). Park et al. propose a stacking policy that weights several criteria for container stacking and progressively updates its weight vector. This

could be adapted to our setting and built on in future works.

He et al. [4] have studied yard crane management in automated container terminals with mixed import and export containers, whose allocation is typically separated in traditional terminals. They work with a more complex model than this thesis; for example, they let multiple cranes operate on the same group of rows, so they sometimes enter into a conflict.

2.2 Multi-Agent Pathfinding

In an environment with multiple agents moving at the same time, we usually want to plan their paths in a way that they will not collide when following them. This is what the Multi-Agent Pathfinding (MAPF) problem is concerned with. Stern et al. [5] have made an effort to unify the terminology and benchmarks of MAPF, while giving an overview of the state of research in traditional MAPF on graphs, as well as in several specific variants of MAPF.

The movement of cranes and their cables in a container warehouse can be modeled very efficiently as a traditional MAPF on a graph. In the proposed warehouse model, it is even more straightforward as no collision between cranes can occur. On the other hand, different variants of the MAPF would be more suitable for the movement of trucks.

For example, MAPF with large agents (agents that cover multiple vertices at once), which corresponds better to warehouse trucks, has been studied by Li et al. [6], although they considered only agents that cannot rotate, as well as by Yakovlev and Andreychuk [7], who focused on circular-shaped agents. Since warehouse trucks have an asymmetric shape and their rotation is important in determining the covered area, we cannot directly apply their results. Still, the idea of generalizing agents conflicts proposed by Li et al., adapted to account for the shape and rotation of the agents, will be developed later in the thesis. Similarly, prioritized planning as described by Yakovlev and Andreychuk will be adapted to better suit our setting.

The results of lifelong Multi-Agent Pickup and Delivery (MAPD) research, where agents repeatedly move to a pick-up location and then to a delivery location, while avoiding collisions, as studied for example by Ma et al. [8], could also be applied in the container warehouse setting. However, it would be more suitable for automatically guided vehicles that operate in some automated container terminals and do not leave them than for trucks that just enter and leave as in our setting. This instance of MAPF specifically applied to container terminals has been studied, for example, by Zhong et al. [9].

Online MAPF, as studied by Švancara et al. [10] could be applied in warehouses in which the truck incoming schedule is not known in advance and their paths have to be planned and adjusted continuously, as well as in warehouses, which follow container stacking policies and do not have all containers preassigned. For their approach to be robust toward truck delays during execution of their plans, we would likely need to be combined with some collision avoidance approach. In addition, the truck paths would probably need to be re-planned more often than in an ideal setting, which would decrease scalability. However, it could be explored further in future works.

Prioritized planning of agent paths, as described, for example, by Čáp et al.

[11] could then help the path-finding algorithm scale to more robots and find paths efficiently.

3 Solving approach

A detailed description of the solution method used follows. The *Multi-Agent Pathfinding* for the trucks is solved using a movement restriction which makes finding the paths straightforward, but gives no guarantees about the absence of collisions or deadlocks. It is therefore combined with decentralized prioritized online movement algorithm that ensures that the trucks do not collide nor do they form a deadlock when following the found paths. Since every crane is assigned its own separate, non overlapping tracks and can move only on them, the cranes can never collide with each other and we do not have to analyze their movement as precisely. For *container stacking*, several stacking policies are proposed to be implemented and experimentally compared with each other in the following chapter. To conclude this chapter, the validity of the method is discussed.

3.1 Container trucks

Achieving the desired properties of truck movement is the most complex part of the solution. We will begin by defining the necessary terms and describing the simplifications used. Next, we will describe the algorithm for cooperative movement of all trucks followed by the algorithm for the movement of an individual truck. We will conclude this section by proving that the algorithms have the desired properties.

To be able to formally prove that no collisions or deadlocks occur, we first need to formally define these terms.

Definition 1. *A **truck collision** is a non-empty intersection of two trucks with each other or with the containers they are carrying at an arbitrary moment. We want to avoid them, as they could significantly damage the involved trucks or containers.*

Definition 2. *A **deadlock** occurs when a group of trucks are simultaneously blocked in their movement (including transitive blocking) one by the other. This makes it impossible for the plan to finish its execution.*

When reasoning about the movement of container trucks, they can be reduced to two-dimensional rectangles (as if looking from above), since their rotation, position, length, and width are sufficient to fully describe them in relation to other trucks, cranes, or the warehouse. A collision is then simply an intersection of at least two such rectangles. To simplify reasoning even more, we will view the truck floor as a two-dimensional Euclidean space with the x-axis running straight from west to east and the z-axis running straight from south to north.

From a high-level perspective, every truck follows the same pattern during its movement (see Figure 3.1). Upon entering, it is assigned a row by the stacking policy (based on the container it should import or export). It moves in parallel with the x-axis until it is close enough to the assigned row. Then, it turns 90 ° to the left and reverses into the loading section of the crane responsible for that row. The crane loads or unloads the truck, which then moves forward and turns 90 ° to the right to proceed towards the exit gate, finishing its movement.

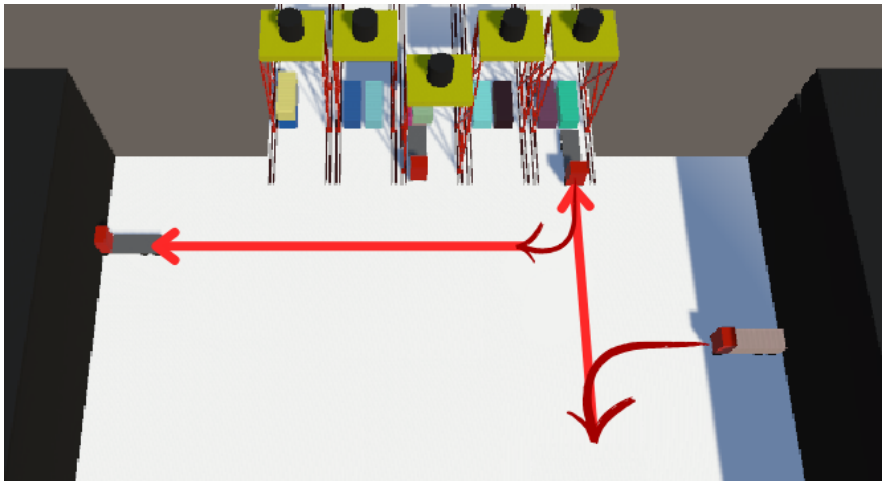


Figure 3.1 Truck movement outline

We will restrict the freedom of truck movement by dividing the truck floor into equally spaced paths, all running in parallel with the x-axis from east to west (from the entrance gate to the exit gate). Every truck will be allowed to move almost exclusively along the available paths. The only exception will be reversing into the loading area of its assigned row or leaving it to rejoin one of the paths. The distance between two neighboring paths will be large enough for one reversing truck to be able to stay between them and block neither of them. If more than one path is available, the one closest to the cranes is designated as the exit path. This means that the trucks will be leaving the warehouse using only this one path. The remaining paths are used to enter the warehouse. In case there is only one path available, the trucks use it both to enter and exit the warehouse. Hence, the exit path will be the most heavily used path, which will likely slow down plan execution, but it should not be significant. This is because, as we will see, anytime a truck is able to use a different exit path, it is almost always able to use this one.

As a consequence of this restriction, the entire path of a truck is uniquely defined by its choice of target row and entry path. We will discuss both of them later, but first we will describe an algorithm for truck movement which will ensure that when multiple trucks at once are following these intersecting paths, the trucks do not collide, nor do they form deadlocks.

3.1.1 Truck movement prerequisites

During their movement, the trucks follow a prioritized planning algorithm. We will proceed in short time frames of a known length. At the beginning of every time frame, we will order the trucks based on a priority ordering. Then, in order from high priority trucks to low priority, every truck will reserve an area that it will occupy at the end of the frame. For this algorithm, we will therefore need to define the priority ordering of the trucks, as well as the area reserving.

The trucks will be assigned priority at the beginning of every time frame based on how close they are heuristically to leaving the warehouse. If they plan their movement in this order, it allows the earlier trucks to leave the path and those behind them to move more freely.

First, trucks are ordered according to their current phase of movement. For

example, a truck heading to the exit gate would have priority over a truck rotating to join the exit pathway. The second truck would have priority over a truck that is reversing to the loading section of the crane. The third truck would have priority over a truck rotating to the left, which in turn would have priority over a truck that has not come close enough to its assigned row to even start rotating yet. Second, trucks are ordered from north to south. Therefore, trucks closer to the cranes move first. Finally, they are ordered from west to east. So, trucks closer to the exit gate move first.

The last thing we need to define before presenting the algorithm for the truck movement is the area reserving. An **area** is always a rectangle with sides parallel to the main axes. During a time frame, every truck starts its movement at a certain position, occupying an area that it reserved in the last time frame. When it gets to its turn in planning (all the trucks with higher priority have already planned their movement during this frame), it chooses a position it wants to occupy at the end of this time frame and reserves an area around it. There are two constraints. First, the position it wants to occupy must be a valid position on its remaining path (including its current position in case it does not move this frame). Second, the smallest rectangle with sides parallel to the main axes containing the entire area occupied right now, as well as the reserved area, but leaving a small *safe distance* (a constant, something like half the width of the truck) toward higher coordinates z and lower x (see Figure 3.2) does not intersect with the reserved areas of other trucks. In other words, the truck will not collide with them while moving during this time frame. The safe distance margin makes the truck that is at a position with lower z and higher x coordinates responsible for not getting too close from this direction to the area reserved by other trucks. If a truck cannot reserve the area it wanted, it will reserve a valid area that is the farthest possible from its current position. It is always able to reserve at least the area it started in.

Figure 3.2 Truck movement reserving (view from above)

3.1.2 Truck movement

The warehouse starts without trucks in it. A new truck can enter the warehouse only by an entry path and can do so as soon as there is enough space on the pathway. The next truck always enters the warehouse in the first time frame it is able to. However, if the truck is exporting a container and the truck importing that exact container has not been unloaded yet, the new truck has to follow *exactly the same* pathway to prevent faster arrival. Similarly, to preserve the order of trucks after entering the warehouse (and to prevent reshuffles caused by later

trucks overtaking earlier trucks), every exporting truck has to follow the same entry path as any truck already in the warehouse that is importing or exporting a container from the same crane. While there is not enough space on this path, it cannot enter the warehouse. Otherwise, if no special restrictions apply, the path with the smallest number of trucks is chosen.

The movement of every truck from the entry to the exit of the warehouse can be divided into 9 distinct phases, starting with the ROW phase, which follow one after another in linear order (see Figure 3.3 for their overview). Trucks in the same phase, as well as trucks moving in the same direction, move at the same speed. In every phase, the truck has a position target. Once reached, it can move to the next movement phase. However, moving to the next movement phase can only be done at the end of a time frame.

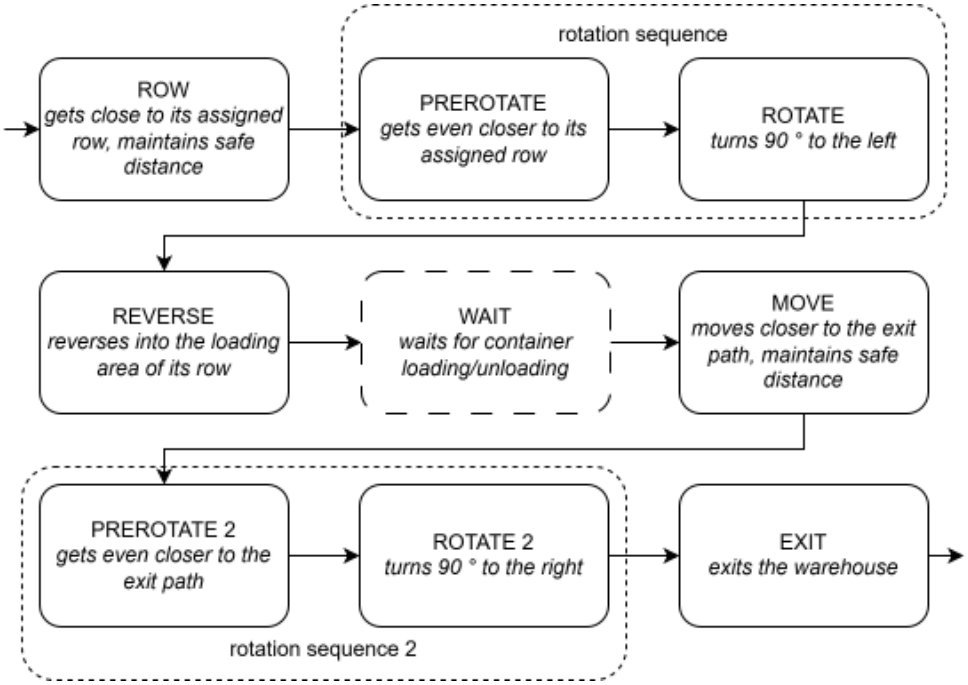


Figure 3.3 Truck movement phases

When a truck is reserving an area during a rotation sequence, it reserves the area it needs to finish the remainder of the sequence, even if it will need multiple time frames to do so. Otherwise, it reserves only the smallest rectangle that will contain it at the chosen position (see Figure 3.4). While it cannot reserve the area for rotation sequence, it stays in the previous movement phase.

In particular, if a truck is reversing and another truck is already at the loading section of its assigned row, it has to reserve its area in such a way that it does not intersect with the exit path. Otherwise, the truck in the loading section might not be able to use the exit path without colliding with the reversing truck. Furthermore, if **Smart Reversing** is enabled, every reversing truck that would be blocked by another reversing truck stops between two neighboring paths so that it does not block any of them (see Figure 3.5), which should result in increased throughput.

Every truck then plans its movement as follows, in the order from higher priority to lower priority. First, it computes the maximal distance that it can

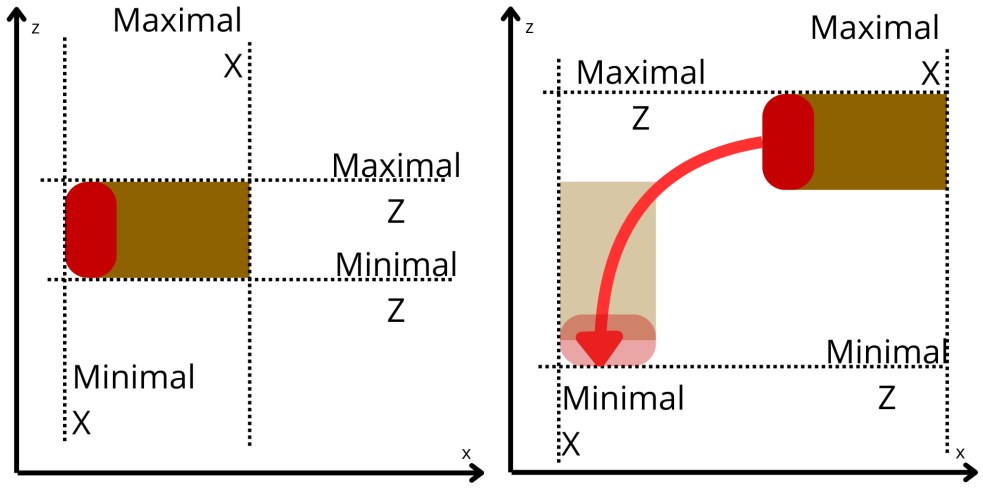


Figure 3.4 Truck reserving an area outside and during a rotation sequence (view from above)

cover in this time frame ($frame\ duration \cdot speed$). Second, it chooses a position along the remaining path that it can reach during its current movement phase. The position must be as far as possible, but it cannot be further than the maximal distance, and the truck must be able to safely reserve an area around it (without the enclosing rectangle, see Figure 3.2, intersecting with the reserved areas of other trucks).

Once all the trucks have planned their moves, they perform them and the planning process repeats itself in the next time frame until the solution execution is finished.

3.1.3 Desired properties of the truck movement

Now we shall formally prove that no collision and no deadlock occur when the above process is followed.

Collision-freeness

In this section, we will prove that at the end of every time frame there is no collision, so the discrete version of collision-freeness. We will proceed by contradiction. Then, we will use this result to show that for short enough time frames no collisions occur even within the time frames.

Theorem 1. *If every truck in the warehouse follows the process described above, at the end of every time frame there will be no collision.*

Proof. For contradiction, let us assume that every truck in the warehouse follows the movement process and there is a time frame at the end of which there is a collision. Without loss of generality, we can assume that only two trucks were involved.

Since no truck can enter the warehouse while it would collide with a truck on the entry path, the two trucks did not start their movement in a collision. Therefore, during some time frame, they have entered into it.

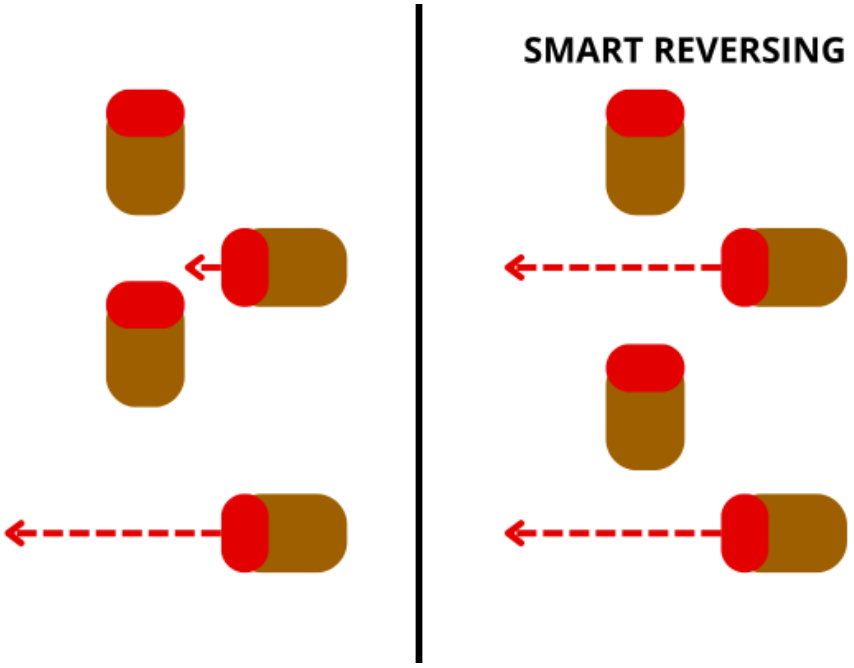


Figure 3.5 Ordinary reversing compared with smart reversing (view from above)

From how the traffic code priority order is defined, one of the trucks had a higher priority during that frame. It therefore has reserved its area first in such a way that it could have gotten to it without entering into a collision.

However, when the second truck was deciding where to move, it had to consider the newly reserved area of the first truck. Since the collision has occurred, the reserved area of the second truck must have had an intersection with the reserved area of the first truck, which is a contradiction to how the area reservation is carried out. □

Corollary. If every time frame has

$$duration < \frac{safe\ distance}{maximal\ truck\ speed}$$

then there are no collisions.

Proof. At the beginning and the end of every time frame, there are no collisions. Since trucks moving in the same direction move at the same speed, for two trucks to enter and leave a collision during that time frame, they have to be moving in different directions. Since trucks in a rotation sequence have their area reserved, no other truck can enter it and collide with them. Similarly, trucks in the MOVE phase rotate before joining the exit path, so they cannot collide with trucks in the EXIT state. This leaves us with trucks in the ROW or EXIT phase crossing the path of a reversing truck. The collision would therefore be perpendicular. During a time frame, every truck covers a distance of at most

$$duration \cdot maximal\ truck\ speed < safe\ distance < truck\ width$$

At the start and at the end of the time frame, the distance between the two trucks is at least as large as the *safe distance* (because of the safe distance margin; see

Figure 3.2). Therefore, to enter and leave the collision within one time frame, the trucks would need to cover at least twice the *safe distance* together. However, each of them can cover only smaller distances in this short time frame, which is a contradiction. □

Deadlock-freeness

In this section, we will prove that no deadlock occurs. We will proceed by contradiction.

Theorem 2. *If every truck in the warehouse follows the process described above, no deadlock can occur.*

Proof. For contradiction, let us assume that every truck in the warehouse follows the process described above and a deadlock occurs. As the warehouse does not begin with a deadlock, there is a frame in which the deadlock occurs for the first time.

Now, we should realize that no truck ever moves eastwards. Also, whenever a truck is moving south (rotations or the MOVE phase), it has an empty area big enough for it to finish this movement and then move north or west.

For multiple trucks to be involved in a deadlock, they cannot all be moving in the same direction. If they were moving towards north or towards west, the north-most or west-most truck would not be blocked by any of the other trucks, which breaks the deadlock definition. Similarly, if they were moving towards the south, they would be moving each in their own area, so they would not be blocked.

Therefore, in a deadlock, there must exist a pair of trucks that are moving in different directions. Since no truck can block a truck during its rotation, no truck involved in the deadlock will be rotating. They will therefore be moving straight in one of the three allowed directions (North, West, South). Now we can analyze all possibilities and show that no deadlock can occur.

If a truck moves towards north (REVERSE phase), it can not block a truck moving south (MOVE phase), because it stops before the exit path, which is the south-most point of the path of any south-moving non-rotating truck. In case it blocks a truck moving westward, it cannot be blocked by it and vice versa *simultaneously*, as they never collide, and thus only one of them is blocked by the second one. A truck moving towards west can block or be blocked by a truck moving south only on the exit way, but then the truck closer to the west is not blocked by the other truck, and no deadlock occurs either. □

3.2 Container cranes

A crane remains idle until a truck stops beside one of its rows. If the truck carries a new container, the crane unloads the truck and places the container on top of its assigned stack. If the strategy of **assigning at crane** is chosen, the container is still assigned to a stack first at the gate. The responsible crane may then reassign it to the most suitable of its stacks before placing the container on top of it.

If the truck arrives empty, the crane first moves every container stacked on top of the target container to different stacks according to the stacking policy. Once

the target container is at the top of its stack, the crane retrieves it and loads it onto the truck.

If multiple trucks are waiting for the crane (each in a different row), the crane chooses the exporting truck with the highest priority (the lowest container ID). If no truck is exporting, it chooses the importing truck with the lowest priority (the largest container ID). This is because the stacks operate on a last-in, first-out basis and this container would otherwise be more likely to block the retrieval of containers with higher priority.

3.2.1 A condition for container retrieval

Remark. A crane can access a container that has n other containers stacked on top of it as long as the remaining crane capacity (sum of empty container slots in its stacks) is at least n.

Proof. The crane has to move the blocking containers one by one onto the stacks with remaining capacity. As every crane has its own group of rows, nothing can interfere with this process. □

Corollary. If the remaining crane capacity is always at least as large as the maximal stack height, every container can be retrieved.

3.3 Stacking policies

The whole warehouse operates according to the chosen policy. Each imported container is assigned to a stack at the entrance gate, based on the current state of the warehouse. If the strategy of assigning at the crane is chosen, it might be reassigned once the truck delivers it to the crane, but only to one of the stacks the crane operates. When a blocking container has to be relocated in order to allow access to the containers below it, it is also reassigned to a different stack operated by the same crane according to the policy.

When assigning, a stack is considered full if the number of containers already assigned to it, inserted, and incoming combined, equals the allowed stack height. It is free as long as it is not full.

No policy can assign to a full stack, unless assigning at crane (see section 3.2 for definition) is allowed. When assigning at crane, the incoming containers could always be reassigned before they are inserted anywhere. Therefore, only the already stacked containers count in that case.

A crane is considered full if the stacks it operates have at most n empty spaces in total, n being the allowed stack height. This is because the bottom container of some full stack operated by this crane could become impossible to retrieve and the solution would fail (see 3.2.1). It is free as long as it is not full.

No policy can assign to a full crane, and only the free cranes along with their stacks will be considered. The number of containers should therefore not exceed the combined capacity of the cranes.

What follows is an overview of the available policies. The policies are described in an approximate order from simpler to more complex, which are expected to produce better results.

3.3.1 Random free stack

When assigning a container following this policy, a free crane is chosen randomly and then a stack is chosen randomly from among its free stacks.

3.3.2 First free stack

When following the First free stack policy, the stacks are filled sequentially. They are ordered from south to north and then from west to east (see Figure 3.6). Ordering ensures that cranes are not filled one by one. Instead, the workload is somewhat spread out, despite obvious sub-optimality.

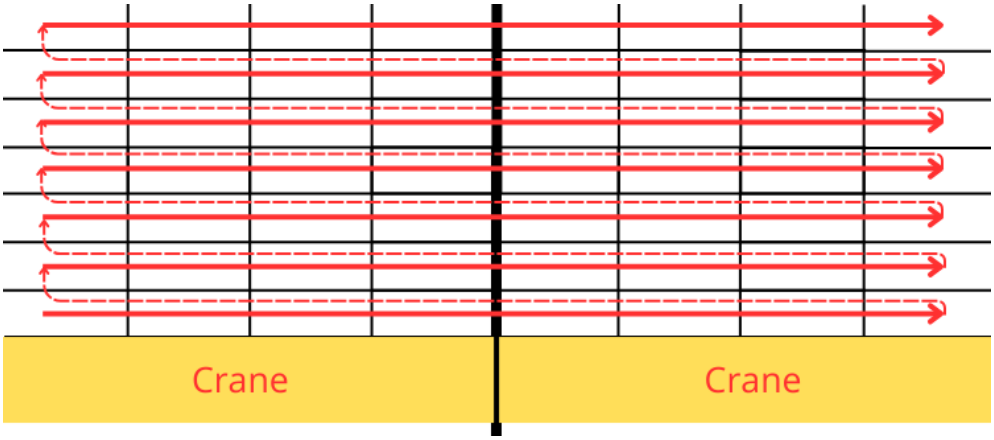


Figure 3.6 Stack ordering in First Free policy (view from above)

3.3.3 First bigger

The First bigger policy works with the same ordering of the warehouse stacks as the First free stack policy. However, instead of filling the stacks sequentially, it assigns the container to the first stack with ID of the top container bigger than that of the container being assigned, or the first empty stack. If there is no such stack, it simply follows the First free stack policy.

As a consequence, given enough space, less unnecessary container reshuffles are required, since the containers in every individual stack are sorted according to the recommended ordering.

3.3.4 Smallest bigger

The Smallest bigger policy builds on the First bigger policy. It still chooses from among the stacks that have ID of the top container larger than that of the container being assigned. However, instead of assigning the container to the first such stack, it picks the one where the difference of IDs is the smallest, or the first empty stack. If there is no such stack, it picks from all the free stacks the one where the difference of IDs is the smallest.

In this way, the IDs of the top containers of the stacks decrease more slowly. Thus, the average number of stacks used decreases, and their average height increases, when compared with the First bigger policy. This should result in

cranes covering less distance and their cables stretching less far (increasing the efficiency). Furthermore, it becomes more scalable, since the number of stacks required for the same number of containers is smaller on average.

3.3.5 Parallel

The Parallel stacking policy assigns according to the Smallest bigger policy, but with a different ordering. Containers are cyclically assigned to cranes. Only then, for each crane, its stack ordering goes from west to east and afterwards from south to north (see Figure 3.7). This policy is making better use of the multiple trucks and cranes available. The trucks spread more evenly between the cranes, so that they can unload them simultaneously.

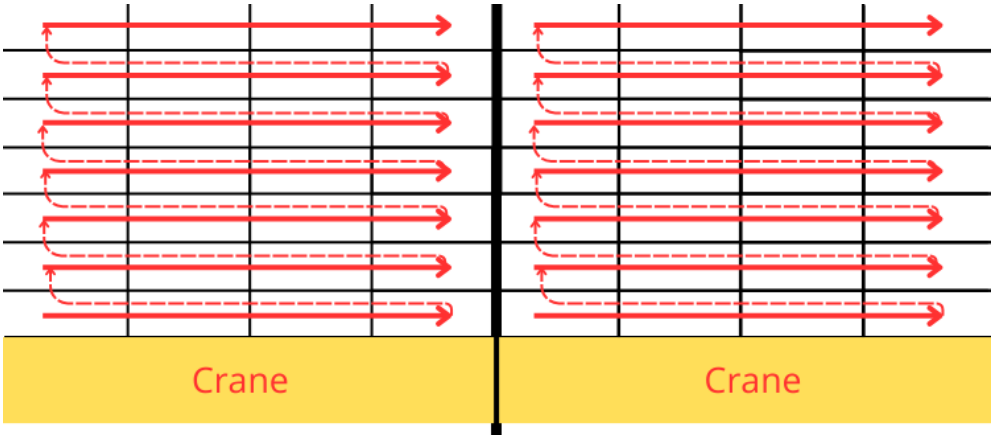


Figure 3.7 Stack ordering in Parallel policy (view from above)

3.3.6 Minimize crane workload

Although the Parallel stacking policy allows for mostly concurrent unloading of the import trucks, it does not guarantee anything about their loading at export, so it can still turn out not to be parallel at all. If multiple containers of similar priority are stored in the stacks of one crane, they are likely to be retrieved together in a short time frame. While concentrating the workload on this crane, the remaining cranes will probably not be used to their full potential.

To account for this, we propose a workload heuristic W of a crane c defined by the following formula:

$$W(c) = I(c) + E(c)$$

Here, $I(c)$ is the number of containers that are currently coming to c to be stored in some of its stacks, thus the present workload. On the other hand, $E(c)$ accounts for the future workload. It is the number of containers already assigned to c (incoming and stored) that have a difference in their priority (IDs) smaller than the number of available cranes. Those are precisely the containers that will likely be retrieved together in a short time frame. To see why, let there be n cranes, and out of a group of n containers with consecutive priorities (so the biggest pairwise difference between them is smaller than n), let at least two be

stored in the stacks of the same crane. From the Pigeonhole principle it follows that out of the remaining $n-1$ cranes, at least one will not be used to export any of the remaining $n-2$ containers from this group. Therefore, to increase the number of simultaneously working cranes (and, as a consequence, also the warehouse efficiency), we want to minimize the number of such groups.

The Minimize crane workload stacking policy builds on the Parallel stacking policy. Instead of assigning to the cranes cyclically, it chooses the one for which it increases its workload the least.

3.4 Properties of the method

The chosen method computes the solution step by step during its execution. Containers are assigned online as they come into the warehouse, which obviously does not guarantee stacking optimality as we are not using all the available information. However, the policies are fast and work well even with a large number of containers. Those more advanced also produce an output of reasonable quality. Furthermore, when using the strategy of **assigning at crane** (see section 3.2), they are robust even against delays in the truck arrivals to the cranes, as containers can always be reassigned.

The movement of trucks is solved by an online decentralized prioritized collision avoidance algorithm. Although an online method usually does not provide guarantees as strong as an offline method, it is robust to the delays of other trucks as well. It also works well with the stacking policies, since they are executed online, and thus the individual container assignments are not known in advance.

The cranes are isolated, so their movement is straightforward, and no collisions or deadlocks can occur there.

3.4.1 Every solution produced is valid

The most fundamental property that we expect from the proposed method is to produce solutions that are guaranteed to be valid.

Theorem 3. *Every solution produced by the proposed method is valid as long as the condition for container retrieval 3.2.1 always holds for every crane.*

Proof. We have already proven that trucks do not collide and do not form deadlocks. From the process description, it also follows that during the solution run, there will always be at least one truck in the warehouse or about to enter it.

As long as the condition for container retrieval is satisfied, the cranes can access every container stored in the stacks they operate.

Therefore, both the trucks and the cranes will always progress towards the goal state of the solution until they reach it. □

4 Empirical evaluation

4.1 Experimental setup

All experiments were run in the attached simulation environment (see Attachment A.2) on CPU *AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz* and 16 GB of RAM.

For all policies except for the Random policy, the simulation behaves almost deterministically. A small amount of noise affects mostly the time frame length as all the required calculations will usually take a different amount of time. This will usually impact mainly the duration of the whole simulation, but may also slightly affect other statistics. Therefore, solving the same problem instances should produce approximately equivalent results. Also, the experiments ran on a very high agent speed, so collision-freeness is guaranteed only at the end of the time frames, and the same problem instances may be solved a little differently at lower speeds.

4.2 Experiments

In this section, we are going to experimentally verify several hypotheses. Unless otherwise stated, we will be using the strategies of **smart reversing** and **assigning at the cranes** (see Chapter 3 for their definitions).

4.2.1 Relationship between time and number of trucks

The first hypothesis will be that allowing more trucks at the same time into the warehouse does not slow down the solution, as they either become helpful by increasing concurrence or they simply do not enter.

To test this hypothesis, we can inspect the mean solution run-time (in minutes) of five randomly generated problem instances for every policy - except the too slow First Free policy - when the warehouse has allowed 10, 20, 30, or 40 trucks at once on 1, 2, 4, 6, 8, and 10 paths (see Figure 4.1). For the higher path counts, 50 and 60 trucks have also been allowed. Each problem instance has been run in a warehouse with ten cranes, each responsible for five rows of 20 stacks able to store 15 containers. Therefore, the warehouses were large enough, allowing for a simple throughput, so the only focus is on how the number of trucks and paths affects its performance.

What we can see is that on one path, our hypothesis seems to hold, and adding more than 20 trucks almost does not affect the simulation duration. However, with more paths, it is likely not true. The graphs show curves shaped more like a "U", indicating that allowing too many trucks may, in fact, decrease the warehouse throughput, and we have to find a balance. Interestingly, this optimal number seems to be around 20 to 30 for most of the policies even across varying numbers of allowed paths.

Since the results were different than expected, we should analyze the reasons for the solution durations being longer when more trucks are allowed. When inspecting the runs on one path, we can see that the warehouse can not really hold

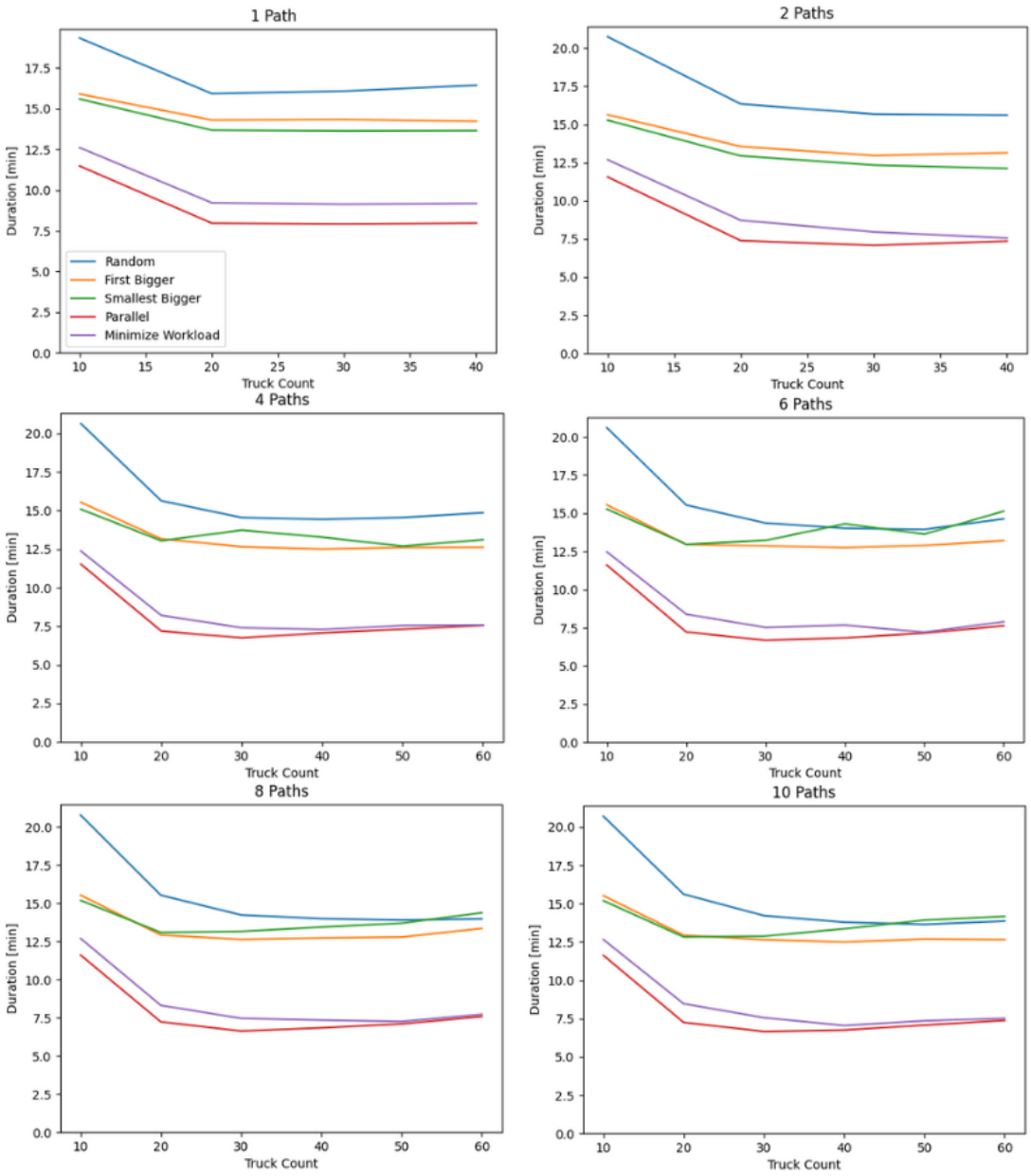


Figure 4.1 Relationship between time and number of trucks across path counts

more than 20 trucks, so allowing more to enter has no effect, just as observed. For more trucks, it is not as straightforward. We will therefore compare the Pearson correlation coefficient of different pairs of statistics of the simulation runs averaged over all policies except the outliers Random policy and First free policy for at most 30 trucks and for more than 30 trucks(see Figure 4.2).

For warehouses with at most 30 trucks, the duration is strongly negatively correlated with the total distances covered by the trucks, cranes, and their cables. This is expected behavior, since when they have to cover less distance at the same speed, they will do so faster. Although not as strongly, it is negatively correlated with the truck count and path count as well, which is the desired behavior. The positive correlation with the number of reshuffles and the average percentage of time frames that agents have to wait is not surprising.

Interestingly, the truck waiting percentage is correlated with the duration

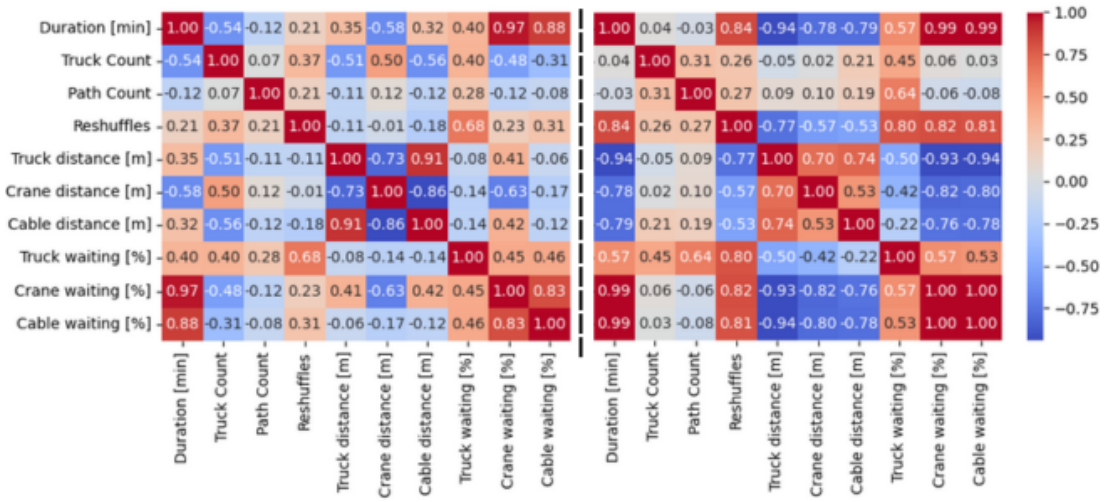


Figure 4.2 Heatmap of Pearson correlation between individual simulation run statistics with at most 30 trucks (on the left) and more than 30 trucks (on the right) averaged over all policies except for the outliers Random and First free

significantly less than the waiting percentage for cranes and cables. This could be explained by the strong correlation between the truck count and the truck waiting percentage. With more trucks, every individual truck has to avoid colliding with more trucks and therefore wait more often, but together they are able to import and export all the containers faster.

For warehouses with more than 30 trucks, the biggest difference is that the duration becomes (although weakly) positively correlated with the truck count. The reshuffles and the truck waiting percentage also have bigger impact. Since both are positively correlated with the truck count, this somewhat explains the slow increase in duration when allowing for more than 30 trucks. We should also account for the increased computational complexity of the simulation, as this is the reason time frames become visibly longer for more than 50 trucks, which slows down the simulation run.

Another observation we can make is that the Smallest bigger policy tends to perform significantly worse as we allow more trucks into the warehouse as opposed to other policies which worsen at a slower pace. Since we know that the reshuffle numbers are strongly correlated with the duration of the simulation runs, we will look at the relationship between the truck count and the mean reshuffle count averaged over all the path counts (see Figure 4.3) for the same solution runs as before.

Clearly, the Smallest bigger policy is the most sensitive to increases in truck count, observing an almost tenfold increase in the number of reshuffles. This is likely due to an imprecision in the truck movement algorithm. Despite order preservation (see Section 3.1.2) preventing the exporting trucks from overtaking other trucks that interact with the same crane, an importing truck can always overtake the exporting truck and place containers on top of the container being exported. The likelihood of this happening increases with more trucks available in the warehouse.

What further supports this explanation is the way other policies react to more trucks. In the above scenario, the export truck is exporting the container with

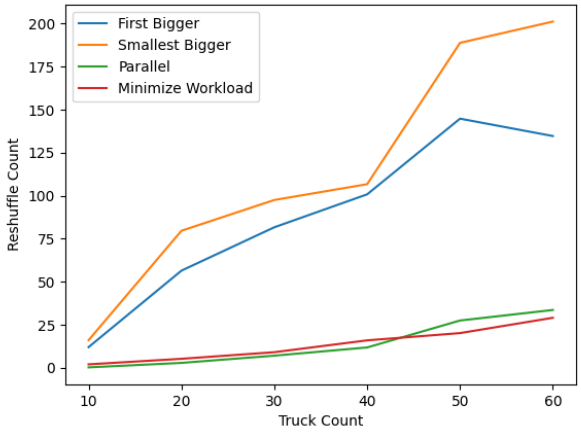


Figure 4.3 Relationship between number of trucks and number of reshuffles averaged over all path counts

highest priority (lowest ID) at the time of its entry. If shortly after its entrance a new truck importing a higher priority container enters the warehouse, and the stack of the container being exported is free, the imported container will be assigned to the same stack (since on its top is precisely the container with the smallest bigger ID). The importing truck is not bounded by the order preservation, so it is likely to choose another path with fewer trucks on it. If it successfully overtakes the exporting truck, the crane will have to perform a reshuffle of the imported container to retrieve the exported container. Although similar, the First bigger policy will choose from all the free stacks which have on top a container with bigger ID, decreasing the likelihood of choosing the stack with exported container. The Parallel and Minimize crane workload policies will very likely (since in the simulation runs analyzed there are 10 cranes available) import the container to a different crane, blocking the exported container only rarely. Interestingly, the reshuffle counts of the Parallel and the Minimize crane workload policies with 10 cranes seem to be very close to precisely one tenth of the reshuffle counts of the Smallest Bigger policy.

The last important observation we will make is about how the number of paths affects the simulation runtime. Interestingly, it makes barely noticeable difference, as the simulation runs have very similar durations even with varying numbers of paths. The main improvement occurs when we increase the number from one path to two, which allows the warehouse to use more trucks at once by providing more space.

4.2.2 Stacking policy comparison

In this section, we compare three pairs of policies and analyze the reasons for their behavior. The hypotheses when proposing them were that the First Free policy will be faster than the Random policy, the Smallest Bigger policy will be faster than First Bigger policy, and Parallel policy will be faster than Minimize Workload policy. We are going to examine all three hypotheses one by one, using the same simulation run data as in the previous section.

First, comparing the First free policy with the Random policy (see Figure 4.4),

we can see that the Random policy absolutely outperforms the First free policy. The reason becomes clear if we look at the average number of reshuffles. While for the Random policy, it is around 5 600, for the First free policy, it is almost five times bigger at around 26 800. This is likely due to the fact that while the Random policy usually spreads out the containers more or less evenly among all the stacks, the First free policy stacks them all into a smaller number of stacks (and does so without using any information about their order). As a consequence, they are blocked more often and by more containers at once. Furthermore, when reshuffling, the Random policy puts the container on top of a random stack, but the First free policy puts it again on top of the first free stack. Therefore, the same group of containers is usually moved to another stack as a whole but in reverse order, which means that they are likely to need repeated reshuffling. Also, since the First free policy has performed so poorly, we will not be using it in further experiments.

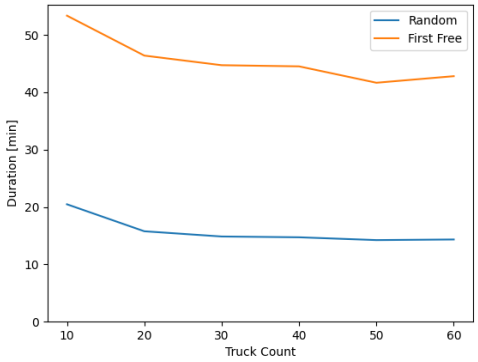
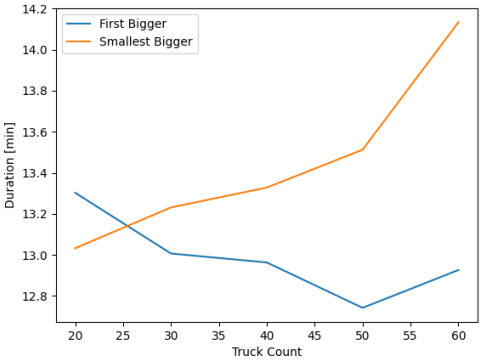


Figure 4.4 Comparison of the Random policy and the First free policy simulation durations averaged over all path counts

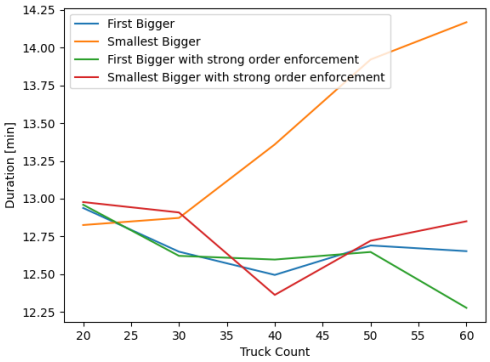
Second, if we compare the First bigger policy and the Smallest bigger policy (see Figure 4.5a), we see that they have similar runtimes, with the First bigger policy performing slightly better. However, this is an unexpected behavior, as the Smallest bigger policy was supposed to work better in theory. It is likely due to the aforementioned reason.

To verify whether this explanation holds and potentially find a way to improve the Smallest bigger policy, we will add a new option of using **strong order enforcement** strategy to the simulation, which will affect the importing trucks as well. Therefore, every truck will be forced to choose the same path as another truck interacting with the same crane. After running one experiment for this strategy for 20, 30, 40, 50 and 60 trucks on 10 paths and comparing it with the original run of the First bigger policy (see Figure 4.5b, we see that the results for the First bigger policy did not change much, but the results for the Smallest bigger policy improved dramatically. This is despite the restriction imposed on the truck concurrency (more strict conditions for a truck entering into the warehouse).

Third, a comparison of the Parallel policy with the Minimize crane workload policy (see Figure 4.6a) reveals that despite using more information, the Minimize crane workload policy performs consistently worse than the Parallel policy. This could be due to the fact that the future workload (see Section 3.3.6 for the



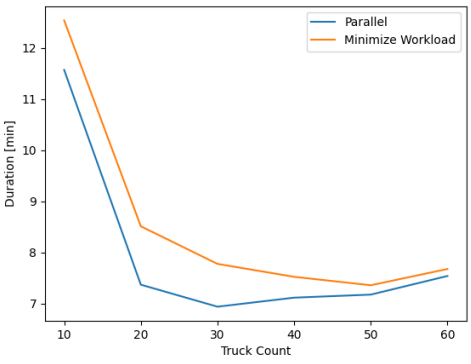
(a) Average solution runtime across all path counts



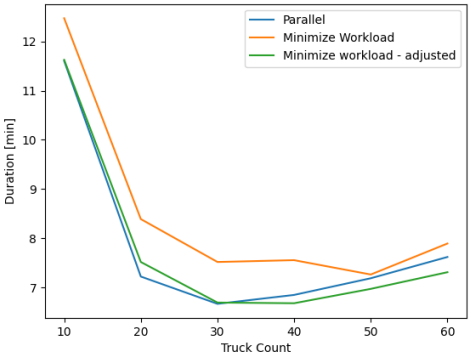
(b) Warehouses with 10 paths with and without strong order enforcement

Figure 4.5 Comparison of the First bigger policy and the Smallest bigger policy mean solution runtimes

definition) can be several times (based on the number of cranes available in the warehouse) as important as the present workload, despite the present workload being certain and the future workload being only likely. To account for this, we should divide the future workload by at least twice the crane count, so it is never greater than the present workload. To test this adjustment, we can experimentally compare its simulation runs on six paths with the original Minimize crane workload policy and the Parallel policy (see Figure 4.6b). Clearly, it is faster on all truck counts than the original Minimize crane workload policy, and for 30 and more trucks it is even faster than the Parallel policy. Therefore, the adjustment has likely helped.



(a) Average solution duration across all path counts



(b) Comparison after adjusting the future workload weight in warehouses with six paths

Figure 4.6 Comparison of the Parallel policy and the Minimize crane workload policy mean solution runtimes

4.2.3 Additional strategy performance comparison

In this section, we will analyze how the strategies of **smart reversing** and **assigning at the cranes** (see Chapter 3 for their definitions) impact the warehouse efficiency. Our hypothesis will be that both of them will improve the overall solution quality individually, and even more when combined. For the random policy, reassigning every container after arriving at a crane should not make a difference, but for other policies, it will likely help them be more robust against any noise.

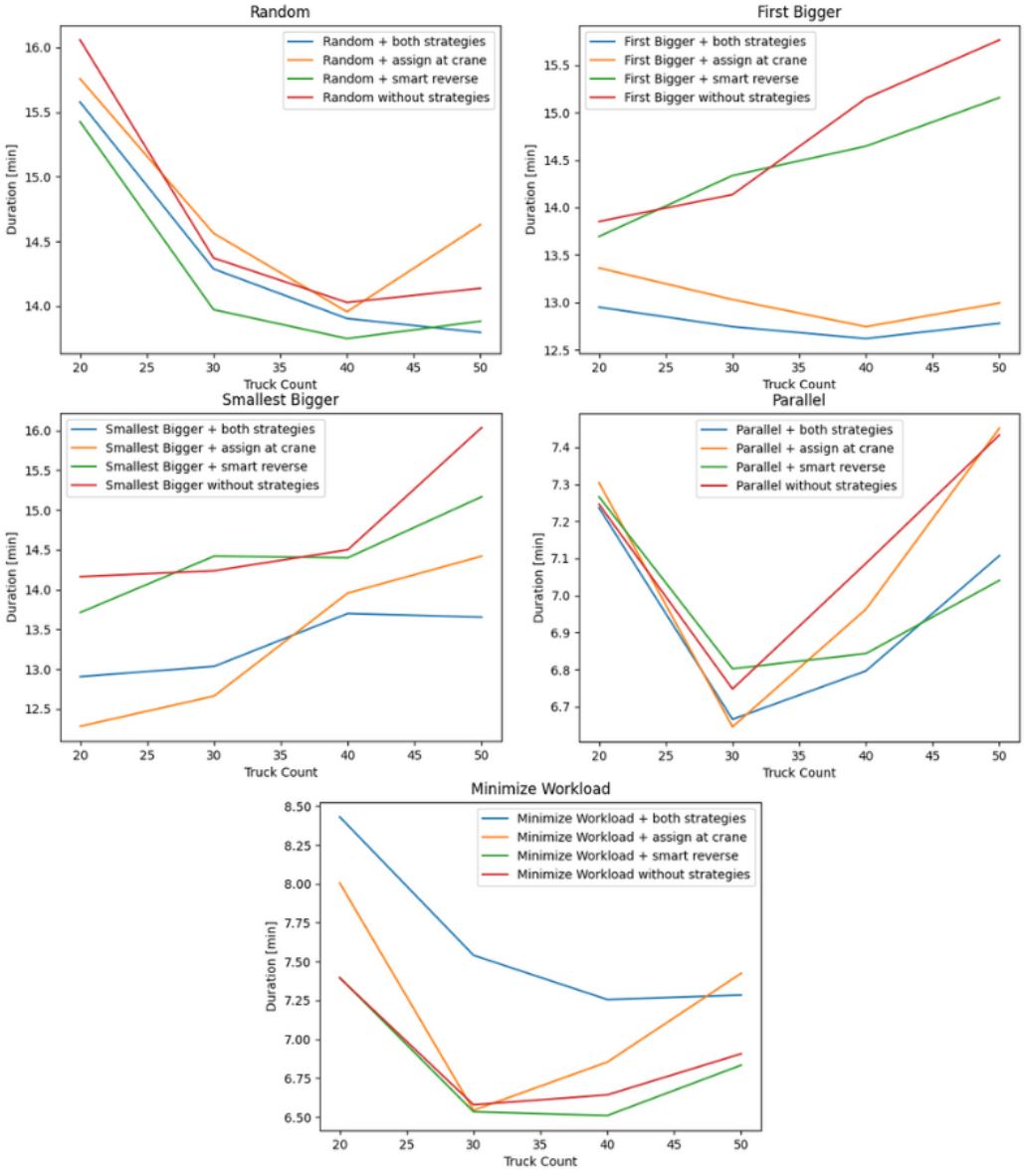


Figure 4.7 Relationship between time and number of trucks with or without the movement strategies for individual policies

To verify this hypothesis, we can observe how the duration of the solution execution has changed for individual policies based on which subset of the two strategies has been used (see Figure 4.7). For every truck count, a problem instance with warehouse dimensions same as in Section 4.2.1, but only on ten and six paths, has been run. To truly decide whether the strategies increase warehouse

efficiency, a more thorough experimentation is required, but this gives us at least an indication about what their effect is.

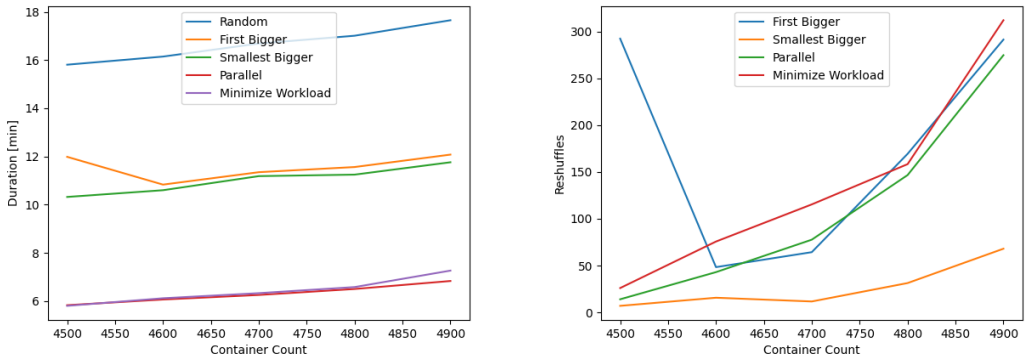
For the Random policy, they behave as expected. Assigning at crane requires an extra few operations for every container, so it slows down the simulation a little and should not be used (and will not be used in the following experiments), but the smart reverse strategy shows a clear improvement over using no additional strategy. For the First bigger and the Smallest bigger policy, smart reversing helps slightly, while assigning at crane makes a significant impact and the solutions are faster by about ten percent. Together, these two strategies work even better. For the Parallel policy, the results change only slightly, but it still seems like the strategies help.

Interestingly, the Minimize crane workload policy is the only one in which the use of both additional strategies makes the warehouse *less* efficient. While using only the smart reversing actually improves its efficiency slightly, assigning at crane probably suffers from the same problem as when using the Random policy - it does not improve the container distribution among the stacks sufficiently to make up for requiring an additional computational power and slowing down the solution. This can be seen when we look at the reshuffle counts. The Smallest bigger policy and the First bigger policy require significantly more reshuffles than the Parallel policy and the Minimize crane workload policy. Assigning at crane is supposed to make the policies more robust against changes in container distribution. That is because when an imported container is assigned to a stack at the entrance gate, by the time it gets to the crane, its stacks can be organized differently, so the optimal stack might be a different one. However, for the latter pair of policies (mainly the Minimize workload policy), the workload is spread more evenly among the cranes, so such changes occur less often and are less significant. This in turn makes assigning at crane less important.

4.2.4 Policy behavior in warehouses with limited space

To see how individual policies behave when the warehouse approaches its maximal capacity, we will look at simulation runs of individual policies in a warehouse with 10 cranes, each operating on 5 rows of 10 stacks able to store 10 containers. From the container retrieval condition (see Section 3.2.1) it follows that the warehouse can safely store at most $(10 \cdot 5 \cdot 10 - 1) \cdot 10 = 4900$ containers, despite having a theoretical capacity of 5 000. The hypothesis we are going to verify is that as the number of containers approaches this limit, the number of reshuffles will increase significantly along with the duration of solution runs. To verify it, we will run experiments for one to five empty stacks in every crane (corresponding to leaving 100 to 500 empty slots for containers) and run every policy on two, six and ten paths with 30 trucks (see Figure 4.8a for simulation durations and Figure 4.8b for the numbers of reshuffles except for the Random policy). Furthermore, for every experiment, we will first import every container and start exporting them only afterward. This will ensure that for every warehouse inspected, the number of containers stored at the same time will meet the desired number.

We can see that, despite some fluctuations (which are likely the result of noise), the duration of solution runs grows approximately linearly with the number



(a) Relationship between number of containers and solution duration

(b) Relationship between number of containers and number of reshuffles

Figure 4.8 Policy behavior in warehouses with limited space

of containers. This is something we would expect to happen simply by adding more containers. However, the number of reshuffles begins to grow approximately exponentially for each deterministic policy as the number of containers approaches the warehouse capacity. The First bigger policy shows a surprising decline in the number of reshuffles between 4 500 and 4 600 containers, but this is likely just noise, caused by the randomized generation of container sequences.

Another interesting observation is that even without the strong order enforcement, the Smallest bigger policy performs better than the First bigger policy for every number of containers. This is likely due to the First bigger policy behaving as the poorly performing First free policy if it cannot find stacks with a lower priority container on top (which occurs more often in warehouses with limited space).

4.2.5 The worst sequence

So far we have compared the policies using mainly their average solution runtime durations on random container and import/export sequences. Another popular metric for comparing algorithms is how they perform in the worst case. The hardest combination of sequences for storing is a monotone container sequence of IDs from one to the number of containers combined with an import/export sequence of all the imports followed by all the exports. This is because we have to store the higher priority containers before the lower priority containers, and the stacks operate in a Last In - First Out manner.

After solving this specific problem instance for 4900 containers, 30 trucks, six paths and the same warehouse dimensions as in Section 4.2.4, so the space is limited, we can see (see Table 4.1) that every policy performs worse than on average. In particular, the First bigger and the Smallest bigger policies perform significantly worse. The Random policy has the most reshuffles, but is still surprisingly faster than the First bigger policy and even a little faster than the Smallest bigger policy. The two fastest policies, the Parallel policy and the Minimize crane workload policies, differ only a little from their average runtime. Interestingly, the First bigger policy has only around half the reshuffles of the Random policy, but it is still the slowest. The main reason is therefore probably the highest average

percentage of time frames spent waiting by the cranes, which indicates a very low parallelization. In other words, the trucks line up in front of one crane instead of being distributed among them. This low parallelization is very likely caused by the First bigger policy reverting to the First free policy after it puts one container into every stack since there is never a stack with a lower priority container on top. As we can see, it would perform better if it behaved randomly.

Policy	Duration Minutes	Reshuffles	Crane Distance	Crane Waiting
Random	18.25	14462	1149444.56	0.85
First bigger	34.89	6756	654998.36	0.95
Smallest bigger	20.74	4753	614426.04	0.93
Parallel	6.68	4406	611852.71	0.76
Minimize crane workload	6.92	4410	611897.89	0.77

Table 4.1 Comparison of selected statistics for the worst sequence in a warehouse

Conclusion

In this thesis, we have analyzed the planning of actions for an automated container warehouse. It makes three main contributions.

First, an approach for collision and deadlock avoidance of the container trucks have been developed. It can be combined even with a very simple incomplete path-finding algorithm and provide the necessary guarantees to make it applicable in reality.

Second, we have focused mostly on the design of container stacking policies, of which six have been proposed and thoroughly compared in multiple experiments. Some experiments have also inspired slight improvements in the policy implementations, which is something that could be built upon in the future works.

Third, an environment for visualizing warehouse behavior and experimentally verifying hypotheses has been developed in Unity 3D.

To achieve maximum warehouse efficiency, further research needs to be carried out. For example, a weighted policy could be developed in which multiple criteria, similar to those considered in the Minimize crane workload policy, are assigned their importance coefficients. The coefficients can be dynamic and adapt to a warehouse by using reinforcement learning or they could be chosen in advance based on thorough experimentation. Similarly, the collision and deadlock avoidance approach could likely increase its efficiency by considering the priority of carried containers.

Bibliography

1. Salido, Miguel A.; Sapena, Oscar; Rodriguez, Mario; Barber, Federico. A Planning Tool for Minimizing Reshuffles in Container Terminals. In: *2009 21st IEEE International Conference on Tools with Artificial Intelligence*. 2009, pp. 567–571. Available from doi: 10.1109/ICTAI.2009.53.
2. Park, Taejin; Choe, Ri; Kim, Young Hun; Ryu, Kwang Ryel. Dynamic adjustment of container stacking policy in an automated container terminal. *International Journal of Production Economics*. 2011, vol. 133, no. 1, pp. 385–392.
3. Dekker, Rommert; Voogd, Patrick; Van Asperen, Eelco. Advanced methods for container stacking. *Container Terminals and Cargo Systems: Design, Operations Management, and Logistics Control Issues*. 2007, pp. 131–154.
4. He, Junliang; Xiao, Xin; Yu, Hang; Zhang, Zhuo. Dynamic yard allocation for automated container terminal. *Annals of Operations Research*. 2022, pp. 1–22.
5. Stern, Roni; Sturtevant, Nathan; Felner, Ariel; Koenig, Sven; Ma, Hang; Walker, Thayne; Li, Jiaoyang; Atzmon, Dor; Cohen, Liron; Kumar, TK, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In: *Proceedings of the International Symposium on Combinatorial Search*. 2019, vol. 10, pp. 151–158. No. 1.
6. Li, Jiaoyang; Surynek, Pavel; Felner, Ariel; Ma, Hang; Kumar, TK Satish; Koenig, Sven. Multi-agent path finding for large agents. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2019, vol. 33, pp. 7627–7634. No. 01.
7. Yakovlev, Konstantin; Andreychuk, Anton. Any-angle pathfinding for multiple agents based on SIPP algorithm. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. 2017, vol. 27, pp. 586–594.
8. Ma, Hang; Li, Jiaoyang; Kumar, TK; Koenig, Sven. Lifelong multi-agent path finding for online pickup and delivery tasks. *arXiv preprint arXiv:1705.10868*. 2017.
9. Zhong, Meisu; Yang, Yongsheng; Dessouky, Yasser; Postolache, Octavian. Multi-AGV scheduling for conflict-free path planning in automated container terminals. *Computers Industrial Engineering*. 2020, vol. 142, p. 106371. issn 0360-8352. Available from doi: https://doi.org/10.1016/j.cie.2020.106371.
10. Švancara, Jiří; Vlk, Marek; Stern, Roni; Atzmon, Dor; Barták, Roman. Online multi-agent pathfinding. In: *Proceedings of the AAAI conference on artificial intelligence*. 2019, vol. 33, pp. 7732–7739. No. 01.

11. ČÁP, Michal; NOVÁK, Peter; KLEINER, Alexander; SELECKÝ, Martin. Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots. *IEEE Transactions on Automation Science and Engineering*. 2015, vol. 12, no. 3, pp. 835–849. Available from DOI: 10.1109/TASE.2015.2445780.

List of Figures

A Attachments

A.1 Simulation development

A simulation developed in Unity 3D 2022.3.27f1, visualizing the described method, the warehouse, and offering important functionality is attached to this thesis. In this section, the simulation is described in more detail from the development perspective. Project files can be found in the *My project* folder of the attached *Container stacking.zip* file.

A.1.1 Simulation objectives

The simulation is designed with the intention of allowing researchers in container warehouse planning to carry out experiments and verify hypotheses. It features an interface that allows for running series of predefined experiments quickly or for more detailed analysis of the individual scenarios.

A.1.2 Simulation architecture

The whole simulation uses only one scene, in which there are multiple objects that interact and are controlled by scripts.

Objects

The main objects present in the scene are all the objects described in the Environment 1.1 section. Furthermore, there are virtual objects representing individual rows, stacks, and paths, allowing for better position referencing in scripts. The user interface consists of several objects, including buttons, sliders, and backgrounds. There is also a spectator object that the user can control and view the simulation through. All objects are spawned at the beginning of a simulation run and destroyed afterwards. The only exceptions are floors and gates, which have only their scales adjusted, as well as containers which are spawned upon entering and destroyed upon exiting the warehouse.

Attached scripts

Most of the objects have an attached controller script. The container yard floor has a *WarehouseController* attached instead, as well as the *WarehouseFileManager*, *WarehouseUI*, *ExperimentManager*, *TruckManager* and *PathManager*, making it the most important object of all. The entrance gate has *SpawnerTrucks* and *SpawnerContainers* scripts attached.

The controllers hold most of the important fields and are responsible for most of the logic related to their objects.

The *TruckController* and *CraneCableController* are the most complex controllers, as they control the state machines responsible for the movement of the objects they're attached to, the containers they are transporting, and they are communicating with other controllers a lot.

The *TruckManager* and *PathManager* script are responsible for all the corresponding *Controller* scripts.

The *WarehouseController* script serves as the main script. It holds most of the global variables and holds (at least transitive) references to every other script, allowing it to access the remaining public variables. Therefore, most of the scripts hold a reference to it. It also controls the initialization and destruction of the other objects.

There are three additional scripts, not attached to any object. The stacking policies are defined in a separate *StackingPolicies* script. *Helper* serves for comparing two floating point numbers with a precision that works well enough, as the inbuilt >,<,= are too strict. *ListExtensions* provides a simple way to generate a random permutation of a sequence, which is useful for generating random container sequences.

A.2 Simulation usage

The following section provides the user with an overview of the available functionalities of the simulation. The main application can be found in the *Build* folder of the attached *Container stacking.zip* file. The target platform is Windows, but it should also work on macOS and Linux.

When the app is launched, a start menu screen appears (Figure A.1).

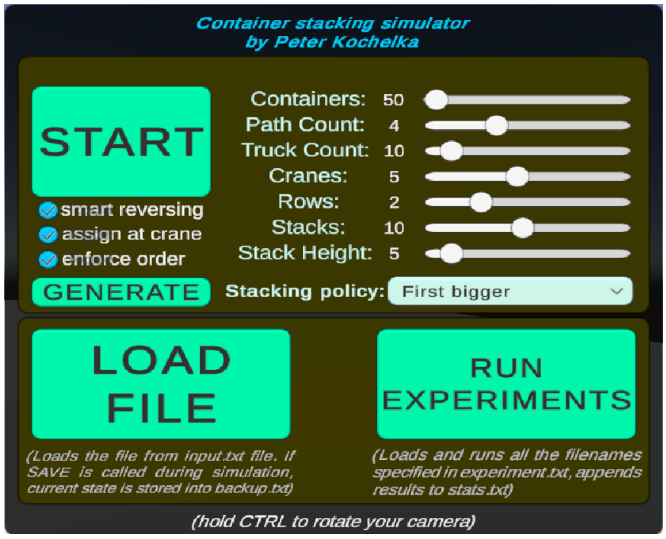


Figure A.1 Start Menu

Now, the user has several options. They can specify the warehouse parameters with the provided sliders, toggle the strategies of smart reverse (see 3.1.2), assignment at crane (see 3.2), or strong order enforcement (see 4.2.2), and choose from a selection of stacking policies (for their description, see 3.3). Afterwards, they can press one of the available buttons:

- **START**: start a simulation based on the specified parameters with a randomly generated container and decision sequence
- **LOAD FILE**: read the file *input.txt* from the application folder, with everything specified and start the simulation

- **GENERATE**: read the file *experiment.txt* from the application folder, where the names of experimental files are stored. Then randomly generate their contents based on the specified parameters
- **RUN EXPERIMENTS**: read the file *experiment.txt* from the application folder, where the names of files with experimental configuration are stored. Then load and run them sequentially.

A.2.1 Parameter setting

The user can set the following parameters with the sliders. Every value is a natural number from the specified interval.

- **Containers**: how many containers will be imported and then exported during the simulation [1-5000]
- **Path Count**: how many paths there will be available for the trucks [1-10]
- **Truck Count**: the maximal number of trucks that will be allowed to operate at once [1-100]
- **Cranes**: how many cranes will be operating [1-10]
- **Rows**: how many rows will each crane operate [1-5]
- **Stacks**: how many stacks there will be in each row [1-20]
- **Stack Height**: how many containers a stack will be able to hold [4-15]

The trucks are the most computationally difficult agents, so setting their number too high may lead to performance issues and longer time frames. The maximum number of containers that a warehouse can store at once, while still being able to certainly retrieve all of them, is

$$\mathbf{Cranes} \cdot (\mathbf{Rows} \cdot \mathbf{Stacks} - 1) \cdot \mathbf{StackHeight}$$

as every stacking policy provided ensures that the condition for container retrieval (see 3.2.1) holds as long as possible. It is therefore not recommended to set the number of containers higher than that, although in certain cases it might work.

A.2.2 Loading a file

All the application text files are stored in a platform-specific user directory, that is persistent across sessions:

```
Windows:
C:\Users\YourName\AppData\LocalLow\Peter Kochelka\Container Stacking\

macOS:
/Users/YourName/Library/Application Support/Peter Kochelka/Container Stacking/

Linux:
/home/YourName/.config/unity3d/Peter Kochelka/Container Stacking/
```

When loading from, or storing into, a file, it has to follow a strict format, which provides all the specified parameters, the whole sequence of not yet imported containers, the whole schedule of import/ export trucks, and also the current partial filling of the warehouse (which containers are stored in which stacks).

The values of sliders and toggles along with the current warehouse state are stored into separate lines as follows:

- *Containers,Path Count,Truck Count,Cranes,Rows,Stacks,Stack Height,Order number of the stacking policy*
- *Smart Reverse,Assign At Crane,Strong Order Enforcement* (0 for `FALSE`, 1 for `TRUE`)
- Container IDs separated by `","`
- Truck schedule in the form `"iiieeeieieie..."` where `"i"` stands for import and `"e"` for export
- For each row from east to west, its stacks from south to north along with their containers are listed, separated by `";"`. For each stack, its containers are listed from bottom to top separated by `","`. The rows are then separated by `"\t"`

The user has the full responsibility for providing valid files. They have complete control over specifying the parameters. However, there is no guarantee that the simulation will work if they set these parameters to values outside the slider ranges.

A.2.3 Simulation run

When running the simulation, the user can inspect the warehouse in detail. They can move around with the keyboard arrows and when holding *CTRL*, they can rotate their camera using the mouse.

When the mouse cursor is pointing at a container, its ID is shown. By clicking on a container, every container not inserted into its stack becomes transparent, which allows for more focused inspection. To make them all appear again, the user should click on any container.

During the run, some basic data are shown in the bottom left corner of the screen. Specifically, reshuffle count, elapsed time (not accounting for the simulation speed), number of already imported containers, and number of containers left to be exported. Furthermore, the whole control menu is available.

Control menu

The control menu contains **QUIT**, **RUN**, **STOP**, **NEXT**, and **SAVE** buttons, as well as a slider affecting the simulation speed (decimal values from 0.2 to 100). The **RUN** button starts (or resumes) the simulation run at the set speed. The **STOP** button stops it. The **NEXT** button can be pressed only when the simulation is stopped. It runs the simulation, but lets only the next truck waiting on each entry pathway into the warehouse and stops the simulation once all trucks

exit and all the containers in the warehouse are stacked. The **SAVE** button can be pressed only when all containers in the warehouse are stacked and no truck is in the warehouse. The current configuration is then saved in the *backup.txt* file in the application folder, in the same format as specified in A.2.2. The **QUIT** button stops the simulation and asks the user if they want to save the current state first or quit instantly. In case of saving, the trucks in the warehouse have to exit first and the cranes have to stop. The speed slider can be used only when the simulation is paused.

A.2.4 Experimentation

When running experiments, the speed of the simulation is automatically set to 100. In addition, the Control menu is reduced only to the **QUIT** button, which does not allow for saving. The user has to provide the experimentation files themselves into the persistent folder and specify their names in the "experiment.txt" file (see section on Loading from a file).

After every completed simulation, some of its statistics are appended to the end of *stats.txt* file in the persistent folder. They are stored as a single line, in the following format:

Current date and time; *Containers*; *Path Count*; *Truck Count*; *Cranes*; *Rows*; *Stacks*; *Stack Height*; *Policy*; *Smart Reverse*; *Assign at Crane*; *Enforce Order*; *simulation duration*; *reshuffle count*; *distance covered by trucks*; *distance covered by cranes*; *distance covered by crane cables*; *proportion of time frames spent waiting by the average truck*; *proportion of waiting time frames by the average crane*; *proportion of waiting time frames by the average crane cable*

A.2.5 Overview of the attached files

Important attached files can be found in the Experimentation folder, where *stats.txt, worst runs.txt*, and *special runs.txt* contain, respectively, the results of most experiments carried out during the writing of the thesis, the results of experiments with the worst container sequence (see section 4.2.5), and the results of experiments in warehouses with limited space (see section 4.2.4). *Experiment evaluation.ipynb* contains the code used to obtain the graphs used in this thesis and the *Images* folder contains its output.