Optimization of Pick Order Assignment

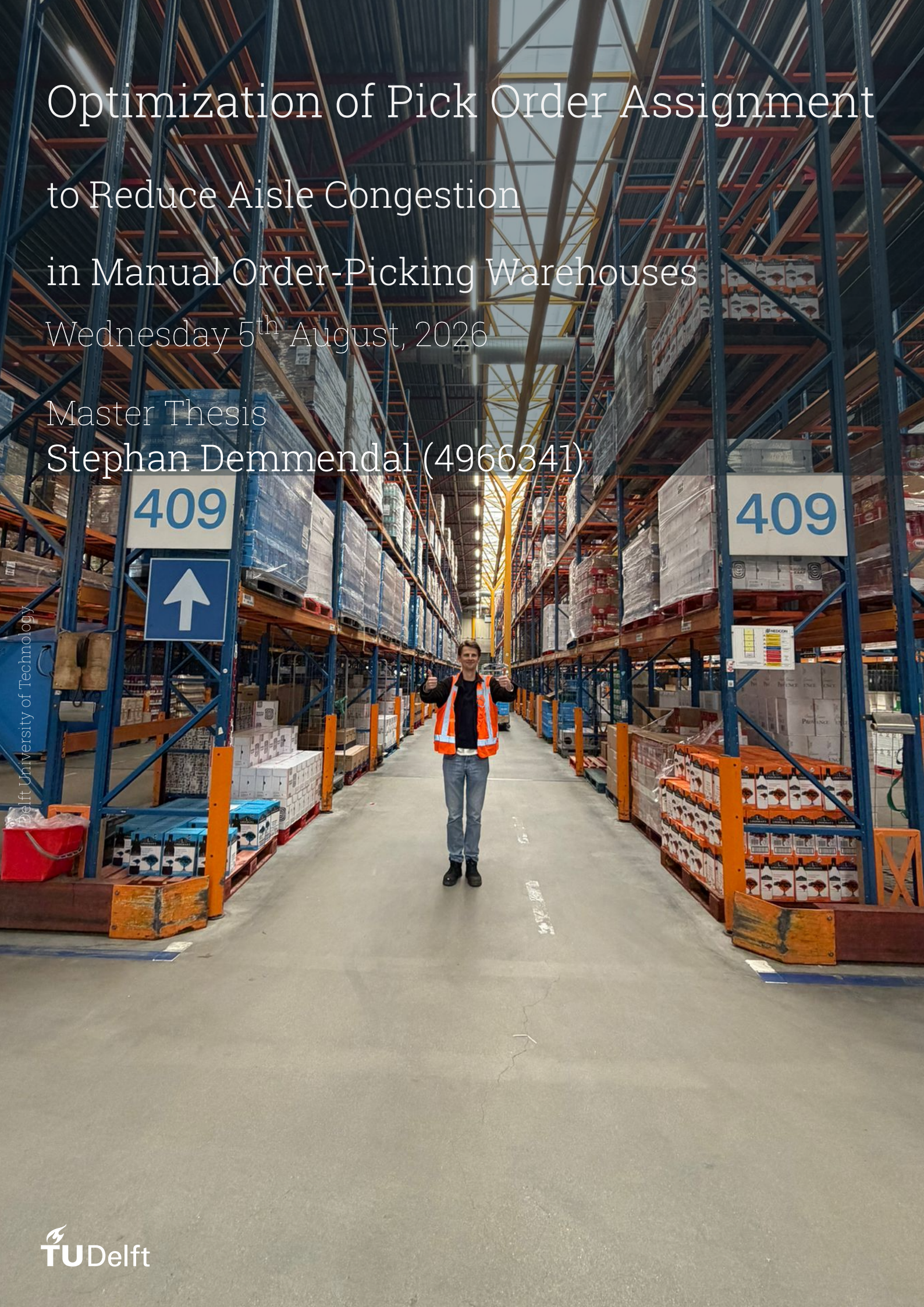
to Reduce Aisle Congestion

in Manual Order-Picking Warehouses

Wednesday 5th August, 2026

Master Thesis

Stephan Demmendal (4966341)



Delft University of Technology

TU Delft

Optimization of Pick Order Assignment

to Reduce Aisle Congestion in Manual Order-Picking Warehouses

Wednesday 5th August, 2026

By

Stephan Demmendal (4966341)

Stephan Demmendal Applied Mathematics 4966341

TU Delft supervisors:	Teun Janssen & Yuki Murakami
Company supervisors:	Willemijn Dietz & Tom Jongen
Responsible supervisor:	Jos Weber
Project Duration:	February 2026 - August 2026
Word Count:	22723

TU Delft

Preface

Stephan Demmendal (4966341)
Delft, August 2026

By writing this preface, this will be one of my last steps towards finishing my studies. I have done both my bachelor's and master's in Applied Mathematics at the TU Delft. When talking about my studies the applied part is something that has always been up for debate. During my masters I started with almost all courses to see what courses were the most applied, and supposedly contained the fewest number of mathematical proofs. This is where I chose to pursue the discrete mathematics and optimization track. I think I peaked in abstract theoretical mathematics during my courses in both Discrete and Semidefinite Optimization, proving things I did not know could be proven. During the first year of my master I made new friends whom without I could not have done it, Ana, Ida and Max thank you!

At last I had to do this master thesis, which I was able to do at a company. This was the first time where I got to apply what I had learned to real world problems. I have enjoyed this time a lot, where I never expected to have so much fun doing so. The project itself really sparked my enthusiasm, however the people involved is what really made this a fun journey. I want to thank my supervisors for helping me get this thesis to what it has become and making my time so enjoyable. My TU Delft supervisors Yuki and Teun with whom I have sparred about all kinds of ideas regarding the topic and helped me a lot in getting to this result. My company supervisors Willemijn and Tom who helped me on the company side of this thesis, providing me guidance when and where needed.

I have enjoyed my time in both Delft and Rotterdam very much, where I could not have done it without all the people who supported me. Also thank you to my friends and housemates who supported me during the long evenings spent on my studies. A big big thank you to the fam who have given me endless mental and financial support throughout my studies.

Finally, Sterre thank you for being my greatest supporter and biggest help for the past 8 years. You have always been there for me during my studies, helping me take it one step at a time, always there to listen to what I had to ramble on about during long evenings and weekends.

AI declaration

I hereby declare that I have used generative AI as a tool to improve the clarity, structure, and quality of my own writing. Generative AI was not used to create any text or content on my behalf.

Executive Summary

Problem definition and Research question

Warehouses rely on order pickers moving through aisles to collect items for shop orders. Large retailers already use algorithms to decide how items are grouped onto load carriers (a process referred to as batching) and which route a picker takes, in order to minimize travel time. These algorithms do not account for what happens when many pickers execute their routes at the same time, however: workers can end up needing the same aisle at the same moment, forcing them to wait for one another. This congestion adds time on top of what was planned, and becomes more pronounced during busy periods.

This thesis treats batching and routing as fixed, since they are already handled by existing systems, and instead asks whether congestion can be reduced through a different lever: when, and to whom, pick orders are assigned. The resulting research question is: how can order-to-picker assignment be optimized to minimize peak aisle congestion in a warehouse?

Methodology

We formalize the problem as a scheduling problem, with pickers as parallel machines, pick orders are referred to as batches, where they are treated as jobs consisting of a fixed sequence of aisle visits. We prove a simplified version, the Aisle Load Balancing Problem (ALBP), to be NP-complete through a reduction from graph edge colouring. An extended version builds on this with realistic constraints: batches visit aisles in sequence with individual processing times, and must start within an earliest and latest allowed time drawn from real delivery deadlines.

Since solving this problem's ILP exactly can take from seconds to hours, we developed two greedy heuristics as faster alternatives. The NeighborhoodN Greedy builds a schedule while actively avoiding congestion increases, using a limited look-ahead among already available batches. MaxM Greedy instead takes a maximum aisle load as a fixed input and schedules as many batches as possible within it.

We tested all methods, plus a benchmark reflecting current practice, on real data from two large distribution centres, across eight representative days each, in the pickzone with the highest realized number of pick orders.

Results

Across all sixteen instances tested, the ILP achieved the lowest maximum aisle load ($M = 2, 3, 4$), followed closely by MaxM Greedy (M between 3 and 6). Both stay well below the operational congestion threshold of six pickers per aisle. Current assignment practice, and its heuristic approximation of real-world practice (NeighborhoodN Greedy ($N = 0$), reaches an average close to $M \approx 9$ to 10, regularly exceeding this threshold.

There are several important caveats. The ILP achieves its result by using the full width of each batch's allowed time window, which lengthens the schedule, though it never finishes a batch late. MaxM Greedy trades off differently: pushing its bound too low causes batches to finish too late instead. The ILP is also expensive and unpredictable to compute, ranging from 21 seconds to over two hours with no clear relation to instance size, while both heuristics run in under one second on every instance. This pattern, the ILP strongest but slowest, MaxM Greedy fast and close behind, current practice weakest, holds consistently across all eight days and both warehouses.

Conclusions

Aisle congestion can be reduced substantially through scheduling alone, without changing how orders are batched or routed, and this holds regardless of warehouse or workload. Achieving the largest reduction exactly, with the ILP, requires accepting a longer schedule or an unpredictable amount of

computation time, limiting it to a benchmark rather than a daily tool. Achieving it approximately, with MaxM Greedy, gives up only a small amount of congestion reduction for a schedule produced in under a second and tunable to whatever aisle load a warehouse accepts. Of the methods tested, MaxM Greedy is the most realistic candidate for day-to-day use.

Further work

Several directions remain open. The model's inputs could be made more realistic, through finer norm time estimation, picker breaks, a safety buffer against real-world delays such as forklift resupply operations. Two assumptions set in this thesis could be relaxed: a constant number of pickers throughout the day, and batching and routing that ignore congestion. The heuristics could also be extended, for instance, by allowing a previously raised aisle load bound to decrease again as time advances. Finally, alternative objectives, such as minimizing makespan under a fixed congestion limit, or a genuine bi-objective formulation producing a full tradeoff curve, would let a warehouse choose its own balance rather than the discrete points explored here.

Contents

List of Figures

List of Tables

Nomenclature

General Mathematical Notation

Table 1: General mathematical notation and set definitions

Symbol	Meaning
$[A]$	$\{1, \ldots, A\}$, a set of elements, where A is a positive number.
$\left\{a_1^{m(a_1)} \ldots a_n^{m(a_n)}\right\}$	A multiset with unique elements $a_1, ..., a_n$, where each element a_i appears $m(a_i) > 0$ times.
$m(\mathcal{M})$	Function returning the maximum multiplicity of an element in multiset M.
$\biguplus$	Multiset sum operator; $M_1 \uplus \cdots \uplus M_k$ is a multiset where the multiplicity of each element is the sum of its multiplicities across all M_i.
$G = (V, E)$	A graph with vertex set V and edge set E.
Simple graph	An unweighted, undirected graph with no loops or multiple edges (Tutte, 1998).
Adjacent edge	Two edges sharing a common vertex.
$c : E \to [k]$	An edge-colouring function assigning one of k colours to each edge.
$\chi'(G)$	Edge chromatic number: the minimum number of colours needed to edge-colour G such that no two adjacent edges share a colour.

Table 2: List of abbreviations

Abbreviation	Full term	Meaning
ALBP	Aisle Load Balancing Problem	The problem of minimizing the maximum multiplicity of the aisles that appear in the routes assigned to the order-picker.
EST/LST	Earliest/Latest Start time	The timestamps that mark the beginning and the end of the feasible time window in which a pick order should start.
ILP	Integer Linear Program	A linear optimization problem in which all decision variables are constrained to take integer values.
NP	Nondeterministic Polynomial time	The class of NP consists of problems of which a proposed solution can be verified within polynomial time and can be solved in nondeterministic polynomial time.
RCPSP	Resource Constrained Project Scheduling Problem	A scheduling problem in which project activities require renewable resources subject to availability limits.
TSP	Traveling Salesman Problem	The problem of finding the shortest route that visits a given set of locations exactly once and returns to the starting point.

Glossary of Warehouse Terms

Table 3: Glossary of warehouse terms

Term	Meaning
Shop order	A list of items requested by an individual shop; the top-level customer order before splitting the order up.
Load carrier	A physical cart onto which items from one shop order are stacked. One load carrier never contains items from multiple shops.
Batch (Pick order)	A group of load carriers, possibly from different shop orders, assigned to and picked by one picker in one route.
Picker	A warehouse worker who retrieves items from slots and places them on load carriers; modelled as a machine in the scheduling formulation.
Slot	A specific storage location for a product within an aisle.
Aisle	A parallel, one-directional lane of the warehouse containing slots, visited in a fixed "S-shape" order.
Pickzone	A section of the warehouse (e.g. non-perishables, 2°C, 14°C) within which pick orders are contained; the ALBP is solved independently per pickzone.
Norm time	The estimated standard time an action, such as moving or picking products from the aisles, should take; used as the model's processing time input, distinct from realized time.
Aisle Load / Aisle Multiplicity	The number of pickers simultaneously active in one aisle at one point in time; M measures the maximum of this over time.
Congestion	The umbrella term for delays, for example caused by pickers obstructing one another.

Named Algorithms and Subroutines

Table 4: Named methods and subroutines

Term	Meaning
NeighborhoodN Greedy	The heuristic with objective of scheduling while minimizing M through limited look-ahead.
MaxM Greedy	The heuristic that takes a fixed maximum aisle load M as input and schedules as many batches as possible without exceeding it.
ExceedM	Subroutine algorithm that checks whether scheduling a batch would exceed the current aisle load M, without committing the change.
UpdateAisleLoad	Subroutine that commits a batch's aisle occupancy to the Aisle Load matrix.

1 Introduction

Warehouses play a central role in modern retail operations. They form the operational backbone that connects suppliers to stores, ensuring that products are available, deliveries are on time, and service levels are met. As retail environments become increasingly demand-driven and time-sensitive, distribution centres are expected to operate at high throughput while keeping costs low and reliability high. Operational inefficiencies within these warehouses therefore translate directly into higher costs and lower service levels.

Within warehouse operations, order picking is consistently the most labour-intensive and costly process, often accounting for more than half of total operating expenses [24]. Much of this cost comes from travel time rather than from the physical act of picking itself, which is why large retailers, including the one studied in this thesis, already rely on algorithms to decide how orders are grouped into pick orders and how pickers route through the warehouse.

Minimizing travel distance, however, does not automatically minimize how long a pick order actually takes to complete. In large warehouses with many pickers working at the same time, pickers can end up needing the same aisle at the same moment. Especially in narrow aisles where passing is difficult, this forces pickers to wait for one another, adding time on top of what the batching and routing algorithms planned for. We refer to this broad phenomenon as congestion, a situation in which picking activities obstruct one another and cause delays. This thesis aims to reduce this congestion. Before we dive into the how, we explain the processes currently seen in warehouses.

1.1. Introduction of the Warehouse and its Processes

The warehouse consists of different pickzones that contain products that need to be stored in different temperatures. The locations of products in the aisles of the warehouse are referred to as slots. The aisles of the warehouse are parallel and one directional. The aisles are visited in an S-shape, which makes it so that the aisles are always visited in the same "ascending" order. Within the warehouses, we consider the processes for handling shop orders, described further in Chapter 2. These processes are handled and steered mostly by existing algorithms. An overview is given in Figure 1.1, sorted in 5 steps. Each step is described below: (1) receiving shop orders, (2) grouping shop orders into load carriers, (3) building pick orders, (4) assigning pick orders to pickers, and (5) collecting the items.

Step 1: Shop orders
The first step is to handle the arriving shop orders from each shop. In essence a shop order is a list of items. These are items that need to be delivered to the shops so that they can restock their shelves within the shop.

Step 2: Load carriers translation
The lists of multiple shop orders are put into an algorithm that decides which items are put onto which load carrier. A load carrier is a cart on which the items can be stacked and later put into a truck to be delivered to the shops. One load carrier contains the items of one unique shop. This "Load Carrier Algorithm" is computationally the most expensive, which considers multiple factors such as shopfriendliness (The carts are packed so that they are easy to unpack; the items on the cart are located in the same area within the supermarket), stackability (some products nicely stack onto other products, while others do not), total weight of the stacked cart and several other relevant factors.

The main objective is to minimize the number of load carriers needed and also the time spent to load and unload these carriers. The output is then shown as the result of step 2 where we have a large set of load carriers with the items from multiple shop orders. The colour of the load carriers in Figure 1.1 represents the shop to which the items on the load carrier belong to. It must be noted that one load carrier cannot have items from multiple shops.

Step 3: Building the pick orders
After the orders are translated to multiple load carriers, the next algorithm takes this set of load carriers as its input. The objective is to group together load carriers in such a way that their total retrieval time is minimized. The order picker is assigned 5 carts. These carts can be from different shop orders, as illustrated by the different colours of the carriers in groups of 5. Load carriers with a large number of the same items can be picked faster if grouped together.

The items are strategically located within the aisles of the warehouse. (e.g. heavy items are placed at the start of the aisle and items which can better be placed on top are located further in the aisle.)

Step 4: Assigning the pick orders
When the set of load carriers are grouped together into groups of 5 load carriers, the groups have to be assigned to a single picker. Every individual shop order and therefore every individual load carrier has a time at which it must be placed in the truck to be transported to their respective shop. We refer to this time as: "*Latest transport time*". The pick orders are assigned to pickers in ascending order of latest start time, a quantity derived from the latest transport time, as explained in Chapter 4. This assignment problem, deciding which pick orders are assigned to which picker and at what time, while respecting the latest transport times, is the central problem addressed in this thesis.

Step 5: Collecting the items
Once pick orders are assigned to pickers, they must be retrieved from a set of slots within the warehouse. The picker has to pick up the products in an order also given by the algorithm in Step 3. A route can be factored into two parts: picking items and placing them on the load carrier, and moving to the next slot. As these routes are fixed, each route can be separated into an ordered list of the aisles which need to be visited in a given order.

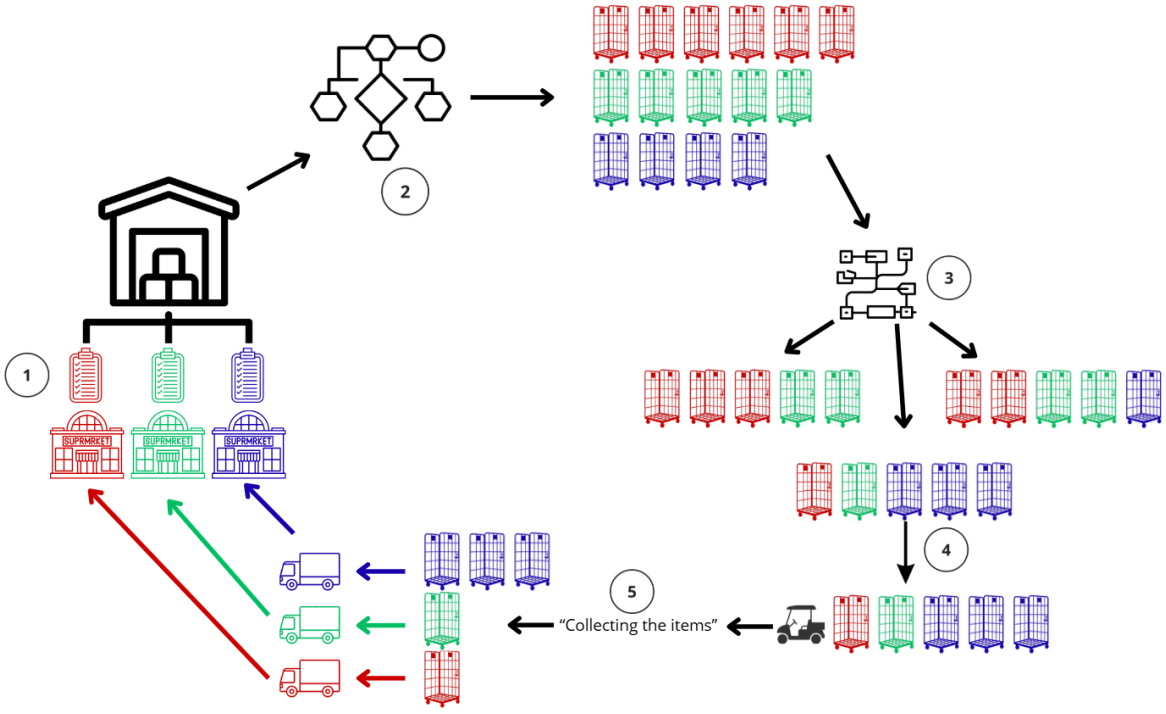


Figure 1.1: Visualization of the warehouse operations divided into 5 steps.
Step 1: Orders of the shops arrive at the warehouse.
Step 2: An algorithm translates the orders to a minimal number of load carriers.
Step 3: An algorithm groups together the load carriers into groups of 5 load carriers, based on item similarity.
Step 4: The load carriers are assigned to a picker in the warehouse.
Step 5: The items are collected and placed on the load carrier, which is then be placed on the outbound lane for transport to the shops.

1.2. Scope

We study a subset of the distribution centres that distribute goods to large regions within the Netherlands, since these are the warehouses where relatively high throughput leads to congestion most often. Central warehouses that store slow-moving products are excluded, since manual observations showed congestion to be far less frequent there. Within the selected warehouses, we further narrow the scope to the pickzones with the highest pick order volumes, where congestion is most likely to occur.

Throughout this thesis, building the load carriers and the routing of pickers through the warehouse are treated as fixed, since these are already determined by existing algorithms, as described in Section 1.1. We do not attempt to change what a picker collects or which route they walk. Instead, we focus solely on the assignment decision: which picker is given which pick order, and when, i.e. Step 4. We also assume a constant number of pickers is available throughout the period we schedule for, an assumption discussed further in Chapter 4.

1.3. Problem Statement and Research Question

The aim of this research is to reduce congestion within the warehouses. Congestion itself is hard to quantify, since it can have a variety of causes. Of these causes, we focus on one that is tangible and measurable: picker density within the aisles. Since congestion has been shown to increase with the number of pickers working simultaneously in a confined area [17], we aim to reduce the number of pickers active at any given time within the same aisle.

Currently, orders are assigned to pickers in ascending order of latest possible start time, mainly to prevent pick orders from finishing too late. In this research, we show that meeting these deadlines and minimizing the number of pickers active in the same aisle can be achieved at the same time. This works because the time window in which a pick order may start is often wide enough to leave a pool of pick orders to choose from when assigning work. As described in step 5 of Section 1.1, a route consists of the aisles a picker must visit, with items to be picked in each aisle. The time needed for these actions is estimated using the sum of norm times of all actions that make up a pick order, for example driving, walking, picking, unpacking and stacking. A route can therefore be seen as an ordered set of aisles, where each aisle has a processing time made up of all actions needed in that aisle.

We want to assign routes to pickers in such a way that we minimize the number of pickers simultaneously present in the same aisle at any given time. This leads to the central research question of this thesis.

Research question: How can order-to-picker assignment be optimized to minimize peak aisle congestion in a warehouse?

To answer this question, the following sub-questions are addressed:

- How can minimizing simultaneous aisle occupancy be formalized as a scheduling problem, and how does its computational complexity behave?
- How can this problem be formulated as an integer linear program that captures realistic operational constraints such as fixed aisle sequences, non-uniform processing times and time windows, and what congestion reduction and computational cost does it achieve on real warehouse data?
- What greedy heuristics can be used at practical computation times, and how do they perform relative to the ILP and to current, congestion-unaware assignment practice?
- What is the tradeoff between minimizing aisle congestion and other scheduling objectives such as makespan and tardiness, and how consistent is this tradeoff across different warehouses and workload levels?

1.4. Contribution

Existing warehouse practice assigns pick orders to pickers purely on the basis of transport deadlines. The batching and routing algorithms that precede this step are optimized for travel time and load carrier efficiency. None of these methods take into account how many pickers end up working in the same aisle at the same time, even though this is a direct cause of congestion. This thesis closes that gap by treating aisle congestion as an explicit part of the assignment decision, rather than as an unintended side effect that is dealt with afterwards.

This leads to four contributions. First, this research formalizes the pick order to picker assignment problem as a scheduling problem, explicitly modelling the number of pickers simultaneously present in each aisle. We refer to this quantity as the aisle multiplicity, denoted M, which serves as the central measure of congestion throughout this thesis. Second, it provides two solution approaches capable of handling realistic warehouse instances: an integer linear program (ILP) that gives optimal assignments for smaller instances, and two greedy heuristics that generate high-quality assignments for larger instances within practical computation times. Third, the research quantifies to what extent minimizing M actually reduces congestion, by comparing the outputs against current, congestion-unaware assignment practice on real warehouse data. Finally, it translates these results into operational insight, identifying the conditions under which assignment decisions can meaningfully mitigate congestion, and the conditions under which congestion is largely driven by workload characteristics, such as order composition or aisle demand concentration, and cannot be resolved through assignment alone.

1.5. Thesis Outline

This thesis is structured as follows. Chapter 2 reviews the literature on warehouse operations, batching, order picking and congestion, and positions the Aisle Load Balancing Problem relative to existing scheduling literature. Chapter 3 formally defines the Aisle Load Balancing Problem, proves that a simplified version is NP-complete, extends the formulation to capture realistic operational constraints, and introduces two greedy heuristics as scalable alternatives to the exact formulation. Chapter 4 describes how the warehouse data used to test these methods is gathered and processed into the model's input parameters. Chapter 5 presents the results of applying the integer linear program and the heuristics to this data, and examines the tradeoff between congestion reduction and other scheduling objectives. Chapter 6 concludes the thesis and discusses recommendations for future work.

2 Literature Review

This chapter reviews the literature relevant to the Aisle Load Balancing Problem (ALBP). Section 2.1.1 places the problem in its warehouse operations context, covering order picking, batching, and assignment. Section 2.1 introduces the ALBP itself. Section 2.3 covers the scheduling literature needed to formalize it. Section 2.4 positions the ALBP within the literature and identifies the gap it addresses.

2.1. Warehouse Processes and Operational Context

Warehouse management covers a broad range of decisions, typically divided into three layers: strategic, tactical, and operational. Strategic decisions concern the long-term design of the warehouse, including its layout, level of mechanization, and information systems. Tactical decisions determine how warehouse resources are organized, such as storage policies, aisle configuration. Operational decisions focus on the day-to-day execution of warehouse activities, including order assignment, routing, picking, sorting, and sequencing. Pardo et al. [28] propose a taxonomy along these lines, which helps structure the many problems that arise in warehouse management. This review focuses on the operational decisions involved in retrieving items from a warehouse and on the tactical choices that directly affect how well these operations perform, with particular attention to how these choices give rise to congestion.

We specifically look at multi-aisle picker-to-parts warehouses, where multiple pickers are active at the same time. In this warehouse type, pickers walk through the aisles themselves to collect items, as opposed to automated systems where products are brought to a fixed picking station. Having several pickers working in the same aisles at once creates a coordination challenge, since the routes and orders assigned to different pickers affect one another. Done poorly, this leads to congestion, slow routes, and inefficient pick orders, all of which drive up cost. This raises the central question of this review: how can items be picked in a way that maximizes profit, or minimizes cost, under these conditions?

Operational activities in an order picking warehouse can be split into five parts: assigning, picking, routing, sorting, and waiting [28]. This review covers assigning, picking, and routing. Routing is discussed through the lens of order batching, since batching decisions are usually evaluated by the resulting route length. Sorting falls outside the scope of this review, and waiting is touched on briefly, where it intersects with assignment.

2.1.1. Order picking and storage policies

Order picking is the process of retrieving the items that make up a shop order. It is the most labor-intensive and costly operation in warehouses, often accounting for more than half of total operating expenses [22]. Its performance depends on warehouse design choices: storage policies, aisle configuration, picker count, and where pickers start and end their routes. Pardo et al. [28] classify these as tactical decisions.

Storage assignment is the first major tactical decision. Strategies such as class-based or order-volume-based storage aim to reduce travel distances by grouping items with similar demand patterns. However, these strategies can increase congestion. When high demand items are stored together, they attract many pickers into the same aisles, which increases blocking and waiting times [30]. Random storage avoids this problem. It is commonly used in warehouses because it is simple to apply, makes efficient use of available space, and spreads activity evenly across picking aisles. Many picker-to-parts studies therefore adopt random storage as the standard assumption when testing routing methods [22].

In sort-while-pick systems, the picker uses a cart with multiple compartments and places each item directly into the order's compartment. This works well for small orders with few, lightweight items. Orders stay separated during picking, without extra sorting steps needed. For larger orders, or when better cart utilization is needed, it becomes less efficient. In that case, a pick-and-sort strategy works better: items for several orders are collected into one container during picking, and sorted later [14].

A related tactical decision is order batching: combining multiple items from shop orders into a single picking tour to exploit spatial proximity and reduce travel distance. Batching is the method used in our case. Because it affects routing patterns and congestion, it is discussed in more detail in Subsection 2.1.2.

2.1.2. Order batching and routing interactions

Order batching combines multiple shop orders into a single picking tour to exploit spatial proximity and reduce travel distance per order. This usually increases the number of aisles a picker visits. More aisle visits raise picker interaction, and in turn, congestion [19].

The outcome of the batching and routing algorithms described above is what we call a *pick order* throughout this thesis. When a pick order is treated as a modelling object, that is, reduced to its startup and wrap-up steps and an ordered set of aisles each with a norm time, without any further operational detail, we instead call it a *batch*. We keep the term pick order for the operational setting and for realized data, where this additional detail still applies.

The batching problem is NP-hard in the strong sense when minimizing total travel time, if pick orders may contain more than two orders. If every pick order is limited to at most two orders, the problem can be solved in polynomial time [15]. Its computational difficulty grows fast as order counts increase. Henn et al. [19] illustrate this: a warehouse with 40 shop orders and 900 storage locations can produce over 350,000 feasible pick orders. This shows how quickly the number of decision variables grows. As a result, exact solution methods are impractical for realistic instances, even when using commercial solvers.

Because of this complexity, the literature commonly relies on heuristic and metaheuristic approaches. Many techniques have been applied: tabu search, genetic algorithms, ant colony optimization, simulated annealing, hill-climbing, iterated local search, variable-neighborhood search, and hybrid metaheuristics. Rather than discussing every individual method, we refer to Cano et al. [6] for a unified review of the major algorithmic families used for batching problems.

Batching is tightly integrated with routing. Most batching models judge the quality of a pick order by computing shortest path distances or travel time, typically using picker routing formulations that mirror the Traveling Salesman Problem (TSP) [7, 21]. This coupling matters: small changes in pick order composition can significantly change route lengths [19]. As a result, combined batching-routing-sequencing approaches are common, where sequencing refers to the order in which pick orders are released to pickers. Examples include tabu-search methods [23], and hybrid evolutionary techniques that combine genetic and ant colony search [9].

Additional batching studies extend the problem to multi-period or multi-warehouse settings, where batching interacts with assignment and routing across longer planning horizons. These extensions reflect a growing need for integrated, time-coordinated decision models [25].

2.1.3. Assignment of pick orders

After pick orders are formed, they must be allocated to pickers and arranged in a sequence. Assignment determines which picker executes which pick order, and in what order. Although assignment is a fundamental operational decision, it has received less attention as a standalone optimization problem than batching or routing [28]. A plausible explanation is that batching and routing have a larger impact on total travel time. The literature on assignment can be roughly divided into three groups:

- studies where assignment is handled implicitly inside integrated batching-routing models,
- studies that address assignment dynamically as the system evolves,
- studies that treat assignment explicitly as a parallel-machine scheduling problem.

Assignment within integrated batching−routing models

In integrated batching-routing models, a single solution encodes three decisions at once: which orders form a pick order, which picker carries out that pick order, and in what order the pick order's stops are visited. A metaheuristic, such as a genetic algorithm or tabu search, searches over these combined solutions and scores each one using one shared objective. This is usually done based on total travel distance or the number of late orders across all pickers. Because picker assignment is only one part of this

combined solution, it is never optimized on its own. It simply comes out however the best-found solution happens to divide the work. Examples of this approach include hybrid heuristics and evolutionary algorithms that jointly optimize batching, sequencing, and routing [9], and cluster-based tabu search in multi-cross-aisle environments [23].

Dynamic assignment
Concentrating work in localized areas can increase blocking and interference between pickers, as shown in studies on pick density and congestion [17, 20]. Dynamic systems handle assignment throughout the day, and must also handle waiting. When orders arrive at irregular intervals, pickers must decide whether to wait for more incoming orders or start the next pick order [28].

Assignment as a parallel machine scheduling problem
A third body of research models assignment explicitly using scheduling frameworks. Here, pickers are viewed as parallel machines, and pick orders as jobs that need to be processed. This perspective raises three key objectives for assignment research:

- balancing of workload across pickers [28],
- minimization of tardiness [18],
- minimization of makespan [3],

Tardiness and makespan are studied in more detail in Section 2.3.1. These models can also include due dates or time-window constraints [7], and can be extended to pickers with different speeds or capacities [8]. Sequencing follows naturally from the assignment decision: once pick orders are allocated to pickers, the order in which they are executed determines completion times and therefore due-date performance. Cao et al. [8] capture this by incorporating a tardiness penalty into the scheduling objective.

Assignment models often struggle with the combinatorial complexity that comes from combining pick order characteristics, picker capacities, and time constraints. Even so, they are crucial for shaping completion times and balancing system load.

2.1.4. Congestion mechanisms in multi-aisle systems

Congestion has been studied in different warehouse environments, where the main source of delay depends on the physical layout and operational decisions. In narrow aisles, the main issue is blocking, since pickers cannot pass each other. In wide aisles, delays happen mostly at pick-faces, when multiple pickers need the same location. A third line of work connects these congestion effects to higher-level decisions such as batching, sequencing, and storage assignment. It shows how planning choices shape where pickers meet, and how much they interfere with each other. Together, these three viewpoints cover the main causes of congestion.

Congestion in narrow aisle systems
In narrow aisles, where passing is impossible, picker interactions strongly affect performance. As more pickers operate in the same area, blocking increases and individual throughput drops. This happens because pickers must wait for others to clear the aisle before moving to their next pick location [17].

Pick density is defined as the number of picks relative to route length. Its effect on congestion is non-linear because increasing pick density has two opposing effects. At low pick densities, pickers rarely stop, so they seldom interfere with one another. As pick density increases, pickers make more stops while still moving through the warehouse, increasing the chance of blocking each other. Congestion therefore peaks at moderate pick densities. At very high pick densities, pickers spend more time picking and less time walking, so they encounter other pickers less often, although blocking at pick locations can still occur [17].

Congestion also depends on the number of pickers working at the same time, and the size of the picking area. Larger areas with more picking points tend to see less blocking, since pickers naturally spread out over longer routes. Smaller areas or higher worker counts have the opposite effect: they raise the chance of pickers meeting in the same one-way aisle [17].

Congestion in wide-aisle systems
Even in wide aisles, where passing is possible, congestion is still an issue. Passing removes most delays within the aisle itself. But blocking still happens at pick-faces, when two or more pickers need items from the same location at the same time. If pickers retrieve only one item per stop, blocking stays limited. If multiple items must be picked at the same location, the time a picker spends stopped increases, which makes blocking much more likely [29].

In wide aisles, blocking has a more linear-like relation with pick density, increasing monotonically, especially when multiple items are picked per stop. This differs from narrow-aisle systems, where congestion follows a non-linear pattern. As in narrow aisles, larger picking areas reduce blocking by giving pickers more room to spread out [29].

Congestion in batching and storage assignment
Batching and storage assignment influence congestion directly by determining which areas of the warehouse are visited simultaneously. Aisle width determines how congestion occurs. Batching and assignment decisions determine where and when picker interactions are likely to happen. Batching and sequencing models therefore need to explicitly account for blocking whenever multiple pickers work at the same time [20]. Batch picking typically releases several pick order in parallel, and since each pick order is carried out by one picker, the number of active pickers during a wave roughly matches the number of pick orders released. This makes congestion an important factor to consider in parallel picking environments [21].

Storage assignment also shapes congestion. As discussed in Section 2.1.1, class-based and random storage spread picks differently across aisles. This changes where picker traffic concentrates, and in turn, where blocking is likely to happen.

Several other studies confirm this picture [4, 17, 20, 27]: aisle width, pick times, worker density, and batching logic together determine how congestion builds up and spreads through the system. Batching and assignment together govern how much and when picker blocking happens [20]. Recent dynamic batching-routing methods show that batching can also cluster pick locations in space, pulling multiple pickers toward the same aisles and increasing blocking if not managed carefully [27].

2.2. The Aisle Load Balancing Problem

Congestion is recognized as an important factor in warehouse performance. Yet existing research rarely treats the density of pickers on aisle-level as an explicit decision variable. Instead, congestion tends to be a side effect of batching, routing, and assignment decisions, rather than something the system controls directly.

Building on these insights, this thesis treats aisle-level picker density as the main mechanism behind congestion. The core idea is that congestion can be reduced by limiting how many pickers are active in the same aisle at the same time. We refer to this scheduling perspective as the **Aisle Load Balancing Problem (ALBP)**. The maximum number of pickers simultaneously active in any aisle is formalized in Chapter 3 as the aisle multiplicity M, which serves as the main congestion metric used throughout this thesis.

In this problem, batching is assumed to be fixed. Each batch corresponds to a set of aisle visits. Each visit has an estimated processing time, capturing how long a picker occupies that aisle. Assignment decisions determine how these aisle visits overlap in time, and this overlap is what shapes congestion.

The next section introduces the scheduling terminology needed to formalize this problem, and highlights several structural properties of the setting. It closes by identifying the remaining research gap.

2.3. Scheduling Foundations for the Aisle Load Balancing Problem

To formulate the Aisle Load Balancing Problem within a scheduling framework we will first introduce the relevant scheduling terminology.

2.3.1. Core scheduling concepts

A scheduling formulation is often described using the $\alpha|\beta|\gamma$ notation, where the α describes the machine environment, β describes the job characteristics and γ gives the objective function [16]. The goal of a scheduling problem is to achieve an optimal value of the objective function while satisfying the machine and job requirements. The output of the problem is then a schedule where all jobs are scheduled in a certain order and on a certain machine. Typically, a set of m machines is given and a set of n jobs J.
The machines allow for different characteristics, these characteristics are given in the α part of the scheduling formulation. Within our research we will consider the following:

- *Single machine* (1): There is one machine which is denoted as a 1 in the formulation.
- *Parallel machines* (Pm): There are m identical machines which all have equal speeds of processing the jobs.

The jobs typically have characteristics of the following type:

- *Processing time* (p_{ij}): The processing time of job j on machine i, where the subscript i is left out when the processing time is independent of the machines.
- *Due date* (d_j): The time t at which a job j is needed to be finished. This allows for an objective function that penalizes jobs that are late.

Within the β field other constraints regarding the jobs can be described. Within this thesis we will consider the following:

- *Release date* (r_j): The time t at which job j becomes available to be scheduled.
- *Precedence constraints (prec)*: These constraints enforce a fixed ordering between jobs, meaning certain jobs must be completed before others can start
- *Resource constraints* (res$\lambda\sigma\rho$): Some jobs might need certain resources denoted λ, σ, ρ in order to be processed. This field of research will be investigated further in subsection 2.3.5.

In the γ field of the scheduling formulation, the objective of the scheduling problem is given. The ones considered in this thesis are the following:

- *Makespan* ($C_{\max}$): The finishing time of the schedule, i.e. the completion time of the last finishing job.
- *Total tardiness* ($\sum_j T_j$): The total amount of time by which all jobs exceed their respective due dates.

Comprehensive overviews of the introduced terminology, as well as the exact and heuristic solution approaches for such scheduling problems can be found in [31].

2.3.2. Scheduling formulation of the Aisle Load Balancing Problem

In the context of the ALBP, pickers can be interpreted as machines, while batches correspond to jobs. Each batch consists of a sequence of aisle visits that must be processed without idle time. Aisles represent shared resources that multiple jobs may need at overlapping times, and congestion arises when these time windows overlap. Assignment decisions determine when each job begins and thus how these aisle visits overlap.

This interpretation places the ALBP within a scheduling framework and highlights four structural components: fixed-sequence no-wait job structures, flow shop scheduling, resource constraints, and a bi-objective of minimizing peak resource usage alongside makespan. Figure 2.1 shows how these four components overlap to define the ALBP.

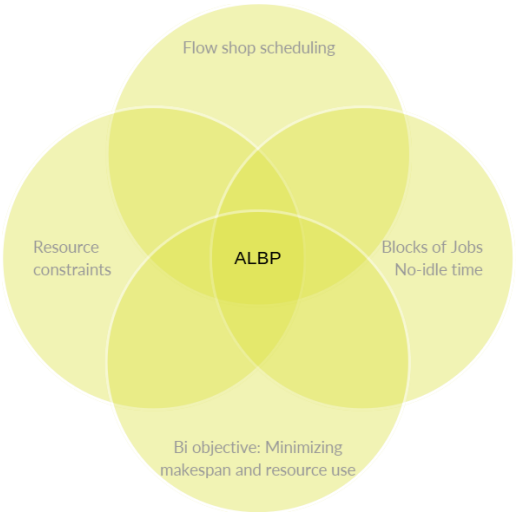


Figure 2.1: The four structural components that combine to form the Aisle Load Balancing Problem (ALBP): flow shop scheduling, blocks of jobs with no-idle time, resource constraints, and the bi-objective of minimizing makespan alongside resource use.

Using the $\alpha \mid \beta \mid \gamma$ notation of [16], we can draw the following parallels. Pickers are assigned batches, and therefore pickers correspond to machines while batches correspond to jobs. We assume identical processing speeds, leading to a parallel-machine environment with m identical machines, i.e., $\alpha = P_m$.

Each batch contains a sequence of aisle visits, forming a fixed sequence of subjobs that must run consecutively, without idle time, on the same machine (i.e. picker). This structure is captured in the β field by block (no-idle) constraints and internal precedence relations. The two work together: precedence relations fix the required order of a batch's aisle visits, while the no-idle condition ensures that each following visit begins exactly when its predecessor ends, with no gap in between. Together, they force each batch to run as a single, uninterrupted, ordered sequence. Each subjob corresponds to work performed in one aisle, and requires one unit of a renewable resource, with feasibility constrained by aisle capacities. This is modelled through renewable resource constraints, denoted res$\lambda\sigma\rho$.

The objective is to minimize either the makespan or the total tardiness of the schedule, giving $\gamma = C_{\max}$ or $\gamma = \sum_j T_j$. For now, we focus on makespan.

Combining these elements, the problem can be classified as

$$P_m \mid \text{block (no-idle)}, \; prec, \; \text{res}\lambda\sigma\rho \mid C_{\max}.$$

The following subsections review the scheduling literature corresponding to each of the four components and identify the gap arising from their combined presence in a single problem.

2.3.3. No-wait fixed sequence job structures

Jobs made up of subjobs with a fixed order and no idle time in between can, in simple cases, be treated as a single aggregated job. Its processing time is then just the sum of the processing times of its subjobs. This simplification is only valid, however, when the subjobs do not interact with external constraints such as shared resources.

When resource interactions are present, for example when multiple jobs need access to the same aisle, this simplification breaks down. Subjobs must then be modelled explicitly, to capture how they overlap in time and compete for resources. If aggregation is still applied anyway, the problem reduces to classical parallel machine scheduling: $P_m \| C_{\max}$ and $P_m \| \sum_j T_j$. Both are NP-complete once more than one machine is involved [31, 32].

Release dates and due dates can also be added to these models. The release date of a batch's first subjob is set to the release date of the batch, and the due date of its last subjob is set to the due date of the batch.

2.3.4. Flow shop and no-Wait flow shop analogies

The problem of scheduling subjobs sequentially with no idle time in between has a no-wait flow shop structure: operations must follow one another immediately, enforced through precedence constraints. Each operation starts exactly when its predecessor finishes. This type of constraint is widely used to model temporal dependencies in scheduling [16].

Classical flow shops work differently, though: every job moves through all machines in the same predetermined order. In the ALBP, each batch instead runs its whole fixed sequence of subjobs on a single machine (the picker). At the same time, each subjob is tied to a specific aisle, which acts as a shared resource across different batches.

Classical flow shop models do not account for shared resource constraints between jobs. These constraints, however, are essential for capturing aisle congestion. This is why an extension toward resource-constrained scheduling is needed.

2.3.5. Scheduling with renewable resource constraints

Resource-constrained scheduling is a major theme in the scheduling literature. In these problems, jobs use up units of (renewable) resources while being processed. Feasibility requires that the combined resource usage of all simultaneously active jobs never exceeds the available capacity. Błażewicz et al. [5] give an early and influential classification of these cumulative resource constraints.

Renewable resources are resources whose availability is restored at every point in time. A machine, in this sense, can also be seen as a renewable resource. During execution, each task consumes a given amount of the resource, which becomes available again once the task finishes. This cumulative view forms the basis of many resource-feasible scheduling models. Nudtasomboon and Randhawa [26] emphasize that the capacity of a renewable resource sets the maximum number of tasks that may use it at the same time. Feasibility then depends on coordinating activity start times, so that overlapping demand never exceeds this limit.

In the ALBP, each aisle can be modelled as a renewable resource: every subjob needs one unit of that resource for the duration of its aisle visit. The subjob's processing time therefore represents how long an aisle's capacity is occupied. This links aisle congestion directly to classical cumulative resource constraints, where feasibility depends on limiting how much simultaneous resource usage is allowed at any point in time.

As introduced earlier, resource-constrained scheduling is represented by res$\lambda\sigma\rho$ in the β field of the scheduling formulation.

- λ represents the number of resources. In our case, this is the number of aisles.
- σ represents the resource capacities, i.e. the maximum number of pickers allowed to be active in an aisle at the same time.
- ρ represents the number of resources a job needs. In our case, this is set to 1, since each subjob has one picker present in one specific aisle (resource).

Resource-constrained scheduling captures aisle interactions, but it does not yet account for the fixed internal structure of jobs. The next step is therefore to combine it with flow shop scheduling.

2.3.6. Resource constrained flow shops

Besides the established no-wait flow shop structure, feasibility is furthermore governed not only by machine assignment, but also by renewable aisle capacity constraints, which limit how many subjobs using the same aisle can overlap in time [33].

When renewable resources are introduced alongside machines, feasibility depends on resource availability, not only on machine order. Even with unit processing times and identically ordered operations, the scheduling problem becomes NP-hard beyond two machines. This motivates the use of bounding and exact search methods in restricted cases [33].

Models that add renewable resources on top of classical flow shop machines connect naturally to resource-constrained project scheduling (RCPSP) formulations. This connection shows that building a feasible schedule requires checking resource availability at every step. It is not enough to fix a single job

order and apply it to every machine, which is what classical permutation flow shop scheduling does [11].

Some models look specifically at how resources flow between operations: when an operation finishes and releases its resources, those resources can be reused by a later operation. In these models, the required spacing between two subjobs is not only set by precedence, but also by when the needed resources become free again, even if the machine itself is idle [10]. In other words, a job's start time can depend on resource availability, not just on whether a machine is free.

Other models let operation durations depend on how many resources are allocated to them. In a fixed, no-idle sequence, this matters: a delay at an early subjob has nowhere to be absorbed, so it propagates through the rest of the sequence [12]. A survey of extended flow shop problems, for example those with setups or similar added attributes, confirms this pattern: adding job- or resource-level constraints reliably increases problem complexity, which is why these problems need specialized algorithms [2].

Closest to our setting is the multi-mode resource-constrained project scheduling framework with generalized precedence relations, proposed by De Reyck and Herroelen [13]. In this framework, each job can be carried out in different modes, for example a fast mode that uses more resources, or a slow mode that uses fewer. The framework also allows general precedence relations between jobs, going beyond simple "one job must finish before another starts" rules. De Reyck and Herroelen show that under these conditions, both finding an optimal schedule, and even just finding a feasible one, are NP-hard.

The multi-mode aspect does not carry over to the ALBP: aisle subjobs have one fixed processing time and one fixed resource requirement, there is no choice between modes. What does carry over is the underlying mechanism: feasibility depends on the interaction between precedence and resource availability, not on machine assignment alone [13].

Further support comes from hybrid flow shop models that combine flow shop routing with limits on renewable resources [34]. In these models, each production stage acts as a capacity-constrained resource, and a feasible schedule must respect both the fixed operation sequence and the resource limits at each stage. This mirrors the ALBP: each batch follows a fixed no-wait order, while each aisle acts as a renewable resource whose capacity limits how much temporal overlap is allowed.

2.3.7. Bi-objective scheduling under resource constraints

The bi-objective nature of the ALBP most closely parallels the setting studied by Yepes-Borrero et al. [36]. They study a parallel-machine scheduling problem where sequence-dependent setups require a limited auxiliary resource. Their model minimizes two conflicting objectives at once: the makespan, and the maximum number of setup resources used at any moment.

Their key insight is that these two objectives pull in opposite directions. Compressing the schedule reduces makespan, but it also increases how much operations overlap in time, which pushes up peak resource usage. Reducing that peak works the other way: operations need to be postponed, or idle time needs to be introduced, both of which increase makespan. Their proposed T-RIPG algorithm handles this trade-off by delaying operations whenever resource limits would otherwise be violated.

This same conflict underlies the ALBP. Aisle visits use up renewable capacity during specific parts of each batch's fixed sequence. Minimizing makespan pushes toward using aisles concurrently. But limiting how many pickers use an aisle at once requires spacing these subjobs out in time, which increases completion times.

Related bi-objective formulations in project scheduling show the same mechanism. Abbasi et al. [1] show that adding robustness as a second objective, measured through activity slack, requires inserting time buffers between activities. This increases makespan. Similarly, Xia et al. [35] show that minimizing resource cost in multi-mode project scheduling pushes the schedule toward slower, more resource-efficient execution modes, which also leads to longer project durations. Across these models, reducing resource usage means spacing activities out in time, which inherently increases makespan.

This trade-off shows up again in resource-constrained flow shop settings. Renewable resource limits restrict how much subjobs can overlap, so a tighter schedule that reduces makespan pushes peak resource load higher [33]. When resource use is added to classical flow shop models, resource scarcity

forces operations further apart, which increases completion times [11]. When resource availability depends on when later subjobs finish, feasibility becomes path dependent, and delays propagate through the sequence [10]. Models where processing times depend on resource allocation show the same effect: limited resources slow down early operations, and these delays then amplify through the rest of the fixed-order sequence [12].

Taken together, these results place the ALBP in a class of bi-objective, resource-constrained scheduling problems, where minimizing makespan and minimizing peak resource usage are fundamentally in conflict. Managing this trade-off is therefore central to how the problem is formulated and solved.

2.4. Research Gap and Problem Positioning

No existing work fully captures the combined structure of the ALBP, despite the large body of literature on warehouse operations, scheduling, and congestion. The preceding sections show that the relevant building blocks have all been studied on their own. Combining them fully, however, introduces new challenges. Figure 2.1 shows where the ALBP sits within the existing scheduling literature.

From a warehouse perspective, batching, routing, and assignment decisions are usually optimized either one after another, or jointly within integrated heuristic frameworks [9, 23]. These approaches influence congestion, but only indirectly: they focus on travel time or tardiness, and do not control the density of pickers on aisle-level as an explicit decision variable. Congestion is typically treated as an outcome of these operational choices, rather than as an objective to minimize directly [20, 27].

From a scheduling perspective, parallel machine models capture assignment decisions. Flow shop formulations capture fixed job sequences. Resource-constrained scheduling models capture shared resource limits [5, 31]. These three frameworks are rarely combined, however, in a setting that enforces no-wait job structures, parallel machine assignment, and renewable resource constraints tied to spatial entities such as aisles, all at the same time. Classical models also tend to treat resource capacities as fixed feasibility constraints, rather than as something to actively minimize.

The literature on resource-constrained flow shops and RCPSP extends these models further, by adding both precedence relations and shared resources [11, 13, 33]. These approaches still do not address how spatial routing structures interact with resource usage over time, and they do not treat resources as physical congestion points within a warehouse.

Finally, bi-objective scheduling models do consider the trade-off between makespan and resource usage [1, 35, 36], but not in this warehouse context. As a result, the link between route-related resource demand and scheduling decisions remains largely unexplored.

Together, this reveals a clear gap: there is no unified framework that combines

1. fixed sequence no-wait job structures,
2. flow shop scheduling,
3. resource constraints, and
4. a bi-objective of minimizing peak resource usage alongside makespan,

in a warehouse environment.

The ALBP addresses this gap. It explicitly models the density of pickers on aisle-level as a controllable scheduling dimension, by linking assignment and release decisions to resource usage over time. This turns congestion from an emergent side effect into a quantity that can be actively optimized.

3 Method

Chapter 2 introduced aisle congestion as a key bottleneck in the picking process, and this chapter turns that operational problem into a mathematical formulation that can be analyzed and solved algorithmically. Section 3.1 introduces the formal version of the problem, gives its ILP formulation, and proves that it is NP-complete. Section 3.2 extends this formulation with aisle sequencing, non-uniform aisle processing times, and startup and wrap-up phases. Since solving these ILP's to optimality does not scale to realistic instances, Section 3.3 introduces two greedy heuristics as faster alternatives. Section 3.4 describes how all three methods are evaluated on the warehouse data presented in Chapter 4.

3.1. The Aisle Load Balancing Problem

We first formulate the Aisle Load Balancing Problem (ALBP) by introducing a simplified representation of the picking process. A pick order is modelled as a batch B_i, where the batch consists of the set of aisles that need to be visited. From this point onward, we use the term batch when referring to a pick order in the model. In the initial formulation, we assume that all batches have equal processing times and that the aisles within a batch are processed in parallel. Under these assumptions, the ALBP focuses on balancing the workload by assigning batches to pickers over consecutive timeslots. Later, we extend this formulation to incorporate more realistic characteristics of the operation, including sequential aisle processing and non-uniform processing times.

Formally, we are given n batches, where each batch corresponds to a subset of aisles from $[A]$. Furthermore, we assume a constant workforce of w pickers. Since all batches are assumed to have equal processing times, each timeslot consists of exactly w batches being processed simultaneously, with each batch assigned to a different picker. Therefore, the n batches must be partitioned into sets P_j of size w, where each set represents the batches processed in parallel by the pickers during one timeslot.

Although this formulation abstracts away several operational details, it already captures the core difficulty of the problem. In fact, even this simplified version of the ALBP is NP-complete, as we show in Subsection 3.1.3.

Remark. *We assume that a batch's route never revisits an aisle. This matches actual picking behaviour, where a picker never returns to an aisle already visited within the same route.*

To describe aisle usage within a set P_j, a plain set union is not enough, since it would lose the information of how many batches in P_j need the same aisle. We therefore use a multiset. A *multiset* over a ground set S is a collection in which elements may occur more than once; the number of occurrences of an element s is called its *multiplicity*. We define the function for the maximum multiplicity over a multiset

$$m : \mathcal{M} \to \mathbb{N}. \tag{3.1}$$

This function gives the maximum multiplicity of the elements in a given multiset. Given multisets $M_1, \ldots, M_k$, their *multiset sum* $M_1 \uplus \cdots \uplus M_k$ is the multiset in which the multiplicity of each s is the sum of its multiplicities across $M_1, \ldots, M_k$. Let us consider the following example of a multiset sum.

Example 3.1.1. *Consider the sets* $M_1 = \{1, 2, 3\}$, $M_2 = \{2, 3, 4\}$, *and* $M_3 = \{2, 5\}$. *Their multiset sum is*

$$\mathcal{M} = M_1 \uplus M_2 \uplus M_3 = \{1, 2, 2, 2, 3, 3, 4, 5\},$$

we can also represent $\mathcal{M}$ *as* $\{1^{(1)}, 2^{(3)}, 3^{(2)}, 4^{(1)}, 5^{(1)}\}$. *Taking the maximum multiplicity of this multiset gives:* $m(\mathcal{M}) = 3$.

For any set P_j in our partition, the aisles used by these pickers are given by the multiset sum of the aisle sets of the batches in P_j. Given a partition, define the *aisle multiplicity* of P_j as the multiplicity attained by an aisle in this multiset sum. Our objective is to choose a partition of the n batches into sets of size w that minimizes the *maximum aisle multiplicity* over all sets P_j in the partition. In the next subsection we introduce the formal definition of the Aisle Load Balancing Problem.

3.1.1. Formal problem definition of the Aisle Load Balancing Problem

Input:

$$\begin{aligned} n, w, A &\in \mathbb{Z} && \text{where } w|n \\ B_i &\subseteq [A] && \text{for } i \in [n] \end{aligned}$$

Output:

$$\begin{aligned} &\text{Partition } [n] \text{ into subsets } P_j \text{ of size } w, \\ &[n] = P_1 \cup P_2 \cup \ldots \cup P_{\frac{n}{w}}, \text{ such that:} \\ &\max_{j \in [n/w]} \left\{ m\left(\biguplus_{i \in P_j} B_i \right) \right\}, \text{ is minimized.} \end{aligned}$$

3.1.2. ILP formulation of the Aisle Load Balancing Problem

There are n batches, and the number of pickers is equal to w. The batches need to be assigned to pickers, where we assume w divides n so that every picker is assigned an equal number of batches. The batches are modelled as subsets of the aisles $[A]$.

Input:

$$\begin{aligned} &n, w, A \in \mathbb{Z}, \quad w \mid n, \\ &B_i \subseteq [A] \quad \text{for } i \in [n]. \end{aligned}$$

Define the parameter indicating whether aisle $a \in [A]$ is contained in batch B_i:

$$b_{ai} = \begin{cases} 1 & \text{if } a \in B_i, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } a \in [A],\ i \in [n]. \tag{3.2}$$

For the maximum multiplicity we introduce M. Furthermore, we introduce a binary decision variable x_{ij} for deciding to which set P_j we assign batch B_i.

Decision variables:

$$\begin{aligned} &M \in \mathbb{Z}_{\geq 0}, \\ &x_{ij} = \begin{cases} 1 & \text{if } i \in P_j, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } i \in [n],\ j \in \left[\frac{n}{w} \right], \\ &\qquad\qquad \text{where } \{P_j\}_{j \in [n/w]} \text{ is the partition of } [n] \text{ into sets of size } w. \end{aligned}$$

The objective is to minimize the maximum multiplicity.

Objective:

$$\min M \tag{3.3}$$

Constraints: Each batch must be assigned to exactly one set P_j:

$$\sum_{j=1}^{n/w} x_{ij} = 1, \qquad \forall i \in [n]. \tag{3.4}$$

Each set P_j has exactly w batches:

$$\sum_{i=1}^{n} x_{ij} = w, \qquad \forall j \in \left[\frac{n}{w} \right]. \tag{3.5}$$

For every set P_j and aisle a, the number of batches in P_j that use aisle a is bounded by M:

$$\sum_{i=1}^{n} b_{ai}\, x_{ij} \leq M, \qquad \forall j \in \left[\frac{n}{w}\right],\ \forall a \in [A]. \tag{3.6}$$

One can verify that this ILP solves the following:

$$\min \max_{j \in [n/w]} \left\{ m\left(\biguplus_{i \in P_j} B_i \right) \right\}, \tag{3.7}$$

with $\{P_j\}_{j \in [n/w]}$ forming a partition of $[n]$ into sets of size w.

Symbol	Type	Index Set	Interpretation
n	Input parameter	–	Total number of batches.
w	Input parameter	–	Number of pickers, size of the sets.
A	Input parameter	–	Total number of aisles.
B_i	Input parameter	$i \in [n]$	Set of aisles in the route of batch i.
b_{ai}	Input parameter	$i \in [n],\ a \in [A]$	1 if aisle a is in B_i.
x_{ij}	Decision variable	$i \in [n],\ j \in [n/w]$	1 if batch B_i is assigned to set P_j.
M	Decision variable	–	Maximum aisle multiplicity

Table 3.1: Overview of variables and parameters in the ILP formulation in Section 3.1.2.

A compact version of the ALBP can be found in Appendix A.1.1. Before we prove NP-completeness of this problem, we illustrate the model by looking at an example of the input and output of the model.

Example 3.1.2. *Suppose we have the following inputs to the model*

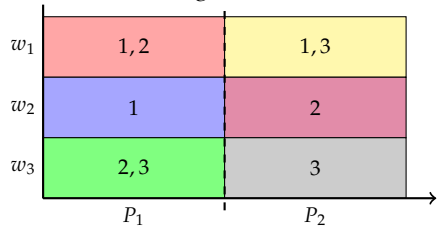
- $n = 6$
- $w = 3$
- $A = 3$

Suppose the 6 batches that we have are the following, where the numbers within each block indicate the aisles they contain.

We have 6 batches that all contain a subset of the 3 aisles available. These need to be partitioned into 2 sets of 3 batches such that we minimize the maximum multiplicity of the aisles in the partitioned multiset:

$$P_1 = \{\{1,2\},\{1\},\{2,3\}\}, \tag{3.8}$$
$$P_2 = \{\{1,3\},\{2\},\{3\}\}. \tag{3.9}$$

This partition is also illustrated in the figure below. The dashed line in the middle represents the "partition" of the sets where we look to minimize the maximum aisle multiplicity in the multiset sum of 3 batches on the left and the 3 batches on the right.



The multiset of the first three batches is the following: $\{1^{(2)}, 2^{(2)}, 3^{(1)}\}$*, it has maximum multiplicity 2 for both*

aisles 1 and 2. The multiset of the second three sets is: $\{1^{(1)}, 2^{(1)}, 3^{(2)}\}$*, which has a maximum multiplicity of 2 for aisle 3. The maximum multiplicity of the multisets that partition the set of batches is thus 2.*

One can observe that, if we take the multiset of all 6 batches, all aisles have multiplicity 3. Since each aisle's total count of 3 must be split across only 2 sets, one set must contain at least $\lceil \frac{3}{2} \rceil = 2$ *aisle occurrences. Every partition will therefore result in a maximum aisle multiplicity of at least 2. As a result the partition shown above, which achieves multiplicity 2, is an optimal solution to this instance of the ALBP.*

3.1.3. NP-completeness of the Aisle Load Balancing Problem

To prove NP-completeness of the ALBP, we use the decision version defined below.

Decision version of ALBP

Input:

$$\begin{aligned} &n, w, A \in \mathbb{Z}, \quad w \mid n, \\ &B_i \subseteq [A] \quad \text{for } i \in [n], \\ &M \in \mathbb{Z}_{\geq 0}. \end{aligned}$$

Question: Does there exist a partition of $[n]$ into subsets P_j of size w,

$$[n] = P_1 \cup P_2 \cup \cdots \cup P_{n/w}, \quad P_j \subseteq [n],\ |P_j| = w,\ P_j \cap P_{j'} = \emptyset \text{ for } j \neq j', \tag{3.10}$$

such that

$$\max_{j \in [n/w]} m\left(\biguplus_{i \in P_j} B_i \right) \leq M\,? \tag{3.11}$$

We now introduce the NP-complete problem: edge colouring. We will reduce an instance of edge colouring to an instance of ALBP to show that the latter problem is NP-complete.

A *graph* $G = (V, E)$ consists of a set of vertices V and a set of edges E, where each edge connects a pair of vertices. Edges are *adjacent* if they share a common node. An *edge colouring* of G is an assignment of colours to the edges such that no two adjacent edges receive the same colour. The minimum number of colours needed to achieve this is called the *chromatic index* $\chi'(G)$.

Edge-Colouring (decision version).
Input: A simple graph $G = (V, E)$ and an integer $k \in \mathbb{Z}_{\geq 0}$.

Question: Does there exist a function

$$c : E \to [k]$$

such that for every pair of edges $e, e' \in E$,

$$e \cap e' \neq \emptyset \implies c(e) \neq c(e')\,?$$

Equivalently, can the edges of G be coloured with at most k colours such that no two edges sharing a vertex receive the same colour?

Next we will prove NP-Completeness.

Theorem 3.1.1. *The Aisle Load Balancing Problem is NP-complete.*

Proof. We first show that the ALBP is in NP. The class of NP consists of problems of which a proposed solution can be verified within polynomial time and can be solved in non-deterministic polynomial time. If we can verify YES-instances of the ALBP within polynomial time we prove the ALBP is in NP, this can be done using the procedure described below.

A certificate consists of a partition of the batches into sets

$$P_1, P_2, \ldots, P_{n/w},$$

each of size exactly w.

Given such a certificate, the following three checks need to be done:

1. **Partition validity.**
   Verify that every batch appears in exactly one set P_j, and that $|P_j| = w$ for all j. This requires taking the union of all batches once, which is polynomial in n.
2. **Multiplicity computation.**
   For each set P_j, compute the multiset sum

   $$\biguplus_{i \in P_j} B_i$$

   and determine, for every aisle $a \in [A]$, the multiplicity

   $$m_j(a) = |\{\, i \in P_j : a \in B_i \,\}|.$$

   Since each batch B_i is a subset of $[A]$, this can be computed in time $O(nwA)$, which is in polynomial time.
3. **Feasibility check.**
   Verify that for every set P_j,

   $$\max_{a \in [A]} m_j(a) \leq M,$$

   this can be done in time $O(\frac{n}{w}A)$.

All verification steps run in time polynomial in the size of the input. Therefore, the ALBP is in NP.

Reduction from Edge-Colouring to ALBP.
We reduce an instance of the *Edge-Colouring* problem

$$(G = (V, E),\ k)$$

to an instance of the *ALBP*

$$(n, w, A, \{B_1, ..., B_n\}, M)$$

with multiplicity bound $M = 1$.

Construction of the ALBP instance.

- **Aisles.**
  Let the number of aisles be

  $$A = |V|.$$

  Each vertex $v \in V$ corresponds to a unique aisle.
- **Batches.**
  Enumerate the edges as

  $$E = \{e_1, \ldots, e_{n_0}\}.$$

  For every edge $e_i = \{u, v\} \in E$, create a batch

  $$B_i = \{u, v\}.$$

  Hence, the initial number of batches is

  $$n_0 = |E|.$$

- **Time slots and number of pickers.**
  Set the number of pickers to
  $$w = |E|.$$
  Since every colour class is a subset of E, no colour class can ever exceed size w, regardless of how the colouring balances the edges.
- **Dummy batches.**
  Add empty batches until the total number of batches satisfies
  $$n = kw,$$
  so that the batches can be partitioned into exactly k sets of size w. More precisely, for
  $$i = n_0 + 1, \ldots, n,$$
  define
  $$B_i = \emptyset.$$
  These dummy batches do not affect aisle multiplicities.

This completes the construction of the ALBP instance.

We now show that the graph G is k-edge-colourable if and only if the ALBP has a feasible solution with multiplicity bound $M = 1$.

Suppose first that G is k-edge-colourable. Let
$$c : E \to [k]$$
be a proper edge-colouring.

For each colour j, let
$$d_j = w - |\{ B_e : c(e) = j \}|$$
be the number of dummy batches required to make P_j have size w. Since $\{B_e : c(e) = j\} \subseteq E$ and $w = |E|$, we have $d_j \geq 0$ for every colour j, regardless of how the colouring distributes edges across classes.

Partition the set of dummy batches into disjoint sets
$$D_1, D_2, \ldots, D_k$$
such that
$$|D_j| = d_j$$
for every $j \in [k]$. Then define
$$P_j = \{ B_e : c(e) = j \} \cup D_j. \tag{3.12}$$

Since no two adjacent edges share a colour, each vertex v appears in at most one edge whose corresponding batch belongs to P_j. The dummy batches in D_j are empty and therefore do not contribute to any aisle multiplicity. Hence,
$$m\left(\biguplus_{B_i \in P_j} B_i\right) \leq 1 = M. \tag{3.13}$$

The sets $P_1, \ldots, P_k$ form a partition of the batches $\{B_1, \ldots, B_n\}$, and the multiplicity of every aisle in every set P_j is at most 1. Hence, the constructed ALBP instance with $M = 1$ has a feasible solution.

Suppose the constructed ALBP instance has a feasible solution with multiplicity bound $M = 1$. For each set P_j, define a colouring $c : E \to \{1, \ldots, k\}$ by setting
$$c(e) = j \quad \text{if and only if} \quad B_e \in P_j.$$

If two adjacent edges $e = \{u, v\}$ and $e' = \{u, v'\}$ received the same colour j, then both corresponding batches B_e and $B_{e'}$ belong to P_j. Since $u \in B_e$ and $u \in B_{e'}$, aisle u appears at least twice in the multiset sum $\biguplus_{B \in P_j} B$. Hence the maximum multiplicity satisfies $m_j \geq 2$, which contradicts the feasibility condition $m_j \leq M = 1$. So all pairwise adjacent edges receive different colours, therefore we can conclude that c is a proper edge-colouring.

Hence, G is k-edge-colourable if and only if the constructed ALBP instance admits a feasible solution with multiplicity bound $M = 1$. Since k-Edge-Colouring is NP-complete and the reduction can be done in polynomial time, the ALBP is NP-hard. Together with the fact that ALBP $\in$ NP, we conclude that the ALBP is NP-complete. □

3.2. The Extended Aisle Load Balancing Problem

The problem defined in Sections 3.1.1 and 3.1.2 abstracts away several aspects of real warehouse operations. In practice, a picker does not operate in multiple aisles simultaneously, batches have non-uniform processing times, and the picking process is embedded in a broader workflow with strict timing requirements. Three extensions are introduced to capture these realities.

First, to capture the time dimension of the order picking process, we discretize the planning horizon into a finite number of uniform time intervals. This is common practice in the scheduling literature, since it lets a continuous-time planning problem be reformulated as an integer program. The time horizon is represented by the set

$$[T]_{\geq 0} = [T] \cup \{0\}, \tag{3.14}$$

where 0 denotes the start of the horizon, before any picking activity begins.

Second, aisle processing times are allowed to vary across batches, replacing the equal-time assumption and capturing the variability in how long a picker spends in each aisle.

Third, each batch is assigned a startup and wrap-up time, reflecting the real-life processes that happen before and after picking. The startup phase, during which pickers retrieve carts, and the wrap-up phase, during which pickers place carts on their outbound lane, are modelled as dummy aisles labelled -1 and 1000. These always appear as the first and last aisle of a batch and contribute to the total processing time of a batch. To accommodate them, the aisle index set is extended to

$$[A]_{\text{ext}} = [A] \cup \{-1, 1000\}. \tag{3.15}$$

Since pickers move freely during these phases and do not occupy a physical aisle, the dummy aisles are excluded when measuring the aisle load.

Together, these extensions allow us to minimize the maximum number of pickers simultaneously present in the same aisle at any given time. The models introduced in this chapter require estimates of processing times, feasible scheduling windows, and aisle-level processing times. The derivation of these parameters from operational warehouse data is described in Chapter 4.

We first present a formal mathematical definition of the Extended ALBP, followed by its corresponding ILP formulation.

3.2.1. Formal problem definition of the Extended ALBP

Input:

$$n, w, A, \beta_i, T \in \mathbb{Z}_{\geq 0}$$
$$B_i = \left((-1_{i1}, p_{i1}), (a_{i2}, p_{i2}), \ldots, (1000_{i\beta_i}, p_{i\beta_i})\right) \in ([A]_{ext} \times \mathbb{Z}_{\geq 0})^{\beta_i}, \qquad \text{for } i \in [n], \beta_i \in |B_i|$$
$$EST_i,\ LST_i \in [T]_{\geq 0}$$

Output:

Start times $x_{it} \in \{0, 1\}$ for all $i \in [n]$, where $\sum_{EST_i}^{LST_i} x_{it} = 1$.

For any time $t \in [T]_{\geq 0}$, it holds that:

$$\sum_{a=1}^{A} m_{at} \leq w.$$

Where m_{at} is the multiplicity of batches B_i processing aisle a at time t (i.e. Number of pickers in aisle a at time t).

The objective is to minimize $\max_{a \in [A],\ t \in [T]_{\geq 0}} \{m_{a,t}\}$.

3.2.2. ILP formulation of the Extended ALBP

This subsection presents an integer linear programming (ILP) formulation for an extended version of the aisle load balancing problem (ALBP). In this extended problem, each batch consists of a predefined sequence of aisles with individual processing times. Aisles within a batch are processed sequentially and non-preemptively, meaning a picker cannot be interrupted once they starts an aisle. Multiple batches may run in parallel subject to a picker capacity limit. To enforce capacity and congestion constraints at the aisle level, the model introduces explicit variables tracking when each aisle within a batch is active.

Input parameters. The model is defined over a discrete planning horizon and uses the following input data:

- n: total number of batches.
- w: number of available pickers, interpreted as the maximum number of batches that can be processed simultaneously.
- T: number of discrete time periods in the planning horizon.
- A: total number of aisles in the warehouse.
- For each batch B_i, a set of tuples (a_{ik}, p_{ik}), where $a_{ik} \in [A]_{ext}$ indicates the index of the k-th visited aisle and $p_{ik} \in \mathbb{Z}_{\geq 0}$ denotes the corresponding processing time of the k-th aisle visited by batch B_i.
- β_i: the number of aisles processed by batch B_i.
- $P_i = \sum_{k=1}^{\beta_i} p_{ik}$: the total processing time of batch B_i.
- $\{EST_i, EST_{i+1} \ldots, LST_i\}$: the allowable time window in which batch B_i may start processing, defined by its earliest and latest start times.

Each batch is represented as a set of tuples containing the aisles and their corresponding processing time

$$B_i = \left(\left(a_{i1}, p_{i1}\right), \left(a_{i2}, p_{i2}\right), \ldots, \left(a_{i\beta_i}, p_{i\beta_i}\right)\right), \tag{3.16}$$

where β_i denotes the number of aisles associated with batch B_i. This thus contains the aisles -1 and 1000 corresponding to the startup and wrap-up step time of a batch. These are always the first and last aisle of a batch, i.e. $a_{i1} = -1$, $a_{i\beta_i} = 1000$.

Decision variables. To determine the schedule, we introduce the following binary variables:

$$x_{it} = \begin{cases} 1 & \text{if batch } B_i \text{ starts at time } t, \\ 0 & \text{otherwise,} \end{cases} \qquad \forall i \in [n],\ t \in [T]_{\geq 0}\,, \tag{3.17}$$

$$r_{ikt} = \begin{cases} 1 & \text{if the } k\text{-th aisle of batch } B_i \text{ is being processed at time } t, \\ 0 & \text{otherwise,} \end{cases} \qquad \forall i \in [n],\ k \in [\beta_i],\ t \in [T]_{\geq 0}\,. \tag{3.18}$$

The x-variables determine the batch start times, while the r-variables explicitly track when each aisle within each batch is active. This disaggregation allows capacity and congestion constraints to be enforced at the aisle level.

To enforce constraints regarding the execution status of an aisle in a batch and the maximum multiplicity of the specific aisles through time we predefine the following sets.

Predefined sets. Since aisles within a batch are processed sequentially, we define the time offset at which the k-th aisle of batch B_i starts relative to the batch start. This is equal to the total processing time of the first $k-1$ aisles of batch B_i:

$$S_{ik} = \sum_{h=1}^{k-1} p_{ih}. \tag{3.19}$$

We use this offset to define the running status of an aisle within a batch.

Objective. The objective is to minimize the maximum number of simultaneous executions of the same aisle over all time periods. This quantity is captured by the auxiliary variable M:

$$\min M. \tag{3.20}$$

Constraints. Each batch must start exactly once within its prescribed time window:

$$\sum_{t=EST_i}^{LST_i} x_{it} = 1 \qquad \forall i \in [n]. \tag{3.21}$$

Remark. *Depending on the application, the start time constraint may be relaxed by replacing the equality with an inequality and adding a penalty term to the objective. This is appropriate when not all batches are required to be scheduled within the planning horizon.*

Using the offset defined in 3.19, the execution status of aisle k of batch B_i at time t is defined as:

$$r_{ikt} = \sum_{\tau=\max\{0,\, t-S_{ik}-p_{ik}+1\}}^{t-S_{ik}} x_{i\tau} \qquad \forall i \in [n],\ \forall k \in [\beta_i],\ \forall t \in [T]_{\geq 0}\,. \tag{3.22}$$

This constraint ensures that $r_{ikt} = 1$ precisely during the time periods in which aisle k is being processed, given the selected start time of batch B_i. A visual illustration of this relationship is provided in Table 3.3 in Example 3.2.1.

Picker capacity is limited in every time period, so the total number of simultaneously processed aisles cannot exceed w:

$$\sum_{i=1}^{n} \sum_{k=1}^{\beta_i} r_{ikt} \leq w \qquad \forall t \in [T]_{\geq 0}\,. \tag{3.23}$$

Finally, the maximum number of batches simultaneously occupying the same aisle over all time periods is bounded by the auxiliary variable M. Aisle identity is determined entirely by the parameter a_{ik}: for a

batch B_i, position k points to exactly one physical aisle, namely $a_{ik} \in [A]$. This index k is local to each batch, so the same value of k across different batches does not refer to the same physical aisle; the aisle a given position corresponds to can only be read off through a_{ik} itself. When aggregating aisle load across batches, this is why positions cannot be compared directly by k, and must instead be filtered by the aisle they point to. Using this aggregation the aisle load is constrained as follows:

$$\sum_{i=1}^{n} \sum_{\substack{k \in [\beta_i] \\ a_{ik}=a}} r_{ikt} \leq M \qquad \forall a \in [A],\ \forall t \in [T]_{\geq 0}\,. \tag{3.24}$$

This constraint ensures that, for each physical aisle a and time t, the total number of batches simultaneously processing that aisle does not exceed M.

Remark. *The aisles* -1 *and* 1000 *are not taken into account when measuring the maximum aisle load.*

Symbol	Type	Index Set	Interpretation
n	Input parameter	–	Total number of batches.
w	Input parameter	–	Number of pickers.
T	Input parameter	–	Number of time periods in the horizon.
A	Input parameter	–	Number of aisles.
B_i	Input parameter	$i \in [n]$	A batch that contain the aisles and their processing times.
β_i	Input parameter	$i \in [n]$	Counter that states how many aisles batch B_i contains.
a_{ik}	Input parameter	$i \in [n],\ k \in [\beta_i]$	Index of the k-th aisle in batch B_i
p_{ik}	Input parameter	$i \in [n],\ k \in [\beta_i]$	Processing time of the k-th aisle in batch B_i.
P_i	Input parameter	$i \in [n]$	Total processing time of batch B_i.
S_{ik}	Input parameter	$i \in [n],\ k \in [\beta_i]$	Cumulative processing time of the $k-1$ aisles preceding position k in batch B_i's sequence, added to B_i's start time to get the start time of the aisle at position k.
x_{it}	Decision variable	$i \in [n],\ t \in [T]_{\geq 0}$	1 if batch B_i starts at time t.
r_{ikt}	Decision variable	$i \in [n],\ k \in [\beta_i]$ $t \in [T]_{\geq 0}$	1 if aisle k of batch B_i is being processed at time t.
M	Decision variable	–	Maximum aisle multiplicity across all aisles and times.

Table 3.2: Notation summary for the extended ALBP ILP formulation.

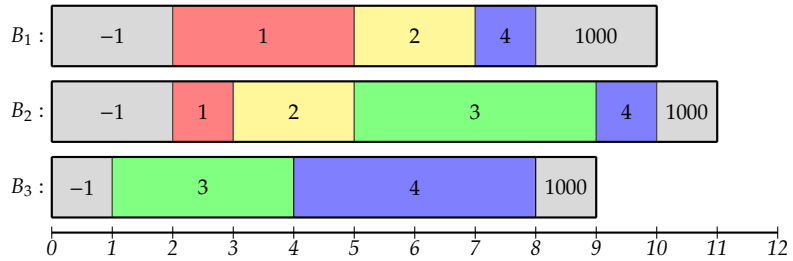
Example 3.2.1. *Suppose we have the following input to the model*

- $n = 3$
- $w = 3$
- $A = 4,\ [A]_{\text{ext}} = \{-1, 1, 2, 3, 4, 1000\}$

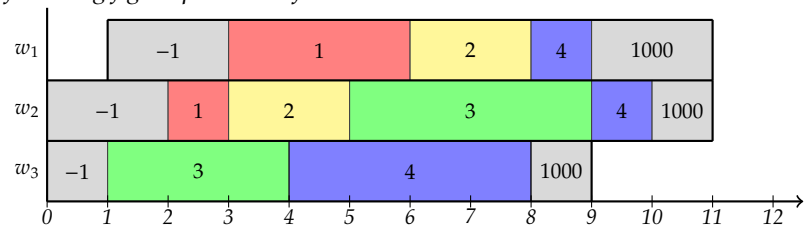
We consider three batches, each containing a subset of aisles and an associated processing time. The batches are visualized as a sequence of blocks, where the horizontal length of each block represents its processing time. To make the connection between the model input and the visualization clearer, we also represented the batches as an ordered set:

$$B_1 = ((-1, 2), (1, 3), (2, 2), (4, 1), (1000, 2)) \tag{3.25}$$
$$B_2 = ((-1, 2), (1, 1), (2, 2), (3, 4), (4, 1), (1000, 1)) \tag{3.26}$$
$$B_3 = ((-1, 1), (3, 3), (4, 4), (1000, 1)) \tag{3.27}$$

The number on the blocks represents the specific aisle that needs to be processed. For all batches we assume their EST_i *and* LST_i *between* $t = 0$ *and* $t = 3$*. As for the time horizon, we take it equal to* $T = 12$*.*



We have 3 batches that all contain a subset of the 4 aisles available together with a processing time. All batches have startup and wrap-up step times represented by aisles −1 *and* 1000*. These values are chosen as they do not correspond to the number of any physical aisle and we do not consider more than 100 aisles. The batches need to be assigned to pickers so that the maximum multiplicity of aisles processed at the same time is minimized. The following figure presents a feasible schedule.*



At all times t the maximum multiplicity of all 4 aisles is either 0 or 1. As the maximum multiplicity is at least 1 for all aisles, it is trivial that this is an optimal solution to the extended ALBP. For this schedule, we have the following x_{it} *equal to 1*

- $x_{11} = 1$
- $x_{20} = 1$
- $x_{30} = 1$

All other x_{it} *are* 0 *for the 3 batches for* $t \in [T]_{\geq 0}$*. Another optimal schedule would be if we shift batch* B_1 *by 2 time units and set* $x_{13} = 1$*. For each of the 3 batches and their* x_{it}*, we have their corresponding variables* r_{ikt} *for all* $k \in [\beta_i]$ *and* $t \in [T]_{\geq 0}$*. The values of* r_{ikt} *can be observed in Table 3.3. From these tables, the processing time with respect to the startup and wrap-up step can be easily observed. We see that they are taken into account with regards to the total processing time and when the picker enters the physical aisles. However, the aisles* −1 *and* 1000 *are not taken into account for the maximum aisle load as stated before.*

Table 3.3: Aisle-processing indicators r_{ikt} induced by the schedule of the second figure in Example 3.2.1.

r_{1kt}

k	a_{ik}	Time t										
		0	1	2	3	4	5	6	7	8	9	10
1	-1	0	1	1	0	0	0	0	0	0	0	0
2	1	0	0	0	1	1	1	0	0	0	0	0
3	2	0	0	0	0	0	0	1	1	0	0	0
4	4	0	0	0	0	0	0	0	0	1	0	0
5	1000	0	0	0	0	0	0	0	0	0	1	1

r_{2kt}

k	a_{ik}	Time t										
		0	1	2	3	4	5	6	7	8	9	10
1	-1	1	1	0	0	0	0	0	0	0	0	0
2	1	0	0	1	0	0	0	0	0	0	0	0
3	2	0	0	0	1	1	0	0	0	0	0	0
4	3	0	0	0	0	0	1	1	1	1	0	0
5	4	0	0	0	0	0	0	0	0	0	1	0
6	1000	0	0	0	0	0	0	0	0	0	0	1

r_{3kt}

k	a_{ik}	Time t										
		0	1	2	3	4	5	6	7	8	9	10
1	-1	1	0	0	0	0	0	0	0	0	0	0
2	3	0	1	1	1	0	0	0	0	0	0	0
3	4	0	0	0	0	1	1	1	1	0	0	0
4	1000	0	0	0	0	0	0	0	0	1	0	0

Remark. *Section 3.1.3 proves that the ALBP is NP-complete. The extended ALBP adds non-uniform processing times and enforces a strict visiting order within each batch, which makes it a genuinely different model rather*

than a direct generalization, so the reduction of Theorem 3.1.1 does not carry over automatically. We do not give a separate hardness proof for the extended ALBP here. Given its similarity to resource constrained flow shop scheduling, which is known to be NP-hard even with unit processing times and only a few machines [33], we expect the extended ALBP to be NP-hard as well, and leave a formal proof for future work.

3.3. Heuristics

Solving the ILP formulations from Sections 3.1 and 3.2 to optimality can be computationally demanding. Therefore, this section introduces two greedy heuristics that construct schedules directly, rather than relying on the optimal solution of an ILP formulation. The two heuristics differ in what they treat as fixed. The first heuristic, presented in Section 3.3.1, minimizes the maximum aisle load M directly while scheduling batches, using a limited look-ahead to avoid unnecessary increases in aisle load. The second heuristic, presented in Section 3.3.2, instead takes a maximum aisle load M as a fixed input, and builds the best schedule it can without exceeding it. Comparing the two shows what schedule quality can still be achieved once a hard limit is placed on aisle load.

3.3.1. Greedy heuristic with limited look-ahead

The heuristic constructs a schedule incrementally, assigning batches to pickers one at a time over a discrete time horizon. At each step, it selects a batch to schedule while checking whether doing so would increase the maximum aisle multiplicity M. Batches are prioritized by urgency, but the algorithm is able to look past the most urgent batch to see if a less congesting alternative is available. If no such alternative exists, the most urgent batch is scheduled and M is updated accordingly. The result is a feasible schedule with respect to the earliest start time constraint that attempts to keep M as small as possible while respecting picker capacity and time-window constraints.

Subroutines: aisle load check and aisle load update

Two subroutines support the main scheduling procedure. Both operate on the aisle load matrix $aisle\_load_a(t)$, which tracks the number of batches concurrently active in aisle a at time t. This matrix is the heuristic counterpart of the variables r_{ikt} from the ILP: rather than tracking individual batch-aisle assignments, it aggregates them into a load count per aisle per time unit, which is sufficient to evaluate the aisle load objective M.

For a batch B_i starting at time t, both subroutines operate only on its non-boundary aisles, i.e. $k \in \{2, \ldots, \beta_i - 1\}$. The boundary aisles a_{i1} and $a_{i\beta_i}$ represent the dummy startup and finishing steps of the batch and therefore do not contribute to the aisle load. The batch's processing sequence begins with a_{i1} at time t, and each subsequent aisle starts once its predecessor's interval has elapsed. For each remaining aisle a_{ik}, its processing interval therefore begins at $t + \sum_{j=1}^{k-1} p_{ij}$, and spans p_{ik} time units. Both subroutines iterate over every time unit τ within this interval.

The distinction between the two subroutines lies entirely in what is done at each (τ, a_{ik}) pair. $ExceedM$ (Algorithm 1) reads $aisle\_load(a_{ik}, \tau)$ and returns *True* as soon as any pair would cause the load to exceed M, leaving the matrix unchanged. $UpdateAisleLoad$ (Algorithm 2) increments $aisle\_load(a_{ik}, \tau)$ by 1 for every such pair, committing the assignment to the matrix. The two are therefore complementary: one checks whether an assignment is admissible, and the other commits it.

Algorithm 1: $ExceedM(t, B_i, aisle\_load(a,t), M)$
Check if scheduling a batch will result in exceeding the maximum aisle load M.

Input :
- Start time t
- Batch B_i (ordered set of β_i aisles a_{ik} and their processing times p_{ik})
- Current load matrix $aisle\_load(a,t)$ (aisle × time)
- Current maximum Aisle Load M

Output: True if scheduling batch B_i would exceed current M, False otherwise

Initialization:
$offset \leftarrow p_{i1}$;
for $k \in \{2, \ldots, \beta_i - 1\}$ **do**
  $start \leftarrow t + offset$;
  $end \leftarrow start + p_{ik}$;
  $current\_aisle \leftarrow a_{ik}$;
  for *each time* τ *in* $[start, end - 1]$ **do**
    if $aisle\_load(current\_aisle, \tau) + 1 > M$ **then**
      return *True*;
    end
  end
  $offset \leftarrow offset + p_{ik}$;
end
return *False*;

Algorithm 2: $UpdateAisleLoad(t, B_i, aisle\_load(a,t))$
Update the Aisle load once a new batch is scheduled.

Input :
- Start time t
- Batch B_i (ordered set of β_i aisles a_{ik} and their processing times p_{ik})
- Current aisle load matrix $aisle\_load(a,t)$ (aisle × time)

Output: Updated aisle load matrix $aisle\_load(a,t)$

Initialization:
$offset \leftarrow p_{i1}$;
for $k \in \{2, \ldots, \beta_i - 1\}$ **do**
  $start \leftarrow t + offset$;
  $end \leftarrow start + p_{ik}$;
  $current\_aisle \leftarrow a_{ik}$;
  for *each time* τ *in* $[start, end - 1]$ **do**
    $aisle\_load(current\_aisle, \tau) \leftarrow aisle\_load(current\_aisle, \tau) + 1$;
  end
  $offset \leftarrow offset + p_{ik}$;
end
return $aisle\_load(a,t)$;

Batch selection and scheduling logic
Algorithm 3 is the main scheduling procedure of this heuristic. It maintains a list of unscheduled batches ordered by ascending LST_i, ensuring the most urgent batches are considered first. This reduces the risk of time-window violations. A batch B_i becomes eligible for scheduling once $t \geq EST_i$, mirroring the release time constraints of the ILP.

At each step, a dummy variable $j \leftarrow -1$ is initialized, indicating that no batch has yet been selected. The algorithm calls $ExceedM$ on the most urgent eligible batch B_i. If $ExceedM$ returns $False$ then B_i is scheduled immediately. If $ExceedM$ returns $True$, the algorithm examines up to N subsequent eligible batches in ascending LST_i order, setting j to the index of the first candidate that does not increase M. If no such candidate is found, j is set back to i, B_i is scheduled, and M is incremented by 1. A batch is only assigned when $j > 0$, so the dummy variable ensures no assignment is made unless a valid batch has been selected. After each assignment, $UpdateAisleLoad$ updates $aisle\_load(a, t)$ to reflect the new assignment, maintaining a running record of aisle load that plays the same role as the constraints on r_{ikt} in the ILP.

The algorithm does not enforce $t \leq LST_i$ as a hard constraint. If all eligible batches would increase M and no look-ahead candidate avoids this, the most urgent batch is scheduled regardless, even if $t > LST_i$. Tardiness is therefore possible and should be considered as a performance metric.

Tardiness is usually measured as the completion time minus the due date. Since our model instead uses a time window in which a batch may start, obtained by subtracting the processing time from the due date, tardiness of a batch is measured here as the start time minus the latest start time:

$$Tardiness = \max_i(0,\ Starttime_i - LST_i) \tag{3.28}$$

When picker capacity is reached or no eligible batch exists, the algorithm advances the time t. If active batches are present, t increments by one unit. If no batches are active and none are yet eligible, t jumps directly to $\min\{EST_i \mid B_i \in \text{unscheduled}\}$, skipping idle periods with no schedulable work. Any pickers freed during such a jump sit idle in the interim, which is unavoidable given the time-window constraints.

The look-ahead parameter N
The parameter N controls a direct trade-off between urgency and aisle load: a larger N gives the algorithm more room to avoid incrementing M at the cost of potentially delaying more time-critical batches.

Remark. *Setting $N = 0$ reduces Algorithm 3 to a pure LST_i-priority rule while respecting $EST_i \leq t$. It always assigns the most urgent eligible batch and increments M if that batch raises the aisle load, without checking for a less congesting alternative.*

Algorithm 3: $NeighborhoodNGreedy(B_i, w, T, N)$
Greedy heuristic with LST_i ordering and limited look-ahead of N batches

Input :

- Ordered list of unscheduled batches B_i (by Latest start time), each with:
  - Earliest and Latest start time EST_i, LST_i
  - Ordered subset $a_{ik} \in [A]_{ext}$ each with respective processing time p_{ik}
  - Total processing time P_i
- Number of pickers w
- Time limit T
- Neighborhood size N

Output:

- Starting time x_{it} for each batch B_i
- Maximum Aisle load M

Initialization:
$M \leftarrow 0$;
$t \leftarrow 0$;
$active\_pickers(t) \leftarrow 0 \; \forall t$;
$aisle\_load(a,t) \leftarrow 0 \; \forall a, t$;
$x_{it} \leftarrow 0 \; \forall i, t$;
while *unscheduled is non-empty* **do**
  $j \leftarrow -1$;
  if $active\_pickers(t) = w$ **then**
    $t \leftarrow t + 1$;
    if $t > T$ **then**
      return *Not all batches could be scheduled within the given time limit;*
    end
  else
    if *there is no batch* $B_i \in$ *unscheduled such that* $EST_i \leq t$ **then**
      $t \leftarrow \min\{EST_i \mid B_i \in unscheduled\}$;
    else
      Take the first $B_i \in$ *unscheduled* such that $EST_i \leq t$;
      if $ExceedM(t, B_i, aisle\_load(a,t), M)$ *returns False* **then**
        $j \leftarrow i$
      else
        C = Number of batches $B_u \in unscheduled$ such that $EST_u \leq t$;
        $N' = \min\{N, C\}$;
        for l *in the next* N'*, where* $B_l \in$ *unscheduled and* $EST_l \leq t$ **do**
          if $ExceedM(t, B_l, aisle\_load(a,t), M)$ *returns* $False$ **then**
            $j \leftarrow l$;
            break
          end
        end
        if $j < 0$ **then**
          $j \leftarrow i$;
          $M = M + 1$;
        end
      end
      if $j > 0$ **then**
        Assign $x_{jt} = 1$;
        $aisle\_load(a,t) \leftarrow UpdateAisleLoad(t, B_j, aisle\_load(a,t))$;
        for $\tau \in [t, t + P_j - 1]$ **do**
          $active\_pickers(\tau) \leftarrow active\_pickers(\tau) + 1$;
        end
        Remove B_j from *unscheduled*;
      end
    end
  end
end
return x_{it}, M;

3.3.2. Greedy heuristic with fixed maximum aisle load

While the previous heuristic treats M as a quantity to be minimized during scheduling, it is also useful to consider what schedules are achievable under a prescribed maximum aisle load limit. The following heuristic takes M as a fixed input and constructs a feasible schedule subject to this constraint, providing a complementary perspective on the trade-off between aisle load and schedule efficiency.

Batch selection and scheduling logic

This heuristic follows the same incremental scheduling structure as Algorithm 3, processing batches in ascending LST_i order and using Algorithm 1 to validate each scheduling decision. The key distinction is that M is treated as a fixed input rather than a value to be minimized: no batch is ever scheduled if doing so would cause any aisle load to exceed M, and M is never incremented during the procedure.

If a batch cannot be scheduled at the current time t without violating the aisle load constraint, it is skipped and the algorithm moves to the next eligible batch. Unlike the neighborhood search in Algorithm 3, there is no fallback that forces a congesting batch through: if no eligible batch can be scheduled without exceeding M, the algorithm simply advances time. This strictness means that not all batches are guaranteed to start within their prescribed time window, and the resulting tardiness directly reflects how restrictive the chosen value of M is.

Algorithm structure and interaction

Algorithm 4 uses Algorithm 1 to check if the given M is exceeded. Time advancement follows three cases, depending on the current state. If the picker limit is reached, no new batch can start regardless of M, so the algorithm jumps directly to the earliest completion time among active batches, the next moment at which a picker becomes available. If pickers are available but no eligible batch can be scheduled without exceeding M, there is no known future moment at which this will change, so time is advanced by a single unit and the check is repeated. If no batches are active, the algorithm jumps directly to the earliest EST_i among unscheduled batches, the next moment at which scheduling becomes possible at all. Together, these three cases ensure that no moment at which a batch could have been scheduled is skipped.

Algorithm 4: $MaxMGreedy(B_i, w, T, M)$
Batch Scheduling with Picker and Maximum Aisle Load Constraints and processing times

Input :
- Ordered list of unscheduled batches B_i (by Latest start time), each with:
  - Earliest start time EST_i
  - Latest start time LST_i
  - Ordered subset $a_{ik} \in [A]_{\text{ext}}$ each with respective processing time p_{ik}
  - Total processing time P_i
- Number of pickers w
- Time limit T
- Maximum allowed aisle load M

Output:
- Starting time x_{it} for each batch B_i

Initialization:
$t \leftarrow 0$;
$active\_batches \leftarrow$ empty list;
$aisle\_load(a,t) \leftarrow 0 \ \forall a,t$;
$x_{it} \leftarrow 0 \ \forall i,t$;
$batch\_end(B_i) \leftarrow 0 \ \forall i$;
while *unscheduled is non-empty* **do**
  Remove all B_i from *active_batches* with $batch\_end(B_i) \leq t$;
  for B_i *in unscheduled, where* $EST_i \leq t$ **do**
    if $ExceedM(t, B_i, aisle\_load(a,t), M)$ *returns False* **then**
      $x_{it} \leftarrow 1$;
      Add B_i to *active_batches* and remove B_i from *unscheduled*;
      $aisle\_load(a,t) \leftarrow UpdateAisleload(t, B_i, aisle\_load(a,t))$;
      $batch\_end(B_i) \leftarrow t + P_i$;
    end
    if *active batches* $= w$ **then**
      break
    end
  end
  if $|active\_batches| = w$ **then**
    $t \leftarrow \min_i\{batch\_end(B_i) \mid B_i \in active\_batches\}$;
  else
    if *active_batches is non-empty* **then**
      $t \leftarrow t + 1$;
    else
      $t \leftarrow \min_i \{EST_i \mid B_i \in unscheduled\}$;
    end
  end
  if $t > T + \max(P_i)$ *for all* B_i *in unscheduled* **then**
    return *Not all batches could be scheduled within the given time limit;*
  end
end
return x_{it};

3.4. Evaluation Approach

The three methods introduced in this chapter, the ILP, the NeighborhoodN Greedy (Algorithm 3), and the MaxM Greedy (Algorithm 4), are evaluated on real warehouse data from two warehouses over eight representative days. The full dataset and its construction are described in Chapter 4.

The primary output metric for all three methods is M, the maximum aisle multiplicity achieved over the full planning horizon. Since congestion is the central concern of all three methods, M provides the most direct basis for comparison. Within the warehouse, an aisle load of 6 or more, corresponding to 6 or more carts with 5 load carriers in the same aisle, is taken as the threshold for the aisle load.

Two secondary performance measures are also reported: makespan, defined as the latest completion time across all batches, and total tardiness, defined as the total time by which batches start after their latest start time. These capture the operational cost of reducing aisle load and give insight into the tradeoff between aisle load control and schedule efficiency. Computational runtime is reported as a fourth indicator to assess practical applicability.

Results are presented for the pickzone containing the non-perishable items, as it contains the most aisles and highest batch volumes across both warehouses, making it the most operationally relevant and the most demanding test case. Within this zone, two days are examined in detail: one of the busiest days in the dataset and an average day, chosen to illustrate how the methods behave under both high and average congestion pressure. Section 5.3 then extends the comparison across all eight days and both warehouses.

The evaluation focuses on the afternoon picking window, defined as the period starting at 07:00 and ending once the last batch is scheduled on the delivery date. This choice is elaborated on in Section 4.1. The time limit T used as input to the models is set large enough that it is never binding in practice, (i.e. $T = \max_i(LST_i) + \sum_i P_i$, which bounds the time needed to schedule every batch sequentially after the last LST).

Two benchmarks are used as reference points. The first is the realized aisle load, measured from the individual item-pick timestamps available in the warehouse data. This reflects the current operational practice, where batch assignment follows the existing rule of sequencing orders by latest start time without explicitly considering aisle load. The second benchmark is obtained by setting $N = 0$ in the NeighborhoodN Greedy algorithm, which effectively removes the aisle load-aware component and assigns batches solely based on their latest start times while still respecting EST constraints. This mirrors the current assignment method and approximates a purely due-date-driven scheduling policy. Comparing it against the aisle load-aware methods isolates the contribution of aisle load-aware assignment decisions.

The comparison proceeds as follows. The ILP provides the optimal value of M for each problem instance, allowing the solution quality of the heuristic approaches to be quantified by measuring their deviation from the optimum. The NeighborhoodN Greedy is evaluated by comparing its achieved aisle load levels and runtime against both the ILP and the benchmark methods. For the MaxM Greedy, results are reported across a range of target values for M, enabling direct analysis of the corresponding makespan and tardiness costs for different aisle load targets. Table 3.4 summarizes the performance indicators reported for each method.

Method	M	**Tardiness**	**Makespan**	**Runtime**
Realized data	✓			
ILP	✓	✓	✓	✓
NeighborhoodN Greedy ($N = 0$)	✓	✓	✓	✓
NeighborhoodN Greedy ($N = 1000$)	✓	✓	✓	✓
MaxM Greedy	✓	✓	✓	✓

Table 3.4: Performance indicators reported for each method. A check mark indicates that the corresponding metric is reported for that method. The realized data serves as a baseline for M only, since no schedule is produced.

Chapter 4 describes how this data is gathered and processed. Section 4.4.1 summarizes the resulting dataset and defines the framework used to ensure a valid and realistic comparison of the three methods.

4 Data gathering and processing

Following the input data requirements identified in Sections 3.2 and 3.3, operational data were collected and transformed into the input parameters required by the optimization models. The resulting dataset combines warehouse characteristics, pick order information, and processing time estimates, directly providing the input parameters $n, w, A, B_i, p_{ik}, EST_i, LST_i$ required by the extended ALBP model. All data originate from internal warehouse management systems and cover multiple warehouses over a five-year period. Since the raw data are recorded at the level of individual item picks, several preprocessing steps were needed to construct pick order level and aisle level variables. To test the models, we apply them to data from two different warehouses labelled as Warehouse 1 and 2. The visualizations presented in this chapter are based on data from a busy day at Warehouse 2.

Section 4.1 describes the warehouse and pickzone structure and the selection of representative days. Section 4.2 discusses the pick order data and the estimation of the picker count w. Section 4.3 presents the norm time estimation methodology, covering aisle processing times as well as startup and wrap-up durations. Section 4.4 describes how the realized aisle load is extracted from the data. Section 4.5 validates the resulting dataset before it is used in Chapter 5.

4.1. Warehouse Data

Warehouses are partitioned into pickzones according to product characteristics. Perishable products are stored in temperature-controlled environments, while non-perishable products are stored under ambient conditions. No pick order spans multiple pickzones of the warehouse. All temperature-controlled items are picked in a physically separated hall maintained at approximately 2°C. Within this hall, two smaller sub-halls operate at temperatures of 14°C and 0°C, respectively. The sub-hall maintained at 0°C is excluded, since items there are picked using crates stacked on a pushable cart rather than load carriers, a process incompatible with the picker-to-parts system assumed in the model. This leaves three pickzones, shown in Table 4.1 together with the number of aisles they contain in each warehouse.

Pickzone	Number of aisles Warehouse 1	Number of aisles Warehouse 2
Non-perishable	24	34
2°C	19	14
14°C	7	6

Table 4.1: Pickzones and their respective number of aisles for Warehouse 1 and 2

Since pick orders never span multiple warehouse pickzones, the Aisle Load Balancing Problem can be solved independently for each zone.

As stated in Section 3.4, the non-perishable pickzone serves as the primary case throughout the evaluation, since it contains the most aisles and highest pick order volumes across both warehouses. From this point onward, the analysis is restricted to this pickzone: the pickzones containing the perishable items are not considered further, and all data, methods, and results discussed in the remainder of this chapter and in Chapter 5 pertain exclusively to the non-perishable pickzone.

The number of pick orders processed varies by day, with Fridays typically representing peak workload. To evaluate the proposed methods under different workload conditions, eight representative calendar days from 2025 were selected, each evaluated independently in both warehouses, giving sixteen day-warehouse instances in total. The year 2025 was chosen because it is the most recent year for which complete and consistent operational data are available.

The eight days are drawn from four separate weeks, with two days sampled per week: a Friday, representing peak workload, and one additional weekday earlier in that same week (Tuesday, Wednesday, or Thursday), representing a more average day. Pairing days within the same week, rather than sampling busy and average days from different weeks, avoids seasonal or promotional effects confounding the busy-versus-average comparison.

A "day" refers to the delivery date of the orders to the stores. Picking takes place in two shifts: the first typically runs from around 23:00 the day before to 07:00, and the second runs from around 07:00 to 15:00 of the specified day. In the realized data, we observe some flexibility around these boundaries.

For testing our model we restrict ourselves to the afternoon pick-window starting at 7:00. This window is chosen for two reasons. First, the number of active pickers during the afternoon picking window is approximately constant as seen in Section 4.2.1, which aligns with the constant picker count assumption underlying the model. Second, the afternoon window contains the majority of pick orders and represents the operationally most demanding part of the day.

Day 1 in Warehouse 1 corresponds to Day 9 in Warehouse 2, as both refer to the same calendar date. The same correspondence holds throughout the dataset: Day 2 aligns with Day 10, Day 3 with Day 11, and so forth. Among these days, Day 2 in Warehouse 1 and Day 10 in Warehouse 2 represent the busiest days for their respective warehouses. Day 10 together with one average day, day 13, are analyzed in detail in Chapter 5. Later the comparison is extended to all eight days and both warehouses in Section 5.3. The total number of pick orders per day and warehouse, for the non-perishable pickzone during the afternoon picking window, is shown in Tables 4.2 and 4.3.

Day	Number of pick orders	Weekday
1	568	Thursday
2	590	Friday
3	157	Tuesday
4	439	Friday
5	278	Wednesday
6	427	Friday
7	381	Wednesday
8	360	Friday

Table 4.2: Selected days for Warehouse 1: total pick orders and weekday. Days 1–8 correspond to the same calendar dates as Days 9–16 in Warehouse 2.

Day	Number of pick orders	Weekday
9	693	Thursday
10	753	Friday
11	430	Tuesday
12	660	Friday
13	469	Wednesday
14	605	Friday
15	569	Wednesday
16	430	Friday

Table 4.3: Selected days for Warehouse 2: total pick orders and weekday. Days 9–16 correspond to the same calendar dates as Days 1–8 in Warehouse 1.

4.2. Pick Order Data

The number of pickers assigned to pick orders is primarily determined by the projected number of pick orders to be processed. This volume varies with both the day and the size of the warehouse. In addition, several operational parameters associated with pick orders, such as earliest and latest feasible start times, depend partly on warehouse characteristics.

Pick orders differ substantially in composition. For example, some pick orders consist exclusively of load carriers containing crates of beverages, which are picked using automated systems in certain warehouses. Such pick orders are excluded from the present study. The final dataset therefore consists solely of manually processed pick orders that traverse warehouse aisles, each forming a batch B_i in the model, executed in zones where congestion is most likely to occur and where it can potentially be mitigated through improved scheduling.

Pickers may switch tasks during a shift, for example moving from order picking to forklift operation after completing a pick order. As a result, the number of active pickers on the floor is not constant throughout the day, which complicates modelling workforce availability. For simplicity, the model assumes a constant picker count. Accommodating for varying number of pickers is discussed in Section 6.2.

4.2.1. Estimation of w

To estimate w, the number of concurrently active pick orders is computed at 1-minute intervals, where a pick order is considered active from its recorded start time until its completion. For the visualization shown in Figure 4.1 this count is plotted using 10-minute intervals, revealing how picker activity fluctuates throughout the day.

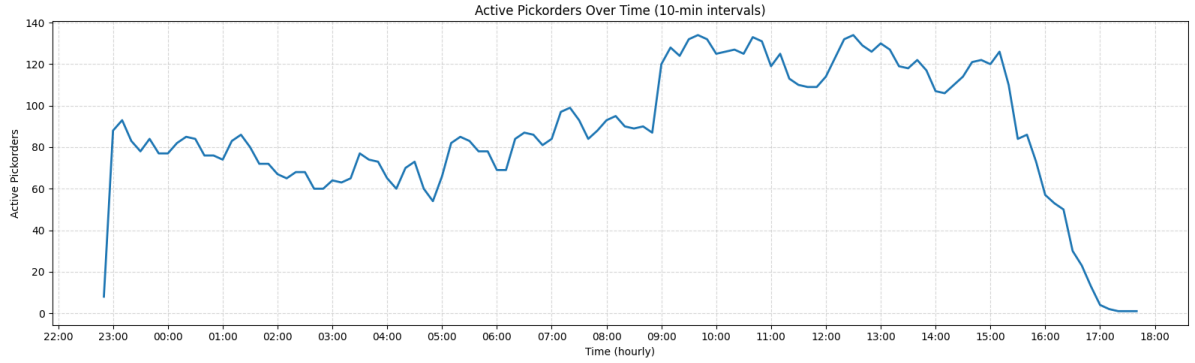


Figure 4.1: Number of active pick orders plotted over time for a given day including all pickzones.

The value of w chosen as input for the different models is taken as the median number of distinct active pickers per 1-minute interval during the afternoon picking window, from this window the start up and wrap up periods at the start and end of the shift are excluded. This gives an estimate that reflects the typical picking intensity without being skewed by the busy start or quiet end of the shift.

Different methods were tested to estimate w, including the mean of the specified time window and the minimum of the maximum of a rolling time window. All of these are reasonable choices, and testing showed they produced similar results; the median was ultimately selected. The number of pickers on all days for both warehouses can be observed in Tables 4.4 and 4.5.

Day	Number of pickers Warehouse 1
1	60
2	65
3	44
4	49
5	42
6	59
7	41
8	36

Table 4.4: Number of pickers for the selected days in Warehouse 1.

Day	Number of pickers Warehouse 2
9	70
10	80
11	51
12	61
13	53
14	66
15	47
16	45

Table 4.5: Number of pickers for the selected days in Warehouse 2.

4.3. Norm Time Estimation

For every pick order, an estimate of its processing time is available. The total processing time is decomposed into four components: the startup time, the picking time, the movement time and the wrap-up time. We first consider the picking and movement time which determine the time spent in the physical aisles.

4.3.1. Picking and movement norm time

The picking time estimate, denoted by $Pick\_time(i)$, represents the time required to pick items and place them on the load carrier. The movement time estimate, $Move\_time(i)$, captures the time spent traveling between pick locations.

For each pick order, the number of aisles visited, $Nbr\_aisles(i)$, and the total number of item picks, $TotalNbr\_picks(i)$, are known. In addition, the number of picks performed in aisle k for pick order

i is given by $Nbr\_picks\_aisle(i,k)$. Using this information, a norm time per aisle is constructed to approximate the time a pick order spends in each individual aisle, yielding the aisle processing time p_{ik} required as input by the extended ALBP formulation. The norm time per aisle is defined as

$$Normtime\_per\_aisle(i,k) = \frac{Move\_time(i)}{Nbr\_aisles(i)} + \frac{Nbr\_picks\_aisle(i,k)}{TotalNbr\_picks(i)} \cdot Pick\_time(i). \tag{4.1}$$

The movement time is assumed to be independent of the number of picks within a particular aisle and is therefore evenly distributed across all aisles visited by the pick order. In contrast, the picking time is assumed to scale linearly with the number of item picks and is therefore proportional to the share of picks performed in each aisle.

Ideally, the norm time per aisle would be derived from a detailed decomposition of all elementary actions performed along the pick route, such as aisle entry, trolley positioning, item handling, and aisle exit. In practice, however, such fine-grained operational data are not available. The proposed formulation is therefore a practical approximation that captures the main drivers of aisle-level workload, while staying within the limits of the available data.

Figure 4.2 compares the estimated aisle norm times with the actual processing times. The scatter plot shows a strong correspondence between the two measures, indicating that the norm time provides a sufficiently accurate estimator of the realized aisle-level processing time.

While most observations are clustered around the identity line, a subset of pick orders exhibits realized aisle times that are a factor of two or more larger than the corresponding norm time. These deviations may be attributable to practical operational disruptions such as delayed stock replenishment, equipment malfunctions, or congestion itself. For the majority of observations, the difference between norm time and realized time falls between -3 and 1 minutes (first and third quartile), indicating that the norm time is a close approximation for most pick orders.

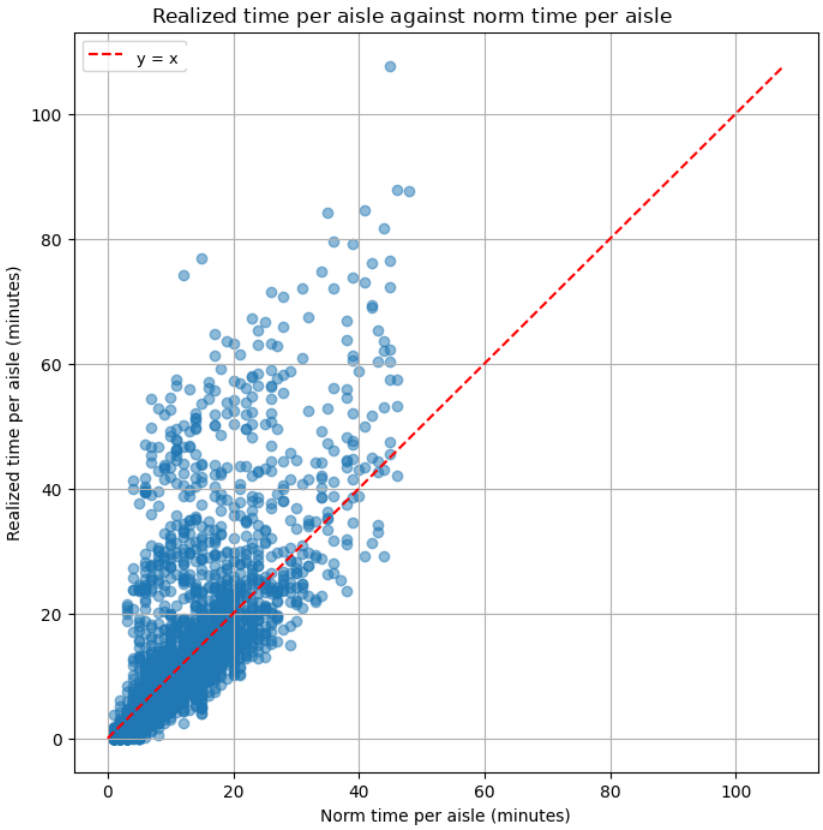


Figure 4.2: Comparison of Aisle norm time and Aisle realized time scatter plots.

4.3.2. Startup and wrap-up norm time

Norm times for the startup and wrap-up steps of a pick order are available in the dataset, represented as aisle −1 (startup) and aisle 1000 (wrap-up), consistent with the dummy aisle convention introduced in Section 3.2.2. The duration of these steps can vary: startup times depend on factors such as retrieving carts, the starting pick location, and printing labels. Wrap-up times similarly vary depending on where carts need to be placed after the pick order is completed.

4.3.3. Earliest and latest start time of a pick order

By operational design, each pick order consists of five load carriers, which may correspond to different shop orders as seen in Figure 1.1. These shop orders can be assigned to different outbound trucks, implying that the individual load carriers may be subject to different outbound deadlines. As a consequence, the earliest and latest feasible times at which the separate load carriers must be completed can differ within a single pick order.

To obtain start time window constraints at the pick order level, the earliest and latest feasible finish times of the load carriers are translated into an earliest and a latest start time, denoted by EST and LST, respectively. For each load carrier, this translation is performed by subtracting the estimated processing time of the pick order from its earliest and latest finish times. The pick order level earliest start time is then defined as the maximum of the resulting earliest start times of its load carriers, ensuring feasibility for all associated shop orders. Similarly, the latest start time is defined as the minimum of the latest feasible start times of these load carriers.

Formally, this aggregation guarantees that all load carriers belonging to a pick order can be processed within the selected time window. A visualization of the resulting EST and LST values for a single operational day is shown in Figure 4.3. In this figure, EST values are indicated in blue and LST values in orange.

A substantial fraction of pick orders arrives before 21:00. For many of these orders, processing is permitted immediately upon arrival. Throughout the remainder of the day, pick orders exhibit varying degrees of scheduling flexibility, as reflected by differences between their EST and LST values.

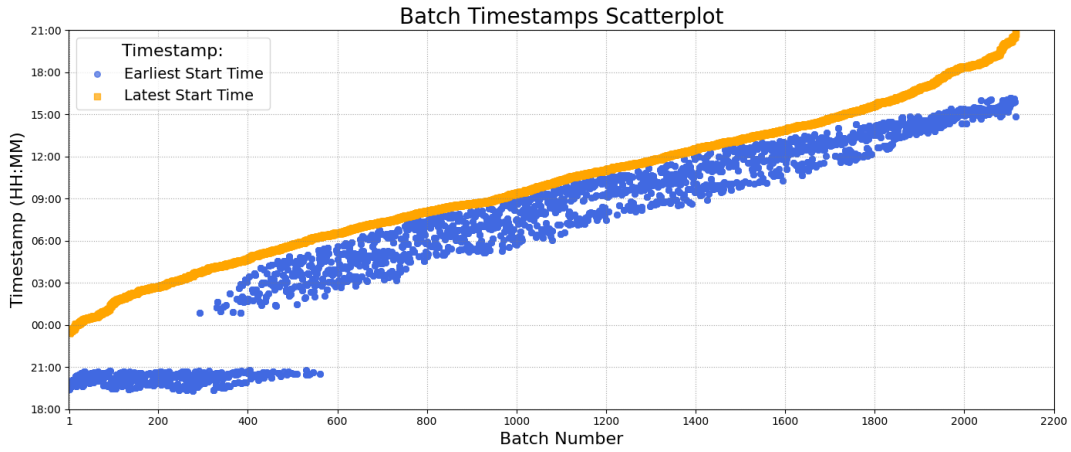


Figure 4.3: Scatter plot of EST and LST of all pick orders of Warehouse 2 on Day 10

The first and third quartiles of the window $LST - EST$ are located at approximately 75 and 200 minutes, respectively, indicating that many pick orders can be scheduled within a relatively wide time interval. A visualization of the difference is given in Figure 4.4. This available flexibility is a key input for the scheduling and batching methods developed in this study, as it enables the reassignment of pick orders in time without violating outbound constraints. An illustrative visualization of the constructed scheduling windows, together with the corresponding realized start times for a representative operational day, is provided in Appendix A.2.1.

Remark. *It is worth noting that, in practice, the EST is not strictly enforced. Historical data show that pick orders regularly start before their assigned EST, typically to keep pickers active and avoid idle time between orders. This is taken into account when interpreting the scheduling results.*

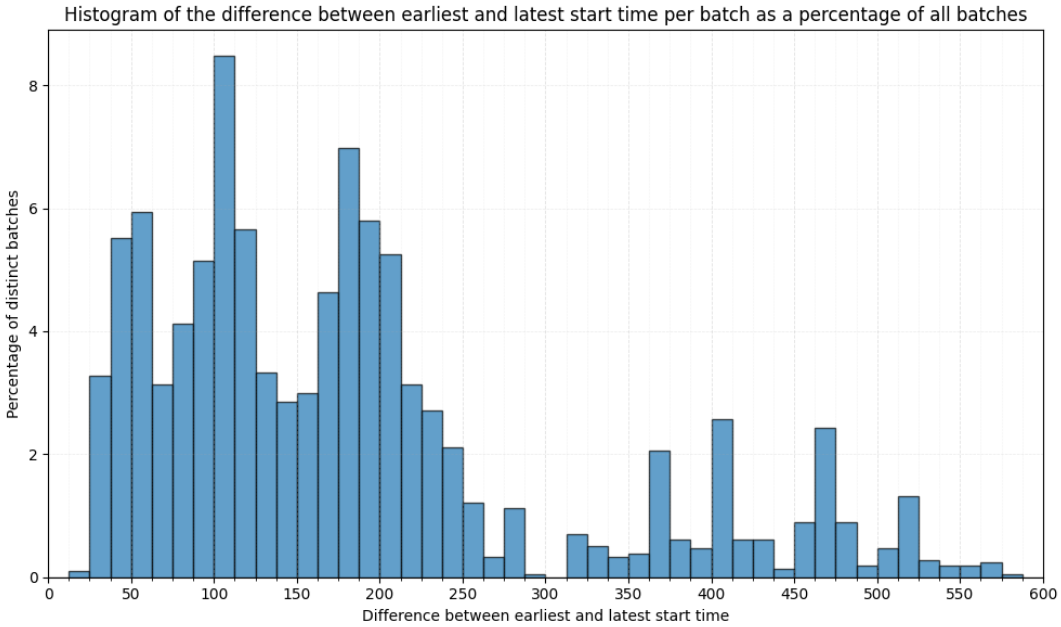


Figure 4.4: Difference of a Earliest start time and Latest start time of all pick orders on day 10 in Warehouse 2.

The resulting EST and LST values per pick order directly define the time window constraints in the extended ALBP formulation, as expressed in Constraint 3.21. These values together with spatial and processing-time information, constitute essential inputs for the aisle load balancing problem considered in this thesis. In Section 4.4.1, we provide an overview of the variables included in the final dataset, organized by item level, pick order level, and aisle level information.

To construct the input for time window specified in Section 4.2, only pick orders with a recorded start time after 07:00 are retained. For the pick orders that have a start time after 7:00 and an Earliest start time before 7:00 this start time is moved, this is done before discretization of time. The earliest start times that fall before 07:00 are changed as follows:

$$EST_i \leftarrow \max(EST_i,\ 07{:}00). \tag{4.2}$$

Since a pick order may have a realized start after 07:00 while its EST predates it, this ensures that no pick order is assigned before the start of the picking window. Latest start times are left unchanged: since every retained pick order starts after 07:00 and LST_i within our data used never precedes the realized start, LST_i is always after 07:00 as well, so no analogous clipping is needed here. This adjustment preserves all feasibility constraints while restricting the scheduling horizon to the period in which the constant picker assumption holds.

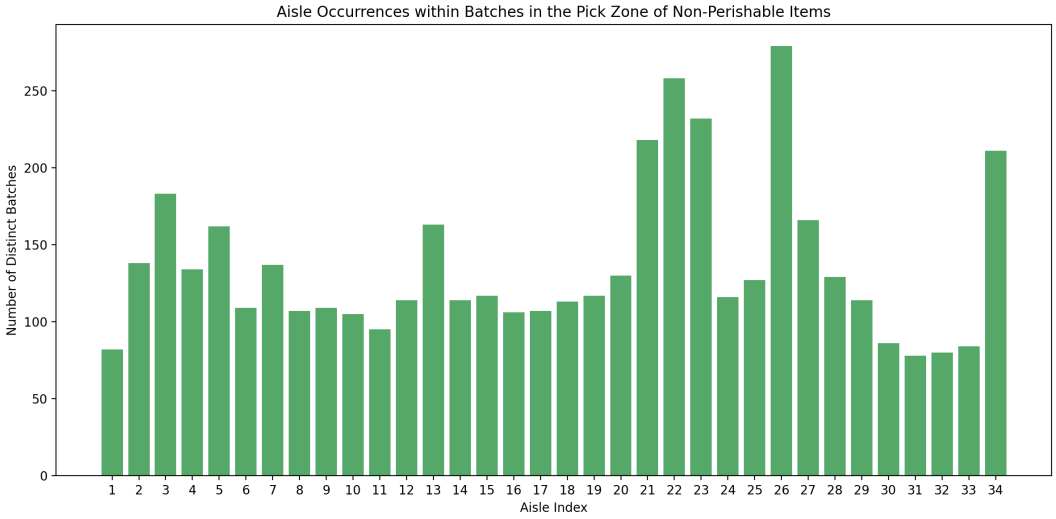
4.3.4. Time discretization

Following the adjustment made, clock time is mapped onto a discrete time index $t \in [T]_{\geq 0}$, with $t = 0$ corresponding to 07:00, the start of the afternoon picking window. One unit of t corresponds to one minute of clock time. Since EST_i and LST_i are by construction at or after 07:00, both are non-negative once expressed in t-units. All time-related parameters used throughout this thesis, including EST_i, LST_i, and S_{ik}, are expressed on this discretized scale, as is t itself in Algorithms 3 and 4.

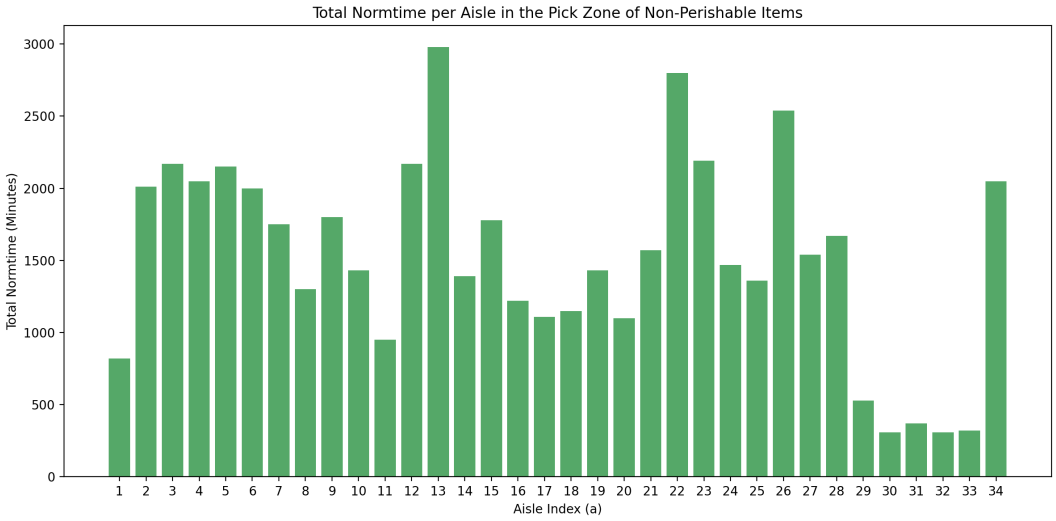
4.3.5. Aisle visit frequency across pick orders

Figure 4.5 shows the total accumulated norm time and visit frequency per aisle across all pick orders on day 10 of Warehouse 2, broken down by pickzone. Within each pickzone, both the visit frequency

and the sum of norm times are distributed unevenly across aisles, with some aisles accumulating substantially more norm time than others. This reflects differences in product demand across locations. It is worth keeping in mind that this uneven distribution of accumulated norm time, both across and within pickzones, may have implications for how the scheduling methods perform in practice, which is explored further in Chapter 5.



(a) Total multiplicity of the aisles



(b) Total norm time per aisle

Figure 4.5: Aisle multiplicity and total norm time of realized pick orders.

4.4. Realized Aisle Load

From the data at an individual item pick-level we can extract the realized aisle load for the days on which we test our model. This requires a start and end time per aisle for each pick order. The start time of an aisle is defined as the moment the first pick within that aisle is made.

For the end time, two definitions are possible: the time of the last pick within that aisle, or the time of the first pick in the next aisle. The latter is used here, so the end time of aisle k is set to the start time of aisle $k + 1$. For the last aisle in a pick order, the end time is always set to the time of the last pick within that aisle, since no next aisle exists. This choice assumes that a picker remains in aisle k until they make their first pick in aisle $k + 1$, which includes the time spent moving to the next aisle. While this may slightly overestimate the time a picker occupies a given aisle, it is a closer approximation of physical presence than cutting off at the last pick. It should be noted that when validating this choice by comparing it with the alternative definition, the resulting outcomes were found to be nearly identical.

With these start and end times, the number of pick orders simultaneously present in a given aisle at any point in time can be determined by counting those that have started processing the aisle but have not yet finished. Figure 4.6 shows how the maximum number of active pick orders in any aisle changes throughout the picking day. At each time point, it displays the largest number of pick orders active in a single aisle. The pattern is similar to what is seen in Figure 4.1, which is consistent with busier periods generally producing higher aisle loads.

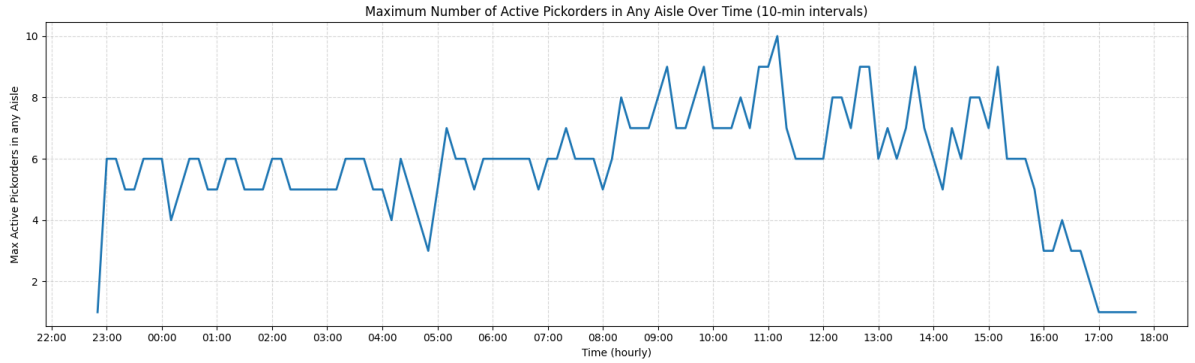


Figure 4.6: Maximum Aisle Load during a day of picking

While Figure 4.6 only reports the maximum load across all aisles, it is also important to examine how aisle load is distributed at the aisle level throughout the day. Therefore, in the comparison between the busy and average day, we also analyse aisle-level load over time. This is visualized in Section 5.2.3 using a heatmap that shows the number of pickers present in each aisle at each point in time. To ensure interpretability, aisle loads are capped at a value of 6, as aisle load levels beyond this threshold are assumed to lead to disproportionately higher operational disruption.

4.4.1. Final dataset

After the data gathering, preprocessing, aggregation, and filtering steps described in the previous subsections, a final dataset is constructed that serves as the sole input for models developed in this thesis. This dataset contains information at multiple levels of aggregation, reflecting the hierarchical structure of warehouse operations: warehouse-level, batch-level, aisle-level and individual item pick-level information. All variables included are directly observable or derived from operational data, and no additional assumptions on data availability are required.

Table 4.6 provides a structured overview of the variables included in the final dataset, grouped by aggregation level and annotated with their corresponding data types. The input highlighted in bold is the input that relates directly to the models. The other variables mentioned are mainly used for validation and measurements.

Warehouse-level input	Variable type	Model variable
Warehouse identifier	Identifier	
Date of picking	Date	
Number of batches	Integer	n
Number of pickers	Integer	w
Number of aisles	Integer	A
Batch-level input		
Batch identifier	Identifier	i
Pickzone	Categorical	
Earliest start time of B_i	Timestamp	EST_i
Latest start time of B_i	Timestamp	LST_i
Number of aisles visited in B_i	Integer	β_i
Batch movement norm time	Duration (minutes)	
Batch picking norm time	Duration (minutes)	
Batch total norm time	Duration (minutes)	P_i
Aisle-level input		
Aisle identifier	Integer	a_{ik}
Sequence index of aisle a in B_i	Integer	k
Norm time of aisle a	Duration (minutes)	p_{ik}
Individual item pick-level input		
Batch identifier	Identifier	i
Timestamp of item pick	Timestamp	
Pick method	Categorical	
Pick location within aisle	Number	

Table 4.6: Overview of all variables included in the final dataset used as input for the aisle load balancing models.

After constructing the final dataset, the operational information is translated into the model input parameters introduced in Chapter 3. The available workforce w is determined from the corresponding shift schedule, while the earliest feasible start times EST_i are derived for every batch based on the procedure described in Section 4.3.3. Together with the estimated processing times and aisle-level processing times, these parameters form the complete input required by the optimization models.

4.5. Validation

The preprocessing and estimation steps described in this chapter each introduce approximations or assumptions. As shown in Section 4.3.1, the norm time formulation closely approximates realized aisle-level processing times for the majority of observations, with deviations plausibly explained by operational disruptions outside the model's scope. The EST and LST derivation, discussed in Section 4.3.3, also relies on norm time estimates, and cases where $LST < EST$ are excluded to ensure all instances in the test set have a feasible schedule. In the data used to test the models, only two such cases across all instances were found. Beyond these specific checks, general consistency checks were carried out throughout the filtering and transformation pipeline to confirm no errors were introduced. Based on these steps, the processed dataset is considered a suitable input for the aisle load balancing models developed in this thesis.

With the dataset created and validated, we can move on to applying the models from Chapter 3 to it. Chapter 5 presents the results of the ILP, the NeighborhoodN Greedy and the MaxM Greedy on this dataset, first in detail for a busy and an average day in Warehouse 2, and afterwards summarized across all eight days and both warehouses.

5 Results

This chapter presents the outputs of the three methods introduced in Chapter 3, applied to the dataset described in Chapter 4. The primary comparison metric is M, the maximum aisle multiplicity, with makespan and total tardiness reported as secondary indicators of schedule efficiency. As established in Section 3.4, the pickzone containing the non-perishable products serves as the primary case throughout this chapter.

Two days are compared in detail throughout Sections 5.1 and 5.2: the busiest day in the dataset (Day 10) and an average day (Day 13), both from Warehouse 2, chosen to show how the methods behave under both high and low aisle load pressure.

The chapter is structured as follows. Section 5.1 establishes the realized aisle load as the starting point, showing the aisle load levels observed in the current warehouse operation before any scheduling model is applied. Section 5.2 compares the achieved M across all models on the busy and average day, including the active picker profiles and spatial heatmaps. Section 5.3 checks whether this pattern holds across the full set of sixteen days and both warehouses. Section 5.4 then examines what the reduction in M costs in terms of tardiness and makespan. Section 5.5 closes the chapter by comparing how long each model takes to produce a schedule.

5.1. Realized Aisle Load

Before evaluating any scheduling model, it is useful to establish what the current warehouse situation looks like without any aisle load-aware assignment. Figure 5.1 shows the maximum aisle load M throughout the busy and average day, measured from the realized pick order timestamps recorded in the warehouse system. Each point represents the maximum aisle load M during a 10-minute interval.

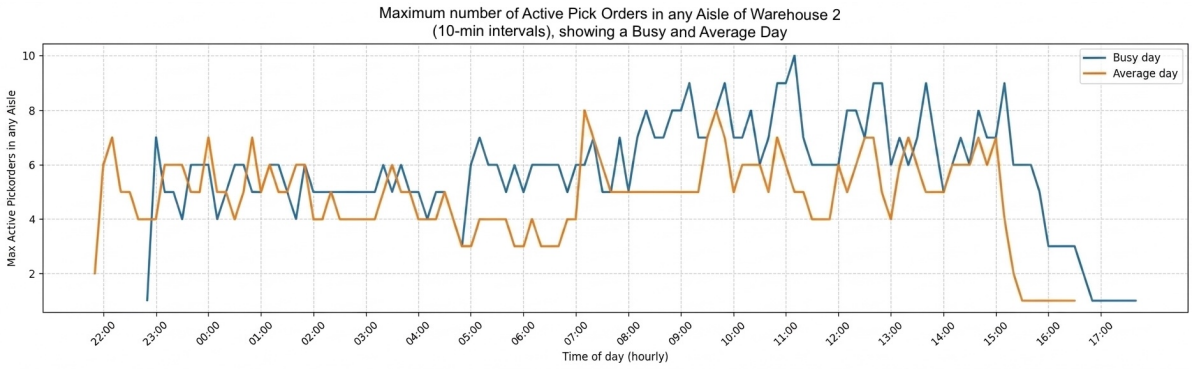


Figure 5.1: Realized maximum aisle load M on the busy and average day, measured in 10-minute intervals. Each point reflects the highest concurrent picker count observed in any single aisle.

On the busy day, M regularly reaches values between 6 and 10, with sustained periods of high load during the early picking process peak. The average day shows a similar pattern in shape but at lower absolute values, reflecting the reduced number of pick orders. Both days show that M frequently exceeds 6, the threshold above which aisle load is considered operationally problematic, motivating the need for a scheduling approach that actively controls aisle load. Figure 5.1 also shows that aisle load is consistently higher during the afternoon picking window than overnight, on both the busy and average day. As stated before, our further analysis will only focus on the afternoon picking window, therefore excluding the part from 22:00 to 7:00; the full reasoning behind this choice is discussed in Chapter 4.

An important observation during the data validation is that pick orders regularly start before their earliest start time (EST), as discussed in Section 4.3.3. This reflects the current practice in the warehouse,

where pickers are assigned work as soon as they become available regardless of the scheduled window. While this keeps pickers active, it means the EST constraint is not enforced in practice, resulting in a lower makespan than would occur under strict time window adherence.

The models, by contrast, uphold the EST, while the ILP also upholds the LST constraint: the goal is to assign batches within their time windows in such a way that it reduces aisle load while remaining operationally feasible. A limitation of this is that some pickers might end up with a bigger workload than others, depending on the availability of batches during scheduling.

5.2. Achieved Maximum Aisle Load across Models

This section compares the achieved M across all three methods introduced in Chapter 3: the ILP, the NeighborhoodN Greedy, and the MaxM Greedy. Throughout this section, the reported M value for MaxM Greedy is the lowest bound for which zero tardiness is still achieved on that day, as detailed in Section 5.4.3. The comparison is structured in three parts. First, the achieved M over time is compared directly across all models. Second, the active picker profiles are examined to explain the workload shape and the divergence in ILP behavior. Third, aisle load heatmaps provide a spatial view of how each model distributes pickers across the warehouse.

5.2.1. Maximum aisle load comparison

Figure 5.2 shows the maximum aisle load M over time on the average day. The ILP achieves the lowest M throughout the day, maintaining a controlled profile well below the other methods, though this comes with a considerably longer makespan, a point we return to in Section 5.4.2. The MaxM Greedy follows a profile that stays close to its input threshold of $M = 4$, showing that the hard upper bound is respected. The NeighborhoodN Greedy models with $N = 0$ and $N = 1000$ produce similar profiles to each other, with $N = 1000$ showing a modest reduction in peak M compared to $N = 0$, suggesting that neighborhood awareness provides some benefit even without a hard constraint on M.

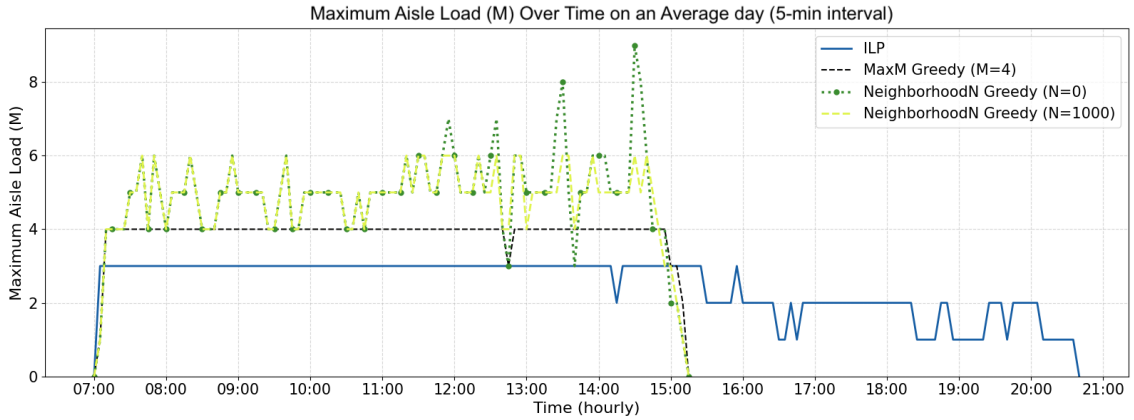


Figure 5.2: Maximum aisle load M over time on the average day, shown in 5-minute intervals for all models. The ILP achieves the lowest M throughout the day, with MaxM Greedy ($M = 4$) following closely below its input threshold. The NeighborhoodN Greedy ($N = 0$) reflects the current assignment practice and peaks at 9.

Figure 5.3 shows the same comparison on the busy day. The overall M values are higher across all models, consistent with the increased number of batches. The relative ordering of the models is preserved: the ILP achieves the lowest M, followed by MaxM Greedy, with both NeighborhoodN Greedy variants producing considerably higher values. On the average day the ILP peaks at $M = 3$ against a NeighborhoodN Greedy ($N = 0$) peak of $M = 9$, a gap of 6, while on the busy day the ILP peaks at $M = 4$ against a NeighborhoodN Greedy ($N = 0$) peak of $M = 10$. The absolute gap stays the same size at 6, but now sits at a higher absolute level on both sides, consistent with the higher workload. It is worth asking what it would cost the MaxM Greedy to match the ILP's aisle load directly. This is examined further in Section 5.4. The NeighborhoodN Greedy ($N = 0$) baseline frequently reaches M values between 6 and 9, reinforcing its role as the upper reference for the comparison. On the busy day, the NeighborhoodN Greedy ($N = 1000$) reaches a peak of $M = 9$, compared to $M = 6$ on the average

day, closing most of the gap it previously held over $N = 0$.

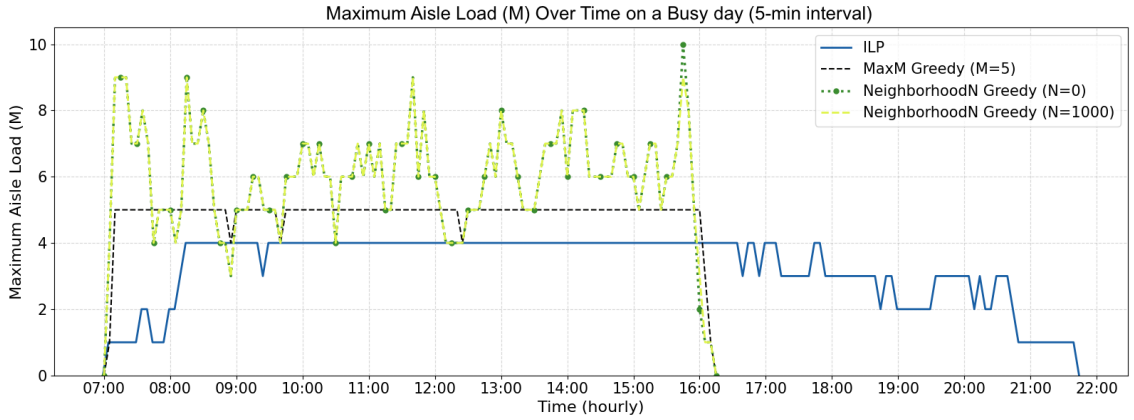


Figure 5.3: Maximum aisle load M over time on the busy day, shown in 5-minute intervals for all models. The absolute gap between the ILP and the greedy variants matches the average day but sits at a higher level, consistent with the higher workload. The NeighborhoodN Greedy ($N = 0$) reflects the current assignment practice and peaks at 9.

The NeighborhoodN Greedy with $N = 0$ should be closest to how the realized data behaves, since it mirrors the current assignment method. Comparing the NeighborhoodN Greedy ($N = 0$) curves in Figures 5.2 and 5.3 to the realized M in Figure 5.1, some similarities emerge in where M peaks throughout both days.

5.2.2. Active batches over time

To understand why the models produce different M profiles, it is useful to look at the total number of active batches over time. This reveals how each model distributes work across the planning horizon, which directly drives the aisle load patterns seen in the previous subsection.

As stated, the NeighborhoodN Greedy with $N = 0$ serves as the proxy for the current assignment practice in the warehouse, where each batch is assigned at its norm time without any aisle load awareness. It provides the upper reference for M and the most concentrated workload profile.

Figure 5.4 shows the total number of active batches over time on the average day. The NeighborhoodN Greedy ($N = 0$) curve closely follows the MaxM Greedy and NeighborhoodN Greedy ($N = 1000$) curves, indicating that all greedy variants assign batches at a similar rate throughout the day. The ILP shows a notably different profile, keeping pickers active later into the day. This behavior is a direct consequence of the ILP exploiting the full EST-LST window to spread assignments and reduce the maximum aisle load, and is discussed further in Section 5.4.

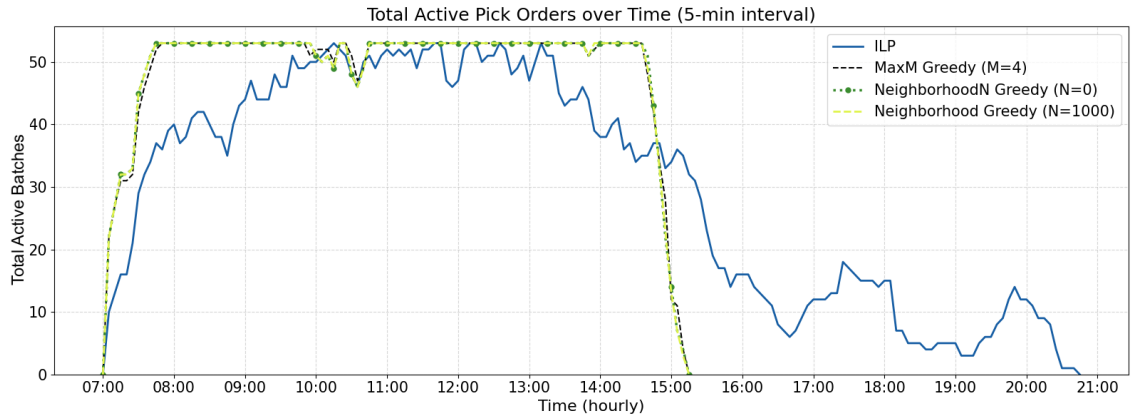


Figure 5.4: Total active batches over time on the average day, shown in 5-minute intervals for all models. The greedy variants follow a similar concentrated release pattern, while the ILP distributes batches more evenly across the available planning horizon.

Figure 5.5 shows the same metric on the busy day. The greedy variants again follow closely, though the total number of active batches is considerably higher, peaking around 80. The ILP profile diverges even more than on the average day.

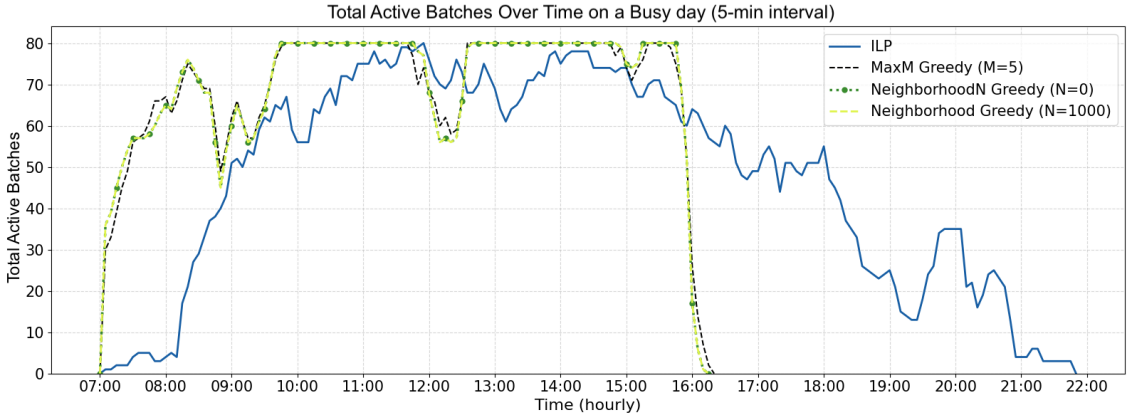


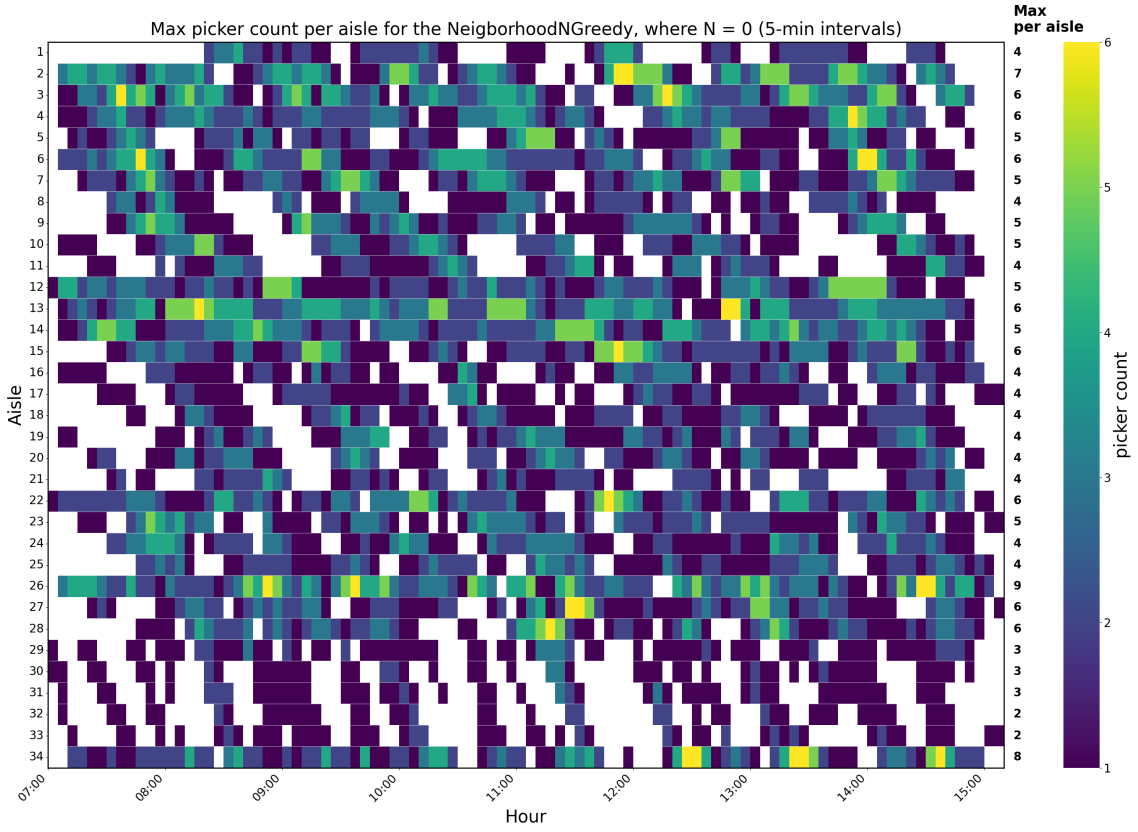
Figure 5.5: Total active batches over time on the busy day, shown in 5-minute intervals for all models. The divergence between the ILP and the greedy variants is more pronounced than on the average day, consistent with the wider scheduling windows available when more batches are present.

5.2.3. Spatial distribution of aisle load

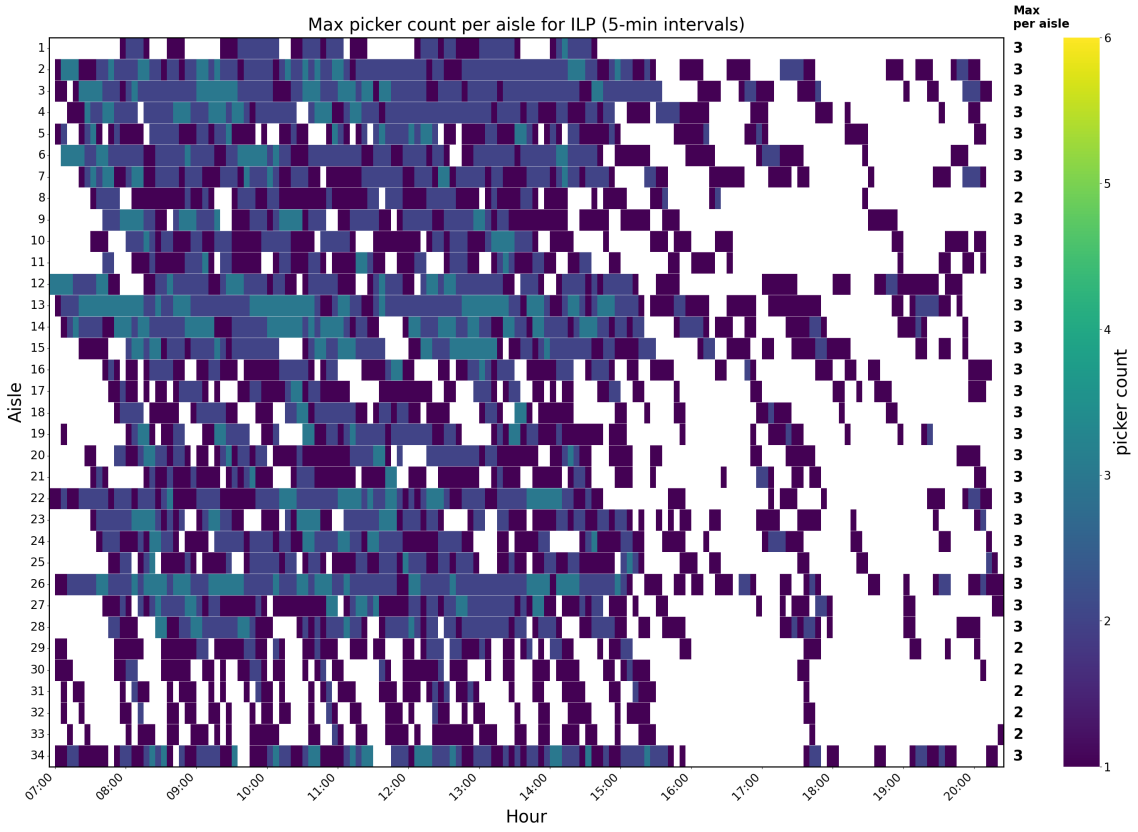
The aggregate M comparison shows how aisle load evolves over time, but does not reveal where in the warehouse aisle load occurs. The heatmaps below show the maximum picker count per aisle in 5-minute intervals, giving a spatial view of how each model distributes load across aisles throughout the day. The colour scale is capped at $M = 6$, reflecting the operational threshold above which aisle load is considered problematic; all values at or above 6 share the same colour. The maximum achieved per aisle is still indicated numerically on the right-hand side of each heatmap.

Figures 5.6 and 5.7 show the aisle load heatmaps for the average day. The NeighborhoodN Greedy ($N = 0$) heatmap shows a scattered distribution of high-load moments across many aisles and time periods, with the right-hand labels confirming that several aisles reach M values well above 6. The ILP heatmap shows a markedly different picture: the per-aisle maximum is 3 for nearly every aisle in the warehouse, half the operational threshold of 6, and high-load cells are correspondingly far less frequent than in the NeighborhoodN Greedy heatmap. The NeighborhoodN Greedy ($N = 1000$) heatmap is visually similar to $N = 0$, though the per-aisle maxima are slightly lower on several aisles, consistent with the modest M reduction seen in Figure 5.2. The MaxM Greedy heatmap reflects the hard bound directly, with cells not exceeding the threshold of 4 and the load spread more evenly across time. All models show a diagonal stripe pattern in the heatmap, this happens because pickers move from one aisle to the next as the day goes on.

Figures 5.8 and 5.9 show the heatmaps for the busy day. The higher workload results in more consistently saturated aisles across all models, with the colour scale reaching its ceiling more frequently and across more aisles. The contrast between the greedy variants and the ILP is sharper here than on the average day: the ILP produces a noticeably more structured distribution of load, with fewer aisles reaching the threshold and a clearer separation between active and idle periods. The MaxM heatmap again shows a clean suppression of cells above the threshold, though the remaining load is more densely packed than on the average day given the higher number of active pickers and batches to assign. The same diagonal stripe pattern shows up again on the busy day, and it stands out clearly in the aisles that are visited less often.

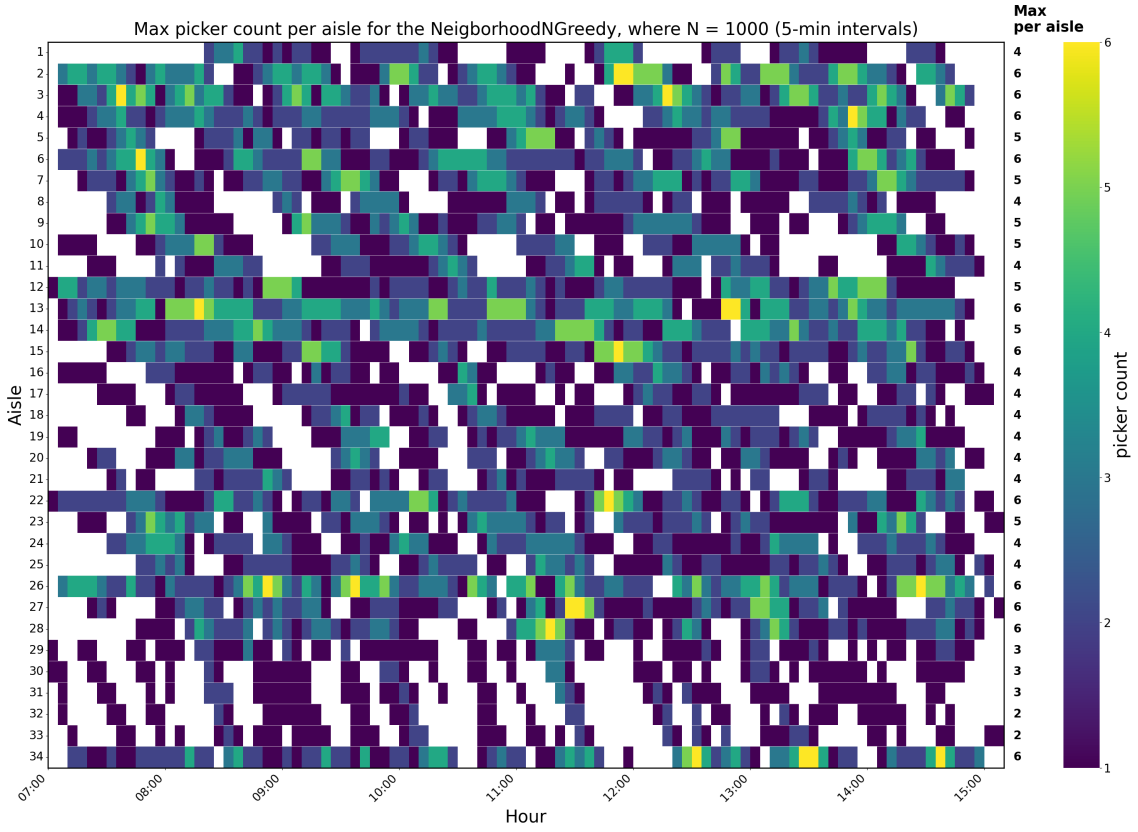


(a) NeighborhoodN Greedy ($N = 0$), realizing $M = 9$

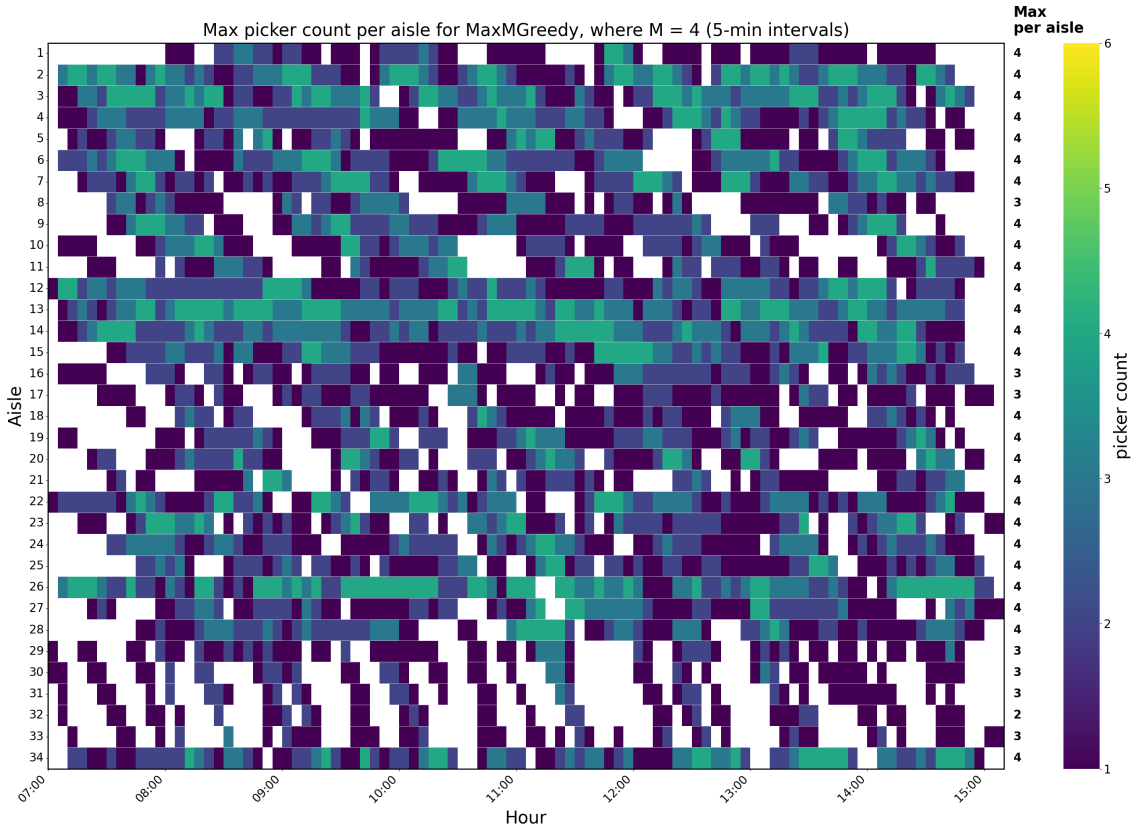


(b) ILP, realizing $M = 3$

Figure 5.6: Maximum picker count per aisle in 5-minute intervals on the average day, shown for NeighborhoodN Greedy ($N = 0$) and the ILP. The colour scale is capped at $M = 6$; the maximum achieved per aisle is indicated numerically on the right. Lighter colours indicate higher aisle load.

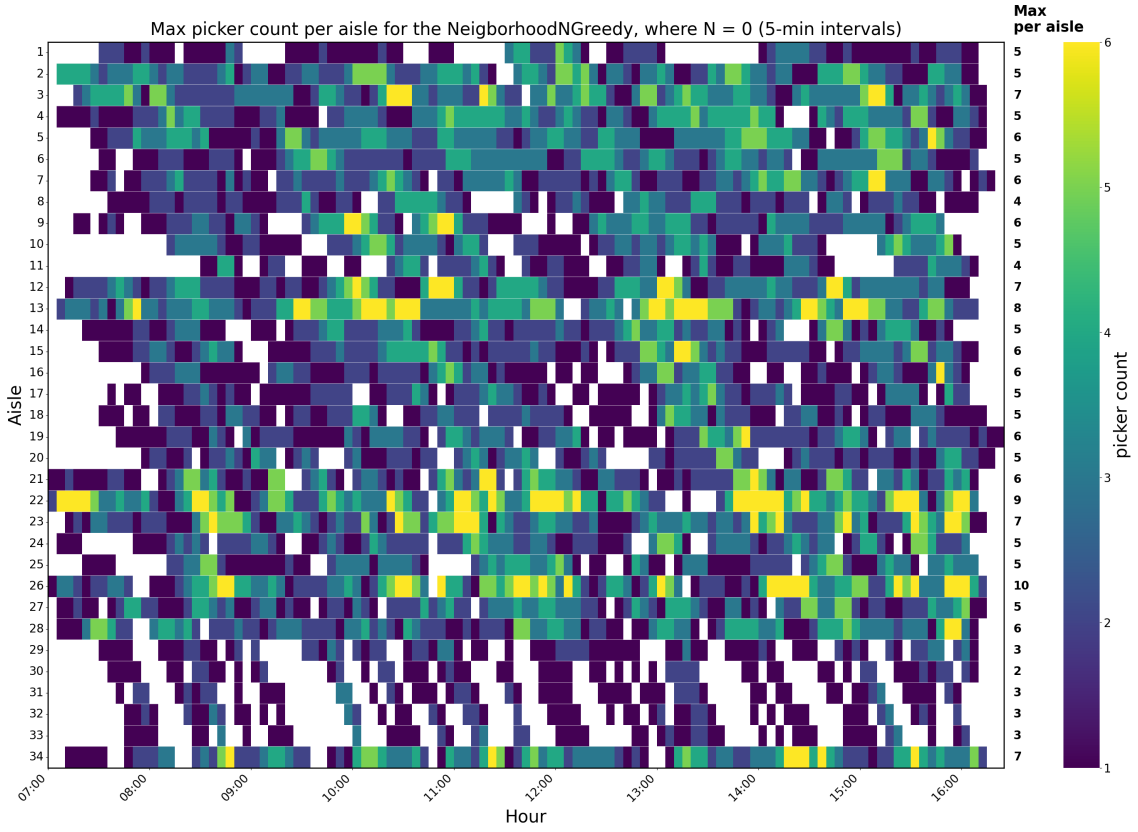


(a) NeighborhoodN Greedy ($N = 1000$), realizing $M = 6$

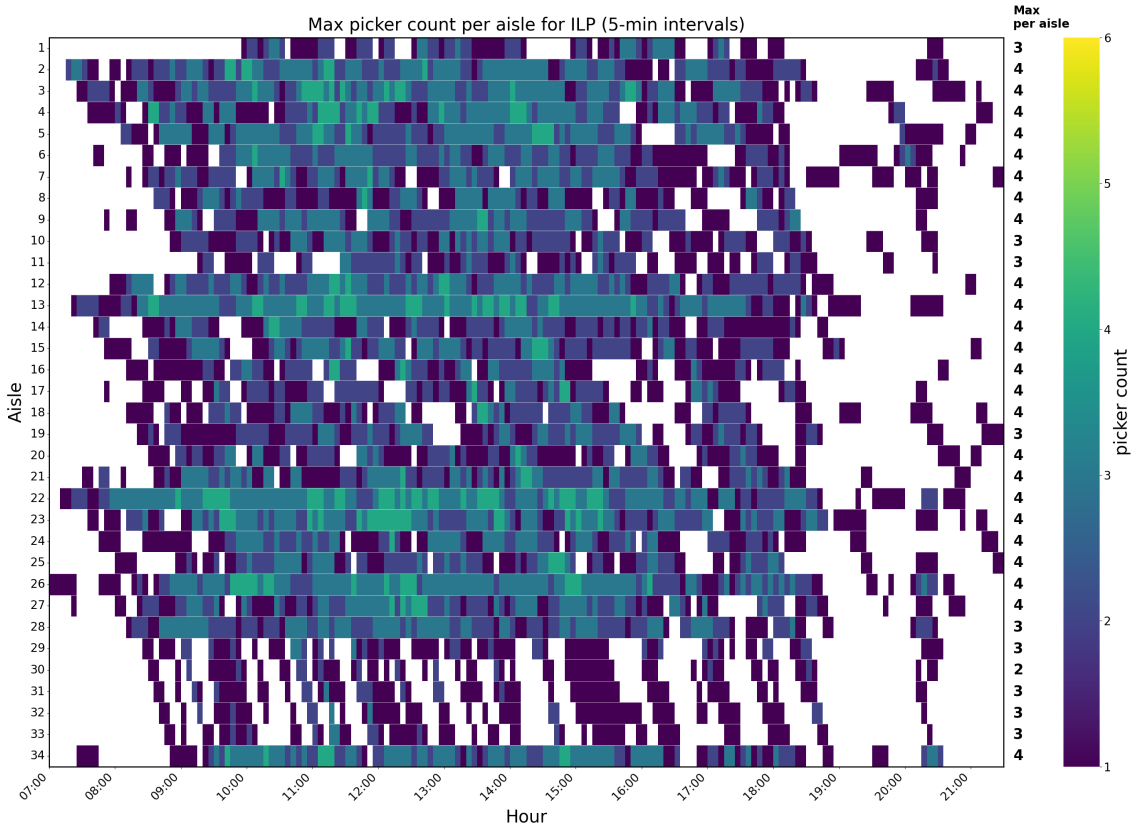


(b) MaxM Greedy ($M = 4$), realizing $M = 4$

Figure 5.7: Maximum picker count per aisle in 5-minute intervals on the average day, shown for NeighborhoodN Greedy ($N = 1000$) and MaxM Greedy. The colour scale is capped at $M = 6$; the maximum achieved per aisle is indicated numerically on the right. Lighter colours indicate higher aisle load.

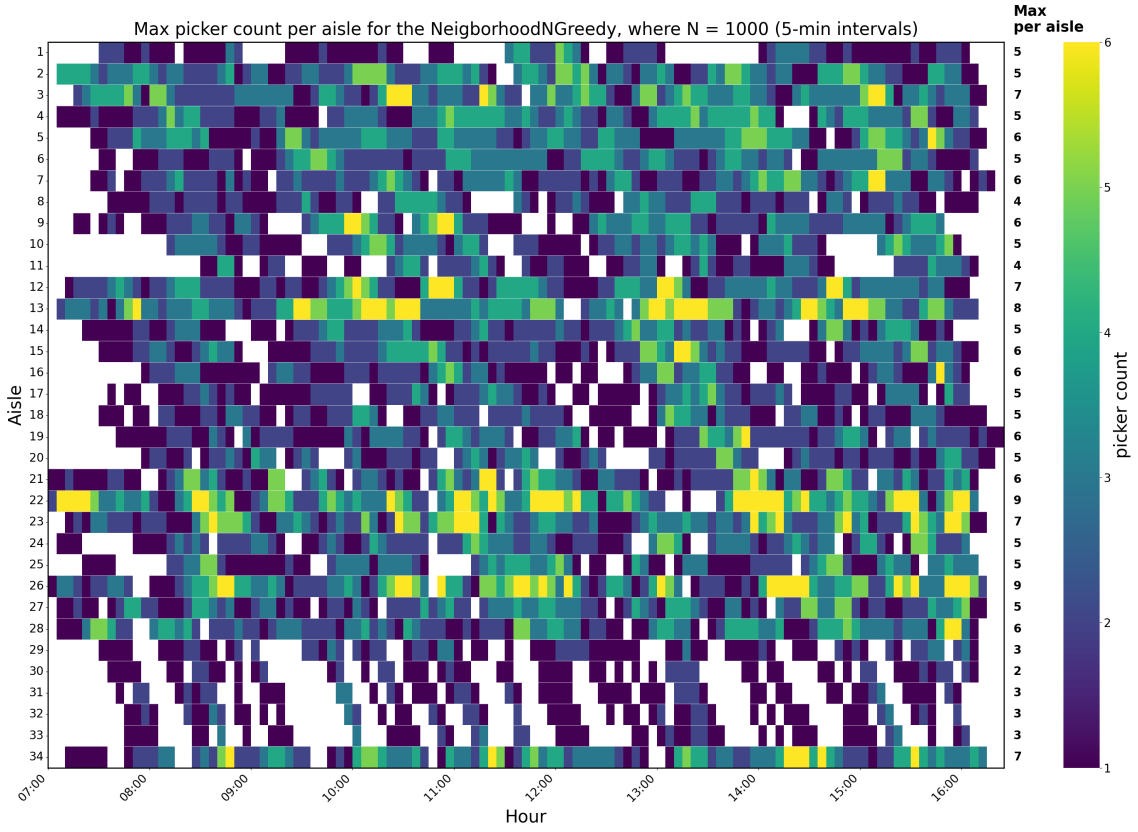


(a) NeighborhoodN Greedy ($N = 0$), realizing $M = 10$

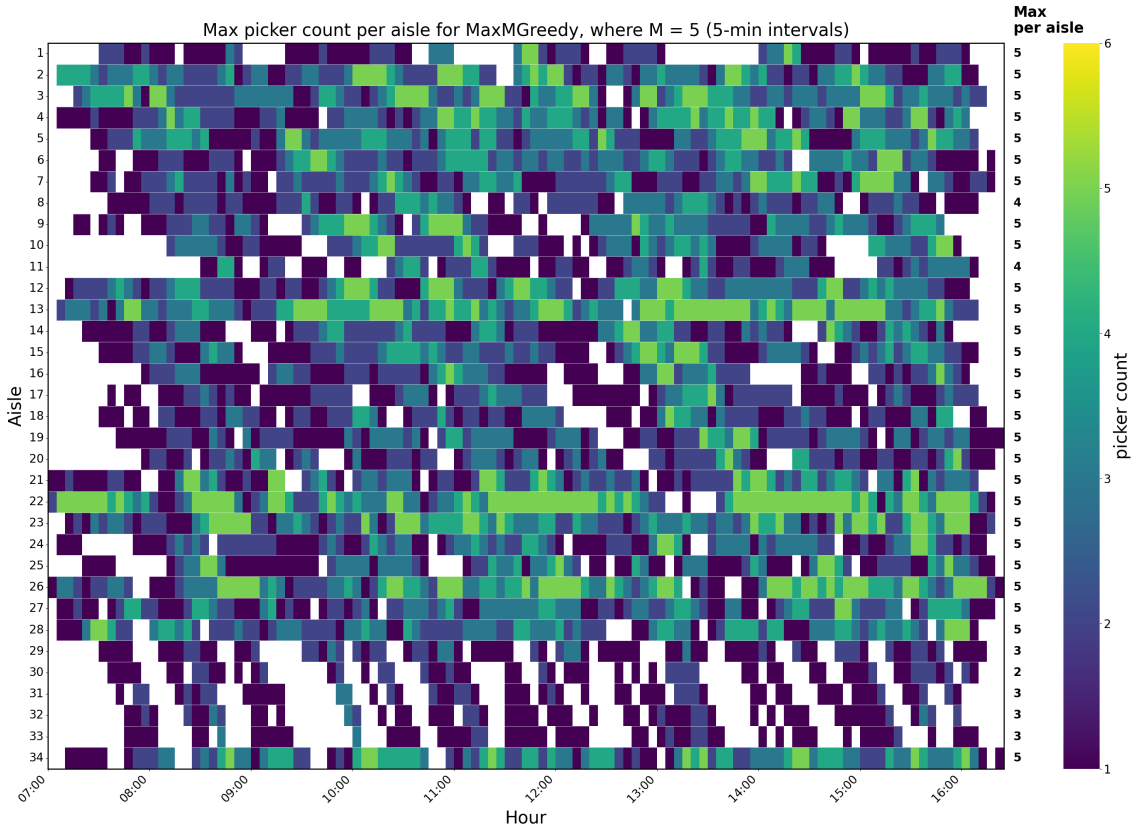


(b) ILP, realizing $M = 4$

Figure 5.8: Maximum picker count per aisle in 5-minute intervals on the busy day, shown for NeighborhoodN Greedy ($N = 0$) and the ILP. The colour scale is capped at $M = 6$; the maximum achieved per aisle is indicated numerically on the right. Lighter colours indicate higher aisle load.



(a) NeighborhoodN Greedy ($N = 1000$), realizing $M = 9$



(b) MaxM Greedy ($M = 5$), realizing $M = 5$

Figure 5.9: Maximum picker count per aisle in 5-minute intervals on the busy day, shown for NeighborhoodN Greedy ($N = 1000$) and MaxM Greedy. The colour scale is capped at $M = 6$; the maximum achieved per aisle is indicated numerically on the right. Lighter colours indicate higher aisle load.

5.3. Achieved Maximum Aisle Load: All Days and Warehouses

The previous sections examined the busy day (Day 10) and the average day (Day 13) in Warehouse 2 in detail. This section extends the comparison to all sixteen days in the dataset and both warehouses, checking whether the patterns observed for these two days generalize more broadly.

Tables 5.1 and 5.2 summarize the achieved M for all models across both warehouses (Days 1–8: Warehouse 1, Days 9–16: Warehouse 2). Several observations stand out. The ILP consistently achieves a lower M than all greedy variants across all days and both warehouses. The realized M without any scheduling model is substantially higher on busy days, consistent with aisle load increasing alongside the number of pickers. The NeighborhoodN Greedy with $N = 1000$ achieves a lower M than $N = 0$ on most days, though typically only by 1 to 4, with the two variants tied on the remaining days (2, 4, 8, 14 and 16). A possible explanation for these ties is that, on some days, only a small group of batches is available to schedule at a given point in time, all requiring the same aisles, leaving no flexibility to spread the load regardless of the neighborhood size. Day 5 in Warehouse 1 is a notable exception: $N = 0$ achieves $M = 13$ while $N = 1000$ achieves $M = 4$, a reduction of 9, compared to a typical reduction of 0 to 3 on the other seven days in Warehouse 1.

What also stands out is that the realized M does not consistently sit below the NeighborhoodN Greedy ($N = 0$) values, even though $N = 0$ is intended to model current practice. Realized M is lower on eight of the sixteen days, higher on five, and equal on the remaining three, with no strong pattern by warehouse: three of eight days lower in Warehouse 1 versus five of eight days in Warehouse 2. This is worth noting given that $N = 0$ assigns batches at their norm time without any aisle load awareness, and is examined further in Chapter 6.2.

For each day, the reported M value of the MaxM Greedy is the lowest bound for which zero tardiness can still be achieved.

Day	**Realized** M	**NeighborhoodN (N=0/N=1000)** M	**ILP** M	**MaxM** M
1	11	14 / 13	4	6
2	13	15 / 15	4	6
3	8	8 / 7	2	3
4	12	10 / 10	4	6
5	9	13 / 4	3	3
6	16	11 / 9	3	6
7	9	8 / 5	3	4
8	7	7 / 7	3	4

Table 5.1: Achieved maximum aisle load M for all models across all eight days in Warehouse 1. Realized M reflects the current situation without aisle load-aware scheduling.

Day	**Realized** M	**NeighborhoodN (N=0/N=1000)** M	**ILP** M	**MaxM** M
9	12	10 / 9	4	6
10	10	10 / 9	4	5
11	8	9 / 7	3	3
12	9	10 / 7	4	5
13	8	9 / 6	3	4
14	10	11 / 11	4	5
15	9	11 / 7	3	4
16	9	7 / 7	4	3

Table 5.2: Achieved maximum aisle load M for all models across all eight days in Warehouse 2. Realized M reflects the current situation without aisle load-aware scheduling.

5.4. Tardiness, Makespan and the Aisle Load Tradeoff

Having confirmed that the reduction in aisle load holds across the full dataset, this section turns to what that reduction costs. Reducing M is not without consequences: as discussed in Section 2.3.7,

minimizing one scheduling objective often comes at the cost of another, and this tension is clearly visible in the results that follow. The three models are compared in turn, showing how the achieved aisle load, tardiness, and makespan pull in opposing directions depending on the model and its configuration.

5.4.1. NeighborhoodN Greedy makespan behavior

Tables 5.3 and 5.4 report the makespan of the NeighborhoodN Greedy with $N = 0$, the proxy for current practice. Unlike the ILP, this model has no explicit mechanism enforcing the *EST-LST* window, yet it still achieves zero tardiness on every day in both warehouses. Two days in Warehouse 1 stand out from the rest: Day 6 has the highest makespan at 767 minutes, and Day 5 has the lowest at 387 minutes, against a 449–621 minute range on the remaining six days. For Warehouse 2, the makespans are more balanced, staying within a narrower 487–573 minute range across all eight days.

Day	Makespan (min)
1	597
2	595
3	533
4	621
5	387
6	767
7	449
8	523

Table 5.3: NeighborhoodN ($N = 0$) makespan across all eight days in Warehouse 1. Makespan is reported in minutes from the start of the afternoon window at 07:00.

Day	Makespan (min)
9	532
10	562
11	528
12	573
13	487
14	507
15	559
16	504

Table 5.4: NeighborhoodN ($N = 0$) makespan across all eight days in Warehouse 2. Makespan is reported in minutes from the start of the afternoon window at 07:00.

5.4.2. ILP makespan and tardiness behavior

The ILP minimizes M by exploiting the full scheduling window available to each batch, assigning batches anywhere between their earliest start time (*EST*) and latest start time (*LST*). Towards the end of a picking day, the *EST-LST* window for remaining batches becomes considerably wider, giving the ILP more flexibility to spread assignments and reduce the maximum aisle load. However, this same flexibility means that some batches are assigned close to their *LST*, extending the overall makespan of the schedule. Since the ILP does not penalize makespan in its current objective, it has no incentive to avoid assigning a batch close to its *LST* whenever doing so helps reduce M. This means the schedule's completion time is effectively bounded by how late the model chooses to push its last assignment, rather than by any explicit makespan target.

This behavior is visible in Figures 5.4 and 5.5, already shown in Section 5.2.2. The ILP keeps pickers active considerably later into the day compared to all greedy variants, which concentrate the workload earlier. On the busy day this effect is more pronounced, consistent with the wider *EST-LST* windows that arise when more batches are present later in the shift.

Comparing the ILP makespan to the NeighborhoodN Greedy ($N = 0$) makespan in Tables 5.3 and 5.4 shows the ILP finishing considerably later on every day in both warehouses, confirming that the reduction in M comes at a real cost and is not just a feature of the two days shown earlier. The gap is smallest on Day 6 in Warehouse 1, where the ILP takes 797 minutes against 767 for the NeighborhoodN Greedy ($N = 0$), and largest on Day 5 in Warehouse 1, where the ILP takes more than twice as long as the baseline (847 against 387 minutes).

Since the ILP assigns every batch within its *EST-LST* window by construction, it produces zero tardiness in every instance across all days and both warehouses. The makespan results are reported in Tables 5.5 and 5.6.

Day	ILP Makespan (min)
1	858
2	827
3	740
4	792
5	847
6	797
7	699
8	802

Table 5.5: ILP makespan across all eight days in Warehouse 1. Makespan is reported in minutes from the start of the afternoon window at 07:00. The ILP produces zero tardiness on every day by construction.

Day	ILP Makespan (min)
9	823
10	870
11	836
12	817
13	804
14	831
15	613
16	790

Table 5.6: ILP makespan across all eight days in Warehouse 2. Makespan is reported in minutes from the start of the afternoon window at 07:00. The ILP produces zero tardiness on every day by construction.

5.4.3. MaxM threshold analysis

The MaxM Greedy model offers a different perspective on this tradeoff. By enforcing a hard bound on M, it directly controls aisle load at the cost of potentially delaying batches that cannot be assigned without exceeding the bound. The key question is therefore: for a given day, what is the lowest M for which zero tardiness can still be achieved? This is in line with the hard constraint of scheduling the batches while upholding the EST_i and LST_i time window of each batch, which the ILP satisfies for every batch by construction. Asking for the lowest M at which the MaxM Greedy also reaches zero tardiness therefore gives us a fair comparison point between the two methods.

Table 5.7 shows this pattern for the busy day (Day 10) and the average day (Day 13). For sufficiently large values of the bound, MaxM achieves zero tardiness, meaning all batches are completed within their time windows. As the bound is tightened, tardiness increases, reflecting the cost of enforcing stricter aisle load control. The busy day needs a bound of $M = 5$ before tardiness reaches zero, while the average day already reaches zero tardiness at $M = 4$, consistent with the higher realized M observed in Section 5.1. This gives the MaxM model a tunable character: the input M bound directly controls where the solution sits on the curve between low aisle load and low tardiness. The same pattern holds across the full set of sixteen days and both warehouses, reported in full in Table A.5 in the Appendix, where the day-specific threshold can be read off directly.

Regarding makespan we see a decrease at first but at higher values of M it can be observed that it does not decrease monotonically once the tardiness constraint is already 0.

Day		$M = 2$	$M = 3$	$M = 4$	$M = 5$	$M = 6$
Day 10 (busy)	Makespan (minutes)	894	686	582	562	562
	Total Tardiness (minutes)	55227	6078	257	0	0
Day 13 (average)	Makespan (minutes)	709	517	488	486	487
	Total Tardiness (minutes)	9389	22	0	0	0

Table 5.7: Makespan C_{max} and total tardiness $\sum T_j$ (both in minutes) for the MaxM Greedy on the busy day and the average day, Warehouse 2, for varying M bounds. A tardiness of 0 indicates all batches were completed within their EST and LST time windows.

Taken together, the three models occupy distinct positions on this tradeoff. The ILP achieves the lowest M across all days and both warehouses, at the cost of a longer makespan. The MaxM Greedy occupies a controllable intermediate point, where the choice of M determines the balance between aisle load and tardiness, and where zero tardiness is achievable above a day-specific threshold. The NeighborhoodN Greedy variants occupy the high-M end of the spectrum, with the most concentrated workload profiles and no direct control over aisle load. Despite always scheduling batches as soon as both pickers and batches become available, The NeighborhoodN Greedy does not achieve the shortest makespan in every case. On several days, MaxM Greedy achieves a lower makespan instead.

5.5. Computation Time

Beyond the aisle load-tardiness tradeoff just discussed, the practical applicability of each model also depends on how long it takes to produce a schedule. Tables 5.8 and 5.9 report the ILP computation time across all days in each warehouse. All computations were performed on a single machine, an HP EliteBook 840 G8, equipped with an [Intel Core i7-1165G7] processor ([4] cores, [8] threads, base clock [2.80] GHz) and [16] GB of RAM, running [Windows 11 (64-bit)]. No parallelization across multiple machines or cloud infrastructure was used; all reported computation times in TablesTables 5.8 and 5.9 reflect single-machine, single-run performance under this configuration.

The greedy variants are omitted from the tables, since they consistently produce a schedule in under one second regardless of day or warehouse, reflecting their sequential assignment logic that does not require solving an optimization problem.

Day	ILP time (mm:ss)
1	20:14
2	0:30
3	0:21
4	0:45
5	136:34
6	3:07
7	4:29
8	51:47

Table 5.8: ILP computation time across all days in Warehouse 1.

Day	ILP time (mm:ss)
9	7:56
10	31:08
11	3:45
12	17:49
13	29:45
14	16:28
15	40:10
16	30:09

Table 5.9: ILP computation time across all days in Warehouse 2.

The ILP computation time varies considerably across days, ranging from 21 seconds to over 2 hours in Warehouse 1, and there is no clear relationship to whether a day is the busy or average case used in the detailed comparison: day 5, the longest runtime in the dataset, is not one of the two days examined in detail in Sections 5.1 through 5.4. Runtime does not scale monotonically with the number of pickers either: day 5 has 42 pickers, fewer than five of the other seven days in Warehouse 1, yet takes the longest to solve. This runtime unpredictability, and its implications for deploying the ILP in practice, is discussed further in Chapter 6.2.

Across all sixteen days and both warehouses, the same pattern holds. The current assignment practice, which does not take aisle load into account, regularly pushes the aisle load above the operational threshold of 6, while all three models bring it down substantially. The ILP achieves the lowest aisle load throughout, but at the cost of a longer makespan and solve times that can take hours. The MaxM Greedy offers a tunable middle ground between aisle load and tardiness, and the NeighborhoodN Greedy variants approximate current practice most closely. The next chapter reflects on what these results mean for warehouse operations and where the models could be extended.

6 Conclusions and Recommendations

6.1. Conclusions

This thesis set out to answer how order-to-picker assignment can be optimized to minimize aisle congestion in a multi-aisle warehouse, without changing how orders are batched or how pickers route through the warehouse. The four sub-questions introduced in Chapter 1 are answered in turn below, using the formulations from Chapter 3 and the results from Chapter 5.

How can minimizing simultaneous aisle occupancy be formalized as a scheduling problem, and how does its computational complexity behave?

The problem is formalized by treating pickers as parallel machines and batches as jobs, where each batch is a fixed sequence of aisle visits that must run without interruption, each aisle acts as a shared, renewable resource with limited capacity (2.3). In its simple form, where a batch is assumed to occupy all its aisles during one shared time slot, this problem is shown to be NP-complete, through a reduction from edge colouring (Section 3.1.3). This means that even in this form, no algorithm is known that finds the optimal assignment in polynomial time. The extended formulation used for the actual experiments in Chapter 5 adds non-uniform processing times and forces the aisles within a batch to be visited one at a time rather than simultaneously. This is a structurally different problem, not a direct generalization of the simplified version, so its computational complexity was not formally established in this thesis. Given its resemblance to known NP-hard resource constrained flow shop problems, we expect it to be NP-hard as well, but a formal proof still needs to be worked out (Section 3.2.2).

How can this problem be formulated as an integer linear program that captures realistic operational constraints such as fixed aisle sequences, non-uniform processing times and time windows, and what congestion reduction and computational cost does it achieve on real warehouse data?

Chapter 3 extends the simplified formulation into an ILP that respects the order in which a batch visits its aisles, gives each aisle its own processing time, and only allows a batch to start within its earliest and latest start time. Applied to real data from two warehouses over eight days each (Chapter 4, Chapter 5), this ILP consistently achieves the lowest maximum aisle load of any method tested. The aisle load falls between 2 and 4 across all sixteen instances, compared to a realized maximum aisle load of 7 to 16 under current, congestion-unaware assignment (Section 5.3). It does so without ever finishing a batch late, since every batch is scheduled within its own time window by construction. This comes at a cost: to keep the aisle load this low, the ILP uses the full width of the available time windows, which pushes batches later into the shift and produces a longer makespan than current practice (Section 5.4.2). It is also expensive to compute, with solve times ranging from about twenty seconds to over two hours across instances of similar size, with no clear relationship between instance size and solve time (Section 5.5). This makes the ILP in its current form most useful as a benchmark for what is achievable, rather than as a tool that could be rerun fresh every day in practice.

What greedy heuristics can be used at practical computation times, and how do they perform relative to the ILP and to current, congestion-unaware assignment practice?

Chapter 3 introduces two greedy heuristics as scalable alternatives to the ILP: NeighborhoodN Greedy, which builds a schedule while trying to avoid increasing the aisle load within a limited look-ahead window, and MaxM Greedy, which takes a maximum aisle load as a fixed input and schedules as many batches as possible without exceeding it. Both run in under one second on every instance tested, compared to the ILP's runtime of up to several hours (Section 5.5), which makes them realistic candidates for daily operational use.

Their solution quality falls between the ILP and current practice. The NeighborhoodN Greedy with no look-ahead ($N = 0$) is intended to approximate current assignment practice, since it mirrors the same latest start time ordering. In practice it does not consistently track the realized aisle load: as shown in

Section 5.3, realized M is lower than the $N = 0$ value on eight of the sixteen days, higher on five, and equal on three, with no clear pattern by warehouse. A likely explanation is that $N = 0$ still uses the norm time and EST_i estimates of the model, whereas the realized data reflects practical inefficiencies the model does not capture, for example pick orders starting before their EST_i, as discussed in Section 4.3.3, or aisle times exceeding their norm time estimate. Since $N = 0$ removes only the congestion-aware component of the heuristic and not these other simplifications, it approximates current practice rather than reproducing it exactly.

Adding look-ahead ($N = 1000$) gives a modest reduction in most cases, though on some days the achieved M is the same. Looking at Figure 5.3, NeighborhoodN Greedy is not able to reduce the aisle load until the last peak, where it keeps M at 9. A likely reason is the mechanism by which batches are assigned: a batch is scheduled as soon as both a picker and a batch are available. If the pool of available batches is small at that moment, and the batches in it all require a similar set of aisles, NeighborhoodN Greedy has no alternative and is forced to push up the aisle load regardless of the look-ahead size. This is consistent with Figure 5.2, where around 12:00 NeighborhoodN Greedy keeps M below 6 throughout and does not peak to values between 7 and 9: here, enough batches are schedulable at each point in time without increasing M, so the look-ahead can find one of them.

A second reason is that even when the pool of available batches is large enough to keep M low at some point in time, batches with similar aisle requirements can simply be postponed. Once a picker becomes available and these postponed batches are the only ones left to schedule, they must be assigned regardless of the resulting increase in M.

The MaxM Greedy algorithm, by construction, respects whatever aisle load bound it is given, and for most days this bound can be pushed down close to the ILP's optimum before batches start finishing late (Section 5.3, Section 5.4.3). None of the heuristics tested match the ILP's aisle load exactly, but MaxM Greedy comes closest while remaining practical to run.

What is the tradeoff between minimizing aisle congestion and other scheduling objectives such as makespan and tardiness, and how consistent is this tradeoff across different warehouses and workload levels?

In the Sections 5.4 and 5.5 we see that reducing aisle congestion comes at a cost. For the ILP, the cost shows up as makespan: to keep the aisle load low, batches are pushed later into the shift, using the full width of their time windows, so the schedule finishes later than it needs to (Section 5.4.2). For MaxM Greedy, the cost is tardiness and makespan. Once the imposed bound on aisle load is set below a day-specific threshold, below which some batches can no longer be scheduled within their time window at all (Section 5.4.3). A threshold on tardiness is not fixed, busier days need a higher aisle load bound before we are able to realize 0 tardiness.

This tradeoff holds consistently across all eight days in both warehouses, not only on the two days examined in detail. In every instance, the ILP achieves the lowest aisle load, MaxM Greedy sits between the ILP and current practice depending on the bound chosen, and both NeighborhoodN Greedy variants stay closest to current practice (Section 5.4.3). The absolute numbers scale with workload, busier days show both higher realized congestion and a higher minimum aisle load bound before MaxM Greedy avoids tardiness, but the ordering between methods, and the existence of the tradeoff itself, does not change from day to day or between the two warehouses.

Taken together, these four findings answer the main research question:

How can order-to-picker assignment be optimized to minimize peak aisle congestion in a warehouse?

Aisle congestion, modeled as the number of pickers active in an aisle, in a warehouse can be reduced substantially through scheduling alone. This can be done without touching how pick orders are batched or how pickers route through the warehouse, and this holds regardless of which warehouse or how busy the day is. The ILP demonstrates what is achievable at its best: it reaches the lowest aisle load of any method tested, by making full use of each batch's available time window, though this comes with a longer makespan and a computation time that can run from seconds to hours. MaxM Greedy turns that potential into a practical, everyday tool: it comes close to the ILP's congestion reduction while producing a schedule in under a second, and it can be tuned to whichever aisle load a warehouse is

willing to accept. Between the two, MaxM Greedy stands out as the stronger candidate for day-to-day use, delivering most of the ILP's benefit at a fraction of the computational cost.

6.2. Limitations and Further Research

The limitations of this research fall into four groups: refinements that make the model's inputs and constraints more realistic, extensions that relax the scope decisions made in Chapter 1, improvements to the solution methods themselves, and alternative objectives that ask a different question of the same underlying problem.

6.2.1. Refining the model's realism

Several refinements could bring the model's inputs closer to what actually happens on the warehouse floor. The norm time per aisle, defined in Section 4.3.1 as a practical approximation given the available data, could be estimated more precisely with a finer breakdown of picking actions. Norm times are currently expressed in whole minutes, and discretizing them into smaller time segments would let the model capture more of this detail. Figure 4.2 shows that for a subset of batches the realized aisle visit time is more than twice the estimated norm time, so a more accurate estimate should improve the model directly. A perfect estimate will remain difficult to achieve, however, since some of the practical inefficiencies behind these deviations cannot be captured in a norm time at all.

One practical factor the model currently has no notion of is picker breaks. In practice, breaks remove pickers from the floor for part of the day, and this would need to be reflected in when pickers are actually available to be assigned work. Adding this would bring the model closer to how a real shift actually runs.

A refinement that could help address these practical inefficiencies directly would be to give the model a margin, or buffer, on top of the norm time it schedules with. This would let a schedule that looks safe under the estimated norm times avoid silently exceeding its intended aisle load once real picking times run long. One way to implement this would be to extend a batch's wrap-up time, which would lengthen its total norm time without changing how long it is estimated to occupy any physical aisle. Whether such a buffer should be a fixed margin or one that only applies under certain conditions needs further investigation. An alternative to a static buffer would be to re-run the scheduling algorithm each time a batch finishes, reassessing which batch is best to schedule next given the actual state of the floor at that moment, rather than relying on a margin decided in advance.

Finally, the aisle load could be measured at a finer spatial resolution than a whole aisle, for example by splitting each aisle into two halves. This would let congestion be measured over the specific area pickers actually share, rather than the aisle as a whole, matching the wide-aisle congestion mechanism described in Section 2.1.4, where blocking occurs mainly at the slot of an item two pickers both need, rather than across an entire aisle. This finer resolution would, however, make the results considerably more sensitive to errors in the estimated norm time, reinforcing the point made above that discretizing norm times into smaller segments than a whole minute would likely be necessary before this refinement is worth pursuing. This finer resolution also raises the question of overtaking: a picker with a shorter norm time in an aisle than another picker occupying the same aisle would need to either overtake or wait, whereas pickers moving through an aisle at the same speed would not face this issue.

6.2.2. Extending beyond the current scope

Three of the boundaries set out in the Scope section of Chapter 1 are natural candidates for relaxation in future work.

The first is the assumption that batching and routing are fixed inputs, determined entirely by the existing algorithms described in Section 1.1. This thesis deliberately treated these as given, so that the reduction in congestion achieved could be attributed to scheduling alone. Relaxing this assumption, so that pick orders are formed with aisle congestion in mind rather than only travel time, could reduce congestion further still, at the cost of the clean separation between batching, routing and assignment that made the problem tractable to isolate here. Sections 2.1.2 and 2.1.3 already show that batching, routing and assignment are tightly coupled in the wider literature, so this would mean moving from the assignment-only formulation used here toward one of the integrated batching-routing-assignment

approaches discussed there.

The second is the assumption of a constant number of pickers throughout the scheduled period, this ties in with refining the model's realism. Section 4.2 estimates this number as the median picker count over the afternoon window specifically to satisfy this assumption, even though Figure 4.1 shows picker counts fluctuating meaningfully within that window. A model that allows the number of available pickers to vary over time, reflecting shift changes or pickers moving between order picking and other tasks, would remove the need for this approximation and could make fuller use of the picker capacity actually available at each moment.

Another, more advanced extension would be to include other warehouse processes and their effect on congestion. The most significant of these is restocking, which is carried out using forklifts that also occupy the aisles, and modelling this alongside order picking would add considerable complexity to the model.

Together, these extensions would not only make the models more representative of warehouse operations, but could also improve the correspondence between the model output and the actual aisle load. As shown in Section 5.3, the maximum aisle load obtained by NeighborhoodN Greedy ($N = 0$) captures some of the realized aisle load, but the two are not in a one-to-one correspondence. Reducing the simplifying assumptions discussed above may narrow this gap, leading to a more meaningful comparison between the current operation and the model results.

6.2.3. Extending the algorithms

Besides extending the heuristics, the ILP formulation itself could likely be solved faster without changing what it models. General techniques for speeding up MIP solve times, such as reformulating constraints into a tighter or more solver friendly form, changing how the input data is presented to the solver, or even adding constraints that are redundant given the ones already present, are known to reduce solve times considerably in other applications. That last option sounds counterintuitive, since a constraint that does not change the feasible region cannot make the problem easier in a strict mathematical sense, but such constraints can still tighten the LP relaxation or help the solver prune branches earlier, which reduces solve time in practice.

As noted above, extending the heuristics to support a varying number of pickers over time would be a natural first step, since both NeighborhoodN Greedy and MaxM Greedy currently check picker capacity against a single fixed value of w at every time step, rather than one that could change throughout the day.

A second extension concerns how NeighborhoodN Greedy's aisle load bound M behaves over the course of a schedule. Once M is increased at some point in the schedule, it is never decreased again. The only operation performed on M in Algorithm 3 is to increment it (line 32). This has a consequence that goes beyond simply carrying a higher bound forward. Once M has been raised, the ExceedM check used by the look-ahead is evaluated against this new, higher value, so it becomes far less likely to reject any candidate batch as exceeding it. In practice, this means a single unavoidable increase in M early in the schedule can disable the look-ahead's ability to prevent further, avoidable increases later on, even at a moment when a lower aisle load would still have been possible.

Two directions could address this. The more targeted one would be to let M decrease again, for example by periodically checking whether a lower bound would still be feasible for the batches scheduled so far. The more general one would be to add a local search or backtracking step after the initial greedy construction. This step could revisit earlier assignments once the full picture of the day is known, potentially recovering some of the gap to the ILP without giving up all of the heuristic's speed.

6.2.4. Exploring alternative objectives

Throughout this thesis, the objective has been to minimize the maximum aisle load M, with makespan and tardiness reported only as secondary indicators. Several alternative formulations follow naturally from flipping this around.

The most direct is to fix a maximum aisle load M as a hard constraint and minimize the makespan instead, effectively asking not how low congestion can go, but how fast the day can be finished under an

acceptable level of congestion.

A related direction is to minimize the number of pickers needed to achieve a target M or a target makespan, which answers a staffing question rather than a scheduling question: given a congestion limit a warehouse is willing to accept, how many pickers are actually needed while maintaining zero tardiness?

A further objective worth minimizing directly is picker idle time. The current formulations do not penalize a picker waiting while capacity is available, only the aisle load and, indirectly through tardiness, how late a batch finishes. Making idle time an explicit part of the objective would likely interact with both points above, since spreading batches out to reduce M, as the ILP already does, tends to leave pickers idle early in the schedule while waiting for later batches to become eligible.

More generally, Section 2.3.7 showed that minimizing aisle load and minimizing makespan or tardiness are structurally conflicting objectives, in the same way Yepes-Borrero et al. [36] describe for parallel machine scheduling with shared setup resources. This thesis explored that tradeoff by running each method under several fixed parameter values and comparing the outcomes, as in Section 5.4. A genuine bi-objective formulation, solved with a method such as theirs, could instead produce a full Pareto front of schedules directly, rather than the discrete set of tradeoff points obtained here. The most direct is to fix a maximum aisle load M as a hard constraint and minimize the makespan instead, effectively asking not how low congestion can go, but how fast the day can be finished under an acceptable level of congestion. Worth noting is that the minimum achievable makespan under this objective is not arbitrary. It is fixed by whichever batches have the latest EST or longest duration, since these determine the earliest possible finish time regardless of how the remaining batches are distributed across aisles. This offers a possible explanation for the observation in Section 5.4 that NeighborhoodN Greedy, despite always scheduling batches as soon as they become available, does not achieve the shortest makespan on every day, the floor is set by specific batches rather than by dispatch speed.

References

[1] Babak Abbasi, Shahram Shadrokh, and Jamal Arkat. "Bi-objective resource-constrained project scheduling with robustness and makespan criteria". In: *Applied Mathematics and Computation* 180.1 (Sept. 2006), pp. 146–152. ISSN: 0096-3003. DOI: 10.1016/j.amc.2005.11.160. URL: https://www.sciencedirect.com/science/article/pii/S0096300305011057 (visited on 03/18/2026).

[2] Ali Allahverdi, C. T. Ng, T. C. E. Cheng, and Mikhail Y. Kovalyov. "A survey of scheduling problems with setup times or costs". In: *European Journal of Operational Research* 187.3 (June 2008), pp. 985–1032. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2006.06.060. URL: https://www.sciencedirect.com/science/article/pii/S0377221706008174 (visited on 03/16/2026).

[3] Ehsan Ardjmand, Heman Shakeri, Manjeet Singh, and Omid Sanei Bajgiran. "Minimizing order picking makespan with multiple pickers in a wave picking warehouse". In: *International Journal of Production Economics* 206 (Dec. 2018), pp. 169–183. ISSN: 0925-5273. DOI: 10.1016/j.ijpe.2018.10.001. URL: https://www.sciencedirect.com/science/article/pii/S0925527318304146 (visited on 03/05/2026).

[4] Francisco Ballestín, Ángeles Pérez, Pilar Lino, Sacramento Quintanilla, and Vicente Valls. "Static and dynamic policies with RFID for the scheduling of retrieval and storage warehouse operations". In: *Computers & Industrial Engineering* 66.4 (Dec. 2013), pp. 696–709. ISSN: 0360-8352. DOI: 10.1016/j.cie.2013.09.020. URL: https://www.sciencedirect.com/science/article/pii/S0360835213003136 (visited on 03/10/2026).

[5] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. "Scheduling subject to resource constraints: classification and complexity". In: *Discrete Applied Mathematics* 5.1 (Jan. 1983), pp. 11–24. ISSN: 0166-218X. DOI: 10.1016/0166-218X(83)90012-4. URL: https://www.sciencedirect.com/science/article/pii/0166218X83900124 (visited on 03/16/2026).

[6] Jose A. Cano, Alexander A. Correa-Espinal, and Rodrigo A. Gómez-Montoya. "A review of research trends in order batching, sequencing and picker routing problems". en-US. In: *Revista ESPACIOS* 39.04 (Jan. 2018). URL: https://www.revistaespacios.com/a18v39n04/18390403.html (visited on 03/18/2026).

[7] Jose Alejandro Cano, Alexander A. Correa-Espinal, and Rodrigo Andrés Gómez-Montoya. "Mathematical programming modeling for joint order batching, sequencing and picker routing problems in manual order picking systems". In: *Journal of King Saud University - Engineering Sciences* 32.3 (Mar. 2020), pp. 219–228. ISSN: 1018-3639. DOI: 10.1016/j.jksues.2019.02.004. URL: https://www.sciencedirect.com/science/article/pii/S1018363918304616 (visited on 03/09/2026).

[8] Zhengcai Cao, Lijie Zhou, Chengran Lin, and Mengchu Zhou. "Solving an order batching, picker assignment, batch sequencing and picker routing problem via information integration". In: *Journal of Industrial Information Integration* 31 (Feb. 2023), p. 100414. ISSN: 2452-414X. DOI: 10.1016/j.jii.2022.100414. URL: https://www.sciencedirect.com/science/article/pii/S2452414X22000814 (visited on 03/09/2026).

[9] Tzu-Li Chen, Chen-Yang Cheng, Yin-Yann Chen, and Li-Kai Chan. "An efficient hybrid algorithm for integrated order batching, sequencing and routing problem". In: *International Journal of Production Economics* 159 (Jan. 2015), pp. 158–167. ISSN: 0925-5273. DOI: 10.1016/j.ijpe.2014.09.029. URL: https://www.sciencedirect.com/science/article/pii/S0925527314003077 (visited on 03/05/2026).

[10] T. C. E. Cheng, B. M. T. Lin, and H. L. Huang. "Resource-constrained flowshop scheduling with separate resource recycling operations". In: *Computers & Operations Research*. Special Issue on Scheduling in Manufacturing Systems 39.6 (June 2012), pp. 1206–1212. ISSN: 0305-0548. DOI: 10.1016/j.cor.2010.07.015. URL: https://www.sciencedirect.com/science/article/pii/S0305054810001462 (visited on 03/16/2026).

[11] Michele Ciavotta and Rubén Ruiz. "Resource constrained scheduling in permutation flowshop problems". en. In: (2011).

[12] Richard L. Daniels and Joseph B. Mazzola. "Flow Shop Scheduling with Resource Flexibility". In: *Operations Research* 42.3 (1994), pp. 504–522. ISSN: 0030-364X. URL: https://www.jstor.org/stable/171889 (visited on 03/17/2026).

[13] Bert De Reyck and Willy Herroelen. "The multi-mode resource-constrained project scheduling problem with generalized precedence relations". In: *European Journal of Operational Research* 119.2 (Dec. 1999), pp. 538–556. ISSN: 0377-2217. DOI: 10.1016/S0377-2217(99)00151-4. URL: https://www.sciencedirect.com/science/article/pii/S0377221799001514 (visited on 03/16/2026).

[14] A.J.R.M. (noud) Gademann, Jeroen P. Van Den Berg, and Hassan H. Van Der Hoff. "An order batching algorithm for wave picking in a parallel-aisle warehouse". en. In: *IIE Transactions* 33.5 (May 2001), pp. 385–398. ISSN: 1573-9724. DOI: 10.1023/A:1011049113445. URL: https://doi.org/10.1023/A:1011049113445 (visited on 03/10/2026).

[15] A.J.R.M. (noud) Gademann and Steef Velde. "Order batching to minimize total travel time in a parallel-aisle warehouse". In: *IIE Transactions* 37.1 (Jan. 2005). _eprint: https://doi.org/10.1080/07408170590516917, pp. 63–75. ISSN: 0740-817X. DOI: 10.1080/07408170590516917. URL: https://doi.org/10.1080/07408170590516917 (visited on 03/11/2026).

[16] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey". en-US. In: *Annals of Discrete Mathematics*. Vol. 5. Elsevier, Jan. 1979, pp. 287–326. DOI: 10.1016/S0167-5060(08)70356-X. URL: https://www.sciencedirect.com/science/chapter/bookseries/abs/pii/S016750600870356X (visited on 03/16/2026).

[17] Kevin R. Gue, Russell D. Meller, and Joseph D. Skufca. "The effects of pick density on order picking areas with narrow aisles". In: *IIE Transactions* 38.10 (Oct. 2006). _eprint: https://doi.org/10.1080/07408170600809341, pp. 859–868. ISSN: 0740-817X. DOI: 10.1080/07408170600809341. URL: https://doi.org/10.1080/07408170600809341 (visited on 03/04/2026).

[18] Sebastian Henn. "Order batching and sequencing for the minimization of the total tardiness in picker-to-part warehouses". en. In: *Flexible Services and Manufacturing Journal* 27.1 (Mar. 2015), pp. 86–114. ISSN: 1936-6590. DOI: 10.1007/s10696-012-9164-1. URL: https://doi.org/10.1007/s10696-012-9164-1 (visited on 03/04/2026).

[19] Sebastian Henn, Sören Koch, and Gerhard Wäscher. "Order Batching in Order Picking Warehouses: A Survey of Solution Approaches". en. In: *Warehousing in the Global Supply Chain: Advanced Models, Tools and Applications for Storage Systems*. Ed. by Riccardo Manzini. London: Springer, 2012, pp. 105–137. ISBN: 978-1-4471-2274-6. DOI: 10.1007/978-1-4471-2274-6_6. URL: https://doi.org/10.1007/978-1-4471-2274-6_6 (visited on 03/04/2026).

[20] Soondo Hong, Andrew L. Johnson, and Brett A. Peters. "Batch picking in narrow-aisle order picking systems with consideration for picker blocking". In: *European Journal of Operational Research* 221.3 (Sept. 2012), pp. 557–570. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2012.03.045. URL: https://www.sciencedirect.com/science/article/pii/S0377221712002706 (visited on 03/04/2026).

[21] Chih-Ming Hsu, Kai-Ying Chen, and Mu-Chen Chen. "Batching orders in warehouses by minimizing travel distance with genetic algorithms". In: *Computers in Industry*. Applications of Genetic Algorithms in Industry 56.2 (Feb. 2005), pp. 169–178. ISSN: 0166-3615. DOI: 10.1016/j.compind.2004.06.001. URL: https://www.sciencedirect.com/science/article/pii/S0166361504000995 (visited on 03/11/2026).

[22] René de Koster, Tho Le-Duc, and Kees Jan Roodbergen. "Design and control of warehouse order picking: A literature review". In: *European Journal of Operational Research* 182.2 (Oct. 2007), pp. 481–501. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2006.07.009. URL: https://www.sciencedirect.com/science/article/pii/S0377221706006473 (visited on 03/04/2026).

[23] Osman Kulak, Yusuf Sahin, and Mustafa Egemen Taner. "Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms". en. In: *Flexible Services and Manufacturing Journal* 24.1 (Mar. 2012), pp. 52–80. ISSN: 1936-6590. DOI: 10.1007/s10696-011-9101-8. URL: https://doi.org/10.1007/s10696-011-9101-8 (visited on 03/05/2026).

[24] Makusee Masae, Christoph H. Glock, and Eric H. Grosse. "Order picker routing in warehouses: A systematic literature review". In: *International Journal of Production Economics* 224 (June 2020), p. 107564. ISSN: 0925-5273. DOI: 10.1016/j.ijpe.2019.107564. URL: https://www.sciencedirect.com/science/article/pii/S0925527319304050 (visited on 03/04/2026).

[25] Fatemeh Nikkhoo, Ali Husseinzadeh Kashan, Ehsan Nikbakhsh, and Bakhtiar Ostadi. "A bi-objective multi-warehouse multi-period order picking system under uncertainty: a benders decomposition approach". en. In: *Soft Computing* 29.4 (Feb. 2025), pp. 2047–2074. ISSN: 1433-7479. DOI: 10.1007/s00500-025-10495-1. URL: https://doi.org/10.1007/s00500-025-10495-1 (visited on 03/05/2026).

[26] Nudtapon Nudtasomboon and Sabah U. Randhawa. "Resource-constrained project scheduling with renewable and non-renewable resources and time-resource tradeoffs". In: *Computers & Industrial Engineering* 32.1 (Jan. 1997), pp. 227–242. ISSN: 0360-8352. DOI: 10.1016/S0360-8352(96)00212-4. URL: https://www.sciencedirect.com/science/article/pii/S0360835296002124 (visited on 03/23/2026).

[27] Yanghua Pan, Shanshan Li, Ting Qu, Liqiang Ding, Naiqi Wu, and George Q. Huang. "Dynamic integrated optimization of batching and routing in narrow-aisle order picking systems with congestion consideration". In: *Swarm and Evolutionary Computation* 98 (Oct. 2025), p. 102109. ISSN: 2210-6502. DOI: 10.1016/j.swevo.2025.102109. URL: https://www.sciencedirect.com/science/article/pii/S2210650225002676 (visited on 03/04/2026).

[28] Eduardo G. Pardo, Sergio Gil-Borrás, Antonio Alonso-Ayuso, and Abraham Duarte. "Order batching problems: Taxonomy and literature review". In: *European Journal of Operational Research* 313.1 (Feb. 2024), pp. 1–24. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2023.02.019. URL: https://www.sciencedirect.com/science/article/pii/S0377221723001534 (visited on 03/09/2026).

[29] Pratik J. Parikh and Russell D. Meller. "Estimating picker blocking in wide-aisle order picking systems". In: *IIE Transactions* 41.3 (Jan. 2009). _eprint: https://doi.org/10.1080/07408170802108518, pp. 232–246. ISSN: 0740-817X. DOI: 10.1080/07408170802108518. URL: https://doi.org/10.1080/07408170802108518 (visited on 03/10/2026).

[30] Charles G. Petersen and Gerald Aase. "A comparison of picking, storage, and routing policies in manual order picking". In: *International Journal of Production Economics* 92.1 (Nov. 2004), pp. 11–19. ISSN: 0925-5273. DOI: 10.1016/j.ijpe.2003.09.006. URL: https://www.sciencedirect.com/science/article/pii/S0925527303002937 (visited on 03/04/2026).

[31] Michael Pinedo. "Scheduling: Theory, Algorithms, And Systems". In: Jan. 2008.

[32] Tapan Sen, Joanne M Sulek, and Parthasarati Dileepan. "Static scheduling research to minimize weighted and unweighted tardiness: A state-of-the-art survey". In: *International Journal of Production Economics* 83.1 (Jan. 2003), pp. 1–12. ISSN: 0925-5273. DOI: 10.1016/S0925-5273(02)00265-7. URL: https://www.sciencedirect.com/science/article/pii/S0925527302002657 (visited on 03/17/2026).

[33] H. Süral, S. Kondakci, and N. Erkip. "Scheduling unit-time tasks in renewable resource constrained flowshops". en. In: *Zeitschrift für Operations Research* 36.6 (Nov. 1992), pp. 497–516. ISSN: 1432-5217. DOI: 10.1007/BF01416242. URL: https://doi.org/10.1007/BF01416242 (visited on 03/16/2026).

[34] Stefan Voß and Andreas Witt. "Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: A real-world application". In: *International Journal of Production Economics*. Scheduling in batch-processing industries and supply chains 105.2 (Feb. 2007), pp. 445–458. ISSN: 0925-5273. DOI: 10.1016/j.ijpe.2004.05.029. URL: https://www.sciencedirect.com/science/article/pii/S0925527305002409 (visited on 03/18/2026).

[35] Mingcong Xia, Guokai Liang, Rui Tong, Jianxin Zhu, Xin Xie, Jintao Chen, Weihua Tan, and Yuting Liu. "Bi-Objective Optimization with Mode-Oriented Genetic Algorithm for Multi-Mode Resource-Constrained Project Scheduling". en. In: *Algorithms* 18.12 (Nov. 2025). ISSN: 1999-4893. DOI: 10.3390/a18120746. URL: https://www.mdpi.com/1999-4893/18/12/746 (visited on 03/18/2026).

[36] Juan C. Yepes-Borrero, Federico Perea, Rubén Ruiz, and Fulgencia Villa. "Bi-objective parallel machine scheduling with additional resources during setups". In: *European Journal of Operational Research* 292.2 (July 2021), pp. 443–455. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2020.10.052. URL: https://www.sciencedirect.com/science/article/pii/S0377221720309450 (visited on 03/18/2026).

A Appendix

A.1. Compact ILP formulations

A.1.1. The Aisle Load Balancing Problem

Input:

$$n, w, A \in \mathbb{Z}, \quad w \mid n,$$
$$B_i \subseteq [A] \quad \text{for } i \in [n].$$

$$b_{ai} = \begin{cases} 1 & \text{if } a \in B_i, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } a \in [A],\ i \in [n]. \tag{A.1}$$

Decision variables:

$$M \in \mathbb{Z}_{\geq 0},$$
$$x_{ij} = \begin{cases} 1 & \text{if } i \in P_j, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } i \in [n],\ j \in \left[\frac{n}{w}\right],$$

, where $\{P_j\}_{j \in [n/w]}$ is the partition of $[n]$ into sets of size w.

Objective:

$$\min M \tag{A.2}$$

Constraints:

$$\sum_{j=1}^{n/w} x_{ij} = 1, \qquad \forall i \in [n]. \tag{A.3}$$

$$\sum_{i=1}^{n} x_{ij} = w, \qquad \forall j \in \left[\frac{n}{w}\right]. \tag{A.4}$$

$$M \geq \sum_{i=1}^{n} b_{ai}\, x_{ij}, \qquad \forall j \in \left[\frac{n}{w}\right],\ \forall a \in [A]. \tag{A.5}$$

Symbol	Type	Index Set	Interpretation
n	Input parameter	–	Total number of batches.
w	Input parameter	–	Number of pickers, size of the sets.
A	Input parameter	–	Total number of aisles.
B_i	Input parameter	$i \in [n]$	Set of aisles in the route of batch B_i.
b_{ai}	Input parameter (0/1)	$a \in [A],\ i \in [n]$	1 if aisle $a \in$ batch B_i.
x_{ij}	Decision variable (0/1)	$i \in [n],\ j \in [n/w]$	1 if batch B_i is assigned to set j.
M	Decision variable ($\mathbb{Z}_{\geq 0}$)	–	Maximum aisle multiplicity across all sets.
P_j	Defined by x_{ij}	$j \in [n/w]$	Set j (set of batches with $x_{ij} = 1$).

Table A.1: Overview of variables and parameters in the ILP formulation.

A.1.2. The Extended Aisle Load Balancing Problem

Input:

$$\begin{aligned} &n, w, T, A \in \mathbb{Z}, \\ &\beta_i,\ EST_i,\ LST_i && \text{for } i \in [n]. \end{aligned}$$

Each batch is represented as a set of tuples containing the aisles and their corresponding processing time:

$$B_i = \left(\left(a_{i1}, p_{i1}\right), \left(a_{i2}, p_{i2}\right), \ldots, \left(a_{i\beta_i}, p_{i\beta_i}\right) \right), \tag{A.6}$$

where β_i denotes the number of aisles associated with batch B_i.

Predefined sets.

$$S_{ik} = \sum_{h=1}^{k-1} p_{ih}. \tag{A.7}$$

Decision variables.

$$x_{it} = \begin{cases} 1 & \text{if batch } B_i \text{ starts at time } t, \\ 0 & \text{otherwise,} \end{cases} \qquad \forall i \in [n],\ t \in [T]_{\geq 0}, \tag{A.8}$$

$$r_{ikt} = \begin{cases} 1 & \text{if the } k\text{-th aisle of batch } B_i \text{ is processed at time } t, \\ 0 & \text{otherwise,} \end{cases} \qquad \forall i \in [n],\ k \in [\beta_i],\ t \in [T]_{\geq 0}. \tag{A.9}$$

Objective.

$$\min M. \tag{A.10}$$

Constraints.

$$\sum_{t=EST_i}^{LST_i} x_{it} = 1 \qquad \forall i \in [n]. \tag{A.11}$$

$$r_{ikt} = \sum_{\tau=\max\{0,\, t-S_{ik}-p_{ik}+1\}}^{t-S_{ik}} x_{i\tau} \qquad \forall i \in [n],\ \forall k \in [\beta_i],\ \forall t \in [T]_{\geq 0}. \tag{A.12}$$

$$\sum_{i=1}^{n} \sum_{k=1}^{\beta_i} r_{ikt} \leq w \qquad \forall t \in [T]_{\geq 0}. \tag{A.13}$$

$$\sum_{i=1}^{n} \sum_{\substack{k \in [\beta_i] \\ a_{ik}=a}} r_{ikt} \leq M \qquad \forall a \in [A],\ \forall t \in [T]_{\geq 0}. \tag{A.14}$$

Symbol	Type	Index Set	Interpretation
n	Input parameter	–	Total number of batches.
w	Input parameter	–	Number of pickers.
T	Input parameter	–	Number of time periods in the horizon.
A	Input parameter	–	Number of aisles.
B_i	Input parameter	$i \in [n]$	Set of batches that contain the aisles and their processing times.
β_i	Input parameter	$i \in [n]$	Indicator that states how many aisles batch B_i contains.
a_{ik}	Input parameter	$i \in [n],\ k \in [\beta_i]$	Index of the k-th aisle in batch B_i
p_{ik}	Input parameter	$i \in [n],\ k \in [\beta_i]$	Processing time of the k-th aisle k in batch B_i.
S_{ik}	Input parameter	$i \in [n],\ k \in [\beta_i]$	Start time offsets for the aisles in batch B_i.
P_i	Input parameter	$i \in [n]$	Total processing time of batch B_i.
x_{it}	Decision variable (0/1)	$i \in [n],\ t \in [T]$	1 if batch B_i starts at time t.
r_{ikt}	Decision variable (0/1)	$i \in [n],\ k \in [\beta_i]$ $t \in [T]$	1 if aisle k of batch B_i is being processed at time t.
M	Decision variable ($\mathbb{Z}_{\geq 0}$)	–	Maximum aisle multiplicity across all times.

Table A.2: Overview of variables and parameters in the ALBP ILP formulation including processing time per aisle.

A.2. Results

A.2.1. Realized start times in a time window of earliest and latest start times

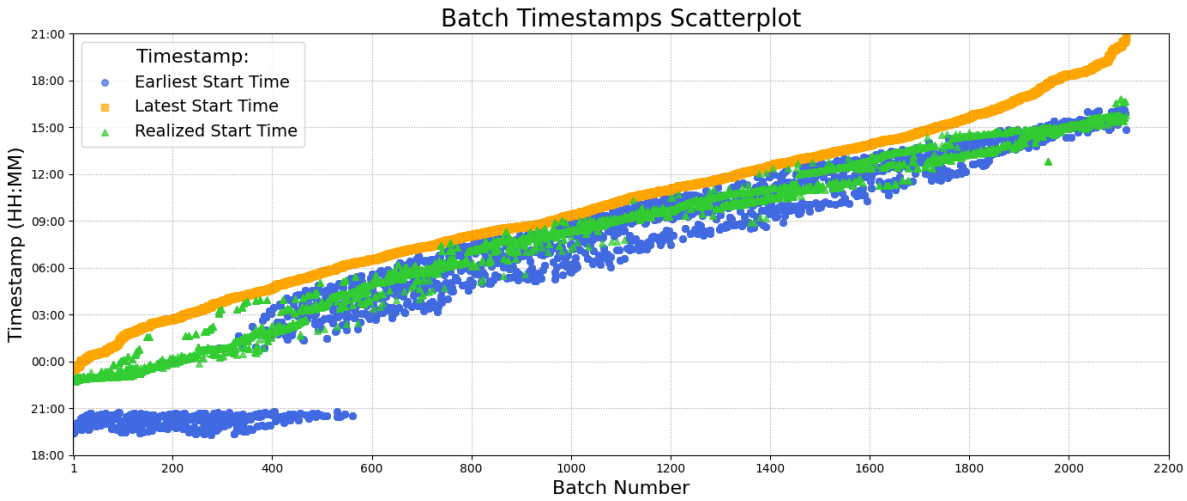


Figure A.1: Realized Start times within an EST LST Time window

A.2.2. Makespan and tardiness results for the NeighborhoodN Greedy

Day	Makespan for $N = 0$ and $N = 1000$	Total tardiness
Day 1	C_{max}: 597 / 597	$\sum T_j$: 0
Day 2	C_{max}: 595 / 595	$\sum T_j$: 0
Day 3	C_{max}: 533 / 533	$\sum T_j$: 0
Day 4	C_{max}: 621 / 621	$\sum T_j$: 0
Day 5	C_{max}: 387 / 385	$\sum T_j$: 0
Day 6	C_{max}: 767 / 767	$\sum T_j$: 0
Day 7	C_{max}: 449 / 448	$\sum T_j$: 0
Day 8	C_{max}: 523 / 523	$\sum T_j$: 0

Table A.3: Warehouse 1 N0/N1000 Results regarding MAkespan and Total tardiness

Day	Makespan for $N = 0$ and $N = 1000$	Total tardiness
Day 9	C_{max}: 532 /532	$\sum T_j$: 0
Day 10	C_{max}: 562 /562	$\sum T_j$: 0
Day 11	C_{max}: 528 /527	$\sum T_j$: 0
Day 12	C_{max}: 573 /572	$\sum T_j$: 0
Day 13	C_{max}: 487 /487	$\sum T_j$: 0
Day 14	C_{max}: 507 /507	$\sum T_j$: 0
Day 15	C_{max}: 559 /556	$\sum T_j$: 0
Day 16	C_{max}: 504 /504	$\sum T_j$: 0

Table A.4: Warehouse 2 N0/N1000 Results regarding Makespan and Total tardiness

A.2.3. MaxM Greedy makespan and tardiness results for all days in both warehouses

Day		$M = 2$	$M = 3$	$M = 4$	$M = 5$	$M = 6$
Day 1	C_{max}	886	641	597	597	597
	$\sum T_j$	11187	500	48	21	0
Day 2	C_{max}	903	816	656	603	595
	$\sum T_j$	49732	9930	827	18	0
Day 3	C_{max}	546	533	533	533	533
	$\sum T_j$	29	0	0	0	0
Day 4	C_{max}	872	744	642	620	622
	$\sum T_j$	18358	2353	63	4	0
Day 5	C_{max}	497	385	385	385	385
	$\sum T_j$	599	0	0	0	0
Day 6	C_{max}	813	769	767	767	767
	$\sum T_j$	12616	434	182	18	0
Day 7	C_{max}	587	453	442	443	437
	$\sum T_j$	1494	35	0	0	0
Day 8	C_{max}	632	528	523	523	523
	$\sum T_j$	974	36	0	0	0
Day 9	C_{max}	896	666	565	552	550
	$\sum T_j$	36966	4184	150	2	0
Day 10	C_{max}	894	686	582	562	562
	$\sum T_j$	55227	6078	257	0	0
Day 11	C_{max}	652	533	529	527	527
	$\sum T_j$	3324	0	0	0	0
Day 12	C_{max}	880	649	570	539	540
	$\sum T_j$	31171	1763	14	0	0
Day 13	C_{max}	709	517	488	486	487
	$\sum T_j$	9389	22	0	0	0
Day 14	C_{max}	879	628	526	508	507
	$\sum T_j$	30786	3455	35	0	0
Day 15	C_{max}	616	479	485	485	485
	$\sum T_j$	13165	198	0	0	0
Day 16	C_{max}	614	492	490	489	489
	$\sum T_j$	3760	0	0	0	0

Table A.5: Makespan and total tardiness (minutes) for the MaxM Greedy for varying M bounds, across all sixteen days (days 1–8: Warehouse 1, days 9–16: Warehouse 2). A tardiness of 0 indicates all pick orders were completed within their respective EST and LST time windows.