

Seminar Paper
Technical University Munich

Extension of the assembly line feeding problem by introducing multiple supermarkets

Examiner: Prof. Dr. Martin Grunow
Chair of Production and Supply Chain Management
Technical University Munich

Supervisor: Baturhan Bayraktar

Area of Study: Specialization in Operations and Supply Chain Management
Master in Management and Technology

Submitted by: Steffen Voigtländer
Matriculation Number: 03714876

Felix Immanuel Reibold
Matriculation Number: 03769602

Submission date: 09.01.2024

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here.

This paper was not previously presented to another examination board and has not been published.

Munich, 09.01.2024

Teer Reesl *Stefan Voglmeier*

Abstract

The Assembly Line Feeding Problem (ALFP) was first formalized in response to the emergence of mass customization. The supply of components to mixed-model assembly lines has proven to be a challenging endeavour, given that the quantity of parts needed for a single assembly line significantly increases.

Much effort has been made to model the complexities of assembly line feeding, as they occur in reality. Noteworthy extensions of the groundwork by Bozer and McGinnis (1992) are the inclusion of feeding policies, vehicle types and route determination. However, this work is the first to consider assembly line feeding from multiple supermarkets, instead of one.

In this study, we extend the mixed integer programming model by Adenipekun et al. (2022) which is designed to assign parts simultaneously to a feeding policy and a vehicle type, with the goal to minimise total feeding costs. We build upon it, by additionally assigning parts to supermarkets. Our work enables feeding the assembly line from multiple supermarkets to reduce distance-related transportation costs. This reflects production settings in the industry.

Table of Contents

| | |
|---|------|
| Declaration of Authorship | ii |
| Abstract..... | iii |
| Table of Contents..... | iv |
| List of Figures..... | v |
| List of Tables | vi |
| List of Abbreviations | vii |
| List of Symbols..... | viii |
| 1 Introduction | 1 |
| 2 Literature Review | 4 |
| 2.1 Disambiguation of the ALFP and the SLP | 4 |
| 2.2 Combining the Assembly Line Feeding Problem with factory-layout aspects | 5 |
| 3 Problem Description..... | 8 |
| 4 Methodology | 10 |
| 4.1 Mixed Integer Linear Problem (MILP) | 10 |
| 4.1.1 <i>Objective function</i> | 10 |
| 4.1.2 <i>General Constraints</i> | 11 |
| 4.1.3 <i>Route determination</i> | 12 |
| 4.1.4 <i>Line-sided storage</i> | 13 |
| 4.1.5 <i>Fleet size determination</i> | 13 |
| 4.2 Cost parameter calculation..... | 14 |
| 4.2.1 <i>Replenishment costs</i> | 14 |
| 4.2.2 <i>Preparation costs</i> | 14 |
| 4.2.3 <i>Transportation costs</i> | 15 |
| 4.2.4 <i>Usage costs</i> | 15 |
| 5 Results | 16 |
| 5.1 Numerical Experiments Design – Input Values | 16 |
| 5.2 Advantage of several supermarkets | 17 |
| 5.3 Variable distances and supermarket fix costs | 18 |
| 5.4 Feeding policies pre-assigned vs. assigned during optimization..... | 20 |
| 6 Conclusion..... | 23 |
| Reference List..... | 25 |
| Appendices | 27 |

List of Figures

| | |
|--|----|
| Figure 1: Layout, routing, and vehicle selection considerations | 8 |
| Figure 2: Total costs for different numbers of stations and supermarkets | 18 |
| Figure 3: Total costs for different numbers of stations and supermarkets | 19 |
| Figure 4: Different locations of one and two supermarkets at the production line | 20 |

List of Tables

| | |
|---|----|
| Table 1: Summary of the literature analysis | 7 |
| Table 2: Comparison of the model by Adenipekun et al. (2022) and the extended model | 9 |
| Table 3: Additional parameters | 17 |
| Table 4: Rules for manual assignment of feeding policies in datasets | 21 |
| Table 5: Comparison of the performance between introduced model and model with pre-assigned feeding policies | 22 |

List of Abbreviations

| | |
|------|-------------------------------------|
| AGV | Automated Guided Vehicle |
| ALFP | Assembly Line Feeding Problem |
| BoL | Border of Line |
| BOM | Bill of Materials |
| JIT | Just-in-Time |
| LP | Loading Problem |
| MILP | Mixed-Integer Linear Program |
| RP | Routing Problem |
| SLP | Supermarket Location Problem |
| SP | Scheduling Problem |
| TUM | Technische Universität München |
| TVSP | Transport Vehicle Selection Problem |
| VSP | Vehicle Selection Problem |
| VT | Vehicle Type |

List of Symbols

Sets

| | |
|-----------------|--|
| \mathcal{A} | Set of part attributes (indexed a) |
| \mathcal{B} | Set of Supermarkets (indexed b) |
| \mathcal{F} | Set of part families (indexed f) |
| \mathcal{F}_s | Set of part families at station s (indexed f) |
| \mathcal{I} | Set of parts (indexed i) |
| \mathcal{I}_f | Set of parts in family f (indexed i) |
| \mathcal{I}_s | Set of parts at station s (indexed i) |
| \mathcal{M} | Set of vehicle types (indexed m) |
| \mathcal{M}' | Set of vehicle types excluding forklift (indexed m) |
| \mathcal{P} | Set of line feeding policies (indexed p) |
| \mathcal{P}_w | Set of line feeding policies that use transportation flow w (indexed p) |
| \mathcal{R} | Set of routes (indexed r) |
| \mathcal{R}_s | Set of routes covering assembly station s (indexed r) |
| \mathcal{S} | Set of assembly stations (indexed s) |
| \mathcal{S}_r | Set of assembly stations served by route r (indexed s) |
| \mathcal{T} | Set of takts (indexed t) |
| \mathcal{V}_i | Set of variants of part i in the family of part i (indexed i) |
| \mathcal{W} | Set of transportation flows (indexed w) |
| \mathcal{W}_p | Set of transportation flows for feeding policy p (indexed w) |

Variables

| | |
|--------------|--|
| f_{fs} | Number of facings for part family f assigned to assigned to sequencing at station s |
| f_{ips} | Number of facings for part i assigned to feeding policy $p \in \{1,2\}$ at station s |
| f_s | Number of facings for stationary kitting at station s |
| k_{msb} | Binary variable that decides if a stationary kit is conveyed by vehicle type m to station s from supermarket b |
| q_{sb} | Binary variable that assigns station s to supermarket b |
| r_s | Number of racks for boxed supply in station s |
| t_{mb} | Binary variable that decides if a travelling kit is conveyed by vehicle type m to the assembly line from supermarket b |
| u_F^{ws} | Number of forklifts used between warehouse and supermarket |
| v_{pmtrwb} | Binary variable that decides on the frequency t of vehicle type m when transporting parts assigned to policy p in flow w over route r from supermarket b |
| x_{ipmb} | Binary variable that assigns part i to feeding policy p and vehicle type m |
| y_b | Binary variable that decides if supermarket b is active or not |

Parameters

| | |
|---------------|---|
| ca_m | Acquisition and maintenance cost for vehicle type m per takt |
| cap_b | Capacity of supermarket b |
| c_{fm} | Transportation cost for part family f and vehicle type m |
| c_{ipmb} | Costs of providing part i with feeding policy p and vehicle type m via supermarket b to the BoL |
| c_{mb} | Transportation cost for travelling kit provided by vehicle type m via supermarket b |
| c_{msb} | Transportation cost for a stationary kit provided by vehicle type m to station s via supermarket b |
| c_{pmtrwb} | Milk run transportation cost for deliveries made with vehicle type m when transporting parts assigned to policy p in flow w over route r with frequency t via supermarket b |
| d | Product demand (equal to the number of takts in the planning horizon) |
| d_b^{ws} | Two-way distance between warehouse and supermarket b |
| d_{bs}^s | Two-way distance between supermarket b and station s |
| d_s^w | Two-way distance between warehouse and station s |
| d_p | Depth of rack used for parts assigned to feeding policy $p \in \{1,2\}$ |
| f_i | Family of part i |
| Fix_S | Wage of logistics operator per 8h shift in a supermarket |
| h_2 | Number of shelves in a rack used for boxed supply containers |
| H | Planning horizon |
| L_p | Length of facings for feeding policy $p \in \{1, 2, 3, 4\}$ |
| l_{pm} | Loading and unloading time for a feeding policy p container on a vehicle type m |
| L_s | Length of an assembly station s at the BoL |
| λ_f | Demand for part family f |
| λ_i | Demand for part i |
| \mathcal{M} | Sufficiently large number |
| m_i | Number of units of part i in one end product |
| mr_{rwb} | Milk run length for route r in transportation flow w via supermarket b |
| μ_m | Utilization rate of vehicle type m |
| n_4^k | Number of stationary kits that fit into a kit container |
| n_5^t | Number of travelling kits that fit into a kit container |
| n_f | Number of parts in a part family f that fit into a sequenced container |
| n_{ip} | Number of parts i that fit into a bin or container of feeding policy $p \in \{1, 2\}$ |
| n_{pm} | Number of bins or containers of feeding policy $p \in \{1, 2, 3\}$ that fit into a vehicle type m |
| q_p | Minimum number of facings allocated to containers for feeding policy p at the BoL |
| r_{ap} | Resource availability of attribute a (weight or volume) for a container of policy p |
| r_{ia} | Resource demand of attribute a (weight or volume) for a unit of part i |
| v_m | Velocity of vehicle type m |
| w_2 | Number of boxes that fit on one shelf of a rack used for boxed supply containers |

1 Introduction

In the manufacturing industry, decentralized in-house logistics areas known as supermarkets are widely used for parts feeding to assembly lines (Faccio 2014). These supermarkets act as intermediate storage areas close to the assembly lines, providing a more efficient and streamlined process for material delivery. The use of supermarkets in assembly line feeding not only reduces inventory levels at the stations but also eliminates the need for long-distance deliveries from centralized storage.

The Assembly Line Feeding Problem (ALFP) was formalized in response to the emergence of mass customization. The supply of components to mixed-model assembly lines has proven to be a challenging endeavour, given that the quantity of parts needed for a single assembly line significantly increases.

Much effort has been made to model the complexities of assembly line feeding, as they occur in reality. Noteworthy extensions of the groundwork by (Bozer and McGinnis 1992) are the inclusion of feeding policies, vehicle types and route determination. However, this work is the first to consider multiple supermarkets; previous works on the ALFP only consider one supermarket since the problem is more concerned with material flows, route determination, loading and scheduling.

Feeding policies serve the purpose to minimise feeding costs while ensuring a timely material supply at the workstations. Feeding policies generally differ from one another, in the way they present parts at assembly stations; i.e. parts may be presented on homogenous pallets, part-variants might come type-sorted in boxes, heterogenous parts of an assembly-step may also come prepared in kits, which contain all necessary parts for a particular model at a workstation. We consider five distinct policies.

- i. Line stocking: parts are supplied in pallets directly from the warehouse to the assembly line.
- ii. Boxed-supply: parts are supplied in boxes, which contain single part-variants. They are usually fed from supermarkets to kanban racks at the BoL.
- iii. Sequencing: parts are supplied in a pre-determined order. In mixed model assembly lines, it is sensible to feed parts, which are unique to certain models, in the same order that the models arrive at the workstation.

- iv. Kitting: kitting in assembly lines involves the process of gathering and grouping together various parts or components needed for a specific task, often presented in a container or kit for streamlined assembly.
 - a. Stationary kitting: parts are presented in a fixed location.
 - b. Travelling kitting: parts are presented in mobile containers or racks that move with the product along the assembly line, facilitating part retrieval at multiple stations.

The consideration of vehicle types in the ALFP serves to minimize the transportation cost. In reality vehicle types may be constrained by on-site infrastructure and complexity of the transportation-process, they also have different operating costs. We consider three vehicle types.

- i. Forklift: High flexibility in part retrieval, low infrastructure requirements, medium capacity, and high operating cost due to human operator. Fork lifts are used for line-stocking; and serve single-stations from warehouse or supermarkets.
- ii. Tow train: Medium flexibility, high capacity, medium operating cost. They serve multiple stations (milk-run delivery)
- iii. Autonomous Guided Vehicle (AGV): Medium flexibility, medium capacity, and low operating cost due to driverless operation. They serve multiple stations (milk-run delivery)

Other vehicles, such as trucks, conveyors, cranes exist, and are not considered here.

Route determination considerations in the ALFP are necessary to estimate costs and make overall cost-optimal decisions. We consider three types of routes: (i) supermarkets to workstations; (ii) warehouse to workstations; (iii) warehouse to supermarkets.

This work aims to investigate the benefits achieved by considering multiple supermarkets, and how well these additional layout-aspects integrate into the existing model provided by Adenipekun et al. (2022). In summary our research questions can be formally formulated as follows:

- To what extent do multiple supermarkets lead to cost improvements?
- Which factors influence the selection of supermarket sites?
- What is the effect on model performance by this extension and how can performance be improved?

To answer those questions we extend and adapt the MILP model proposed by Adenipekun et al. (2022). We test and verify our model on small datasets and subsequently conduct our experiments on larger generated datasets, designed to incorporate the novel layout-aspects, which are not part of the seed paper.

The remainder of this seminar paper is structured as follows: Section 2 offers a review of related literature; section 3 provides a description of the problem; section 4 outlines the employed solution methods; in section 5 we present and discuss numerical experiments design, and subsequently computational and numerical results; finally in section 6, the main findings, limitations, and future research proposals are drawn.

2 Literature Review

The optimization based MILP in this work's seed paper by Adenipekun et al. (2022) aggregates the vast insights made by the previous research-body into one comprehensive model. The researchers highlight that bringing layout aspects into the ALFP is their greatest contribution to the field. Those were previously only part of research dedicated to the Supermarket Location Problem (SLP).

2.1 Disambiguation of the ALFP and the SLP

The Supermarket Location Problem (SLP) revolves around strategically positioning smaller decentralized logistic areas, known as "supermarkets." These supermarkets serve a crucial role in addressing spatial constraints and ensuring a seamless flow of components to the assembly line's "border of line" (BoL). The primary objective is to find optimal locations for these supermarkets to efficiently supply part racks at the BoL, addressing challenges such as space scarcity and considering factors such as part demand at the line's workstations, storage capacity of the station-racks, the proximity of the line-storage to potential supermarket locations.

The Assembly Line Feeding Problem (ALFP) delves into the logistical intricacies of material flow. This involves making decisions on scheduling, routing, and maintaining a continuous and timely supply of parts. The ALFP focuses on optimizing the flow of materials to the assembly line, a critical aspect in manufacturing settings where space scarcity may be addressed best with Just-in-Time (JIT) delivery.

Despite having different primary objectives, the SLP and ALFP exhibit a notable overlap. The efficient operating of supermarkets is central to addressing the challenges posed by assembly line spatial constraints. The benefits of integrating the two problems into a comprehensive model lie in optimizing the locations of supermarkets while also refining strategies for the systematic and timely feeding of materials, thus achieving a synergy. In line with this, Emde and Boysen (2012) emphasized the multifaceted nature of supermarket planning, highlighting several supermarket-related decision problems: (i) determining the number and locations of supermarkets (SLP); (ii) configuring number of tow trains and stations served, posing a Routing Problem (RP); (iii) establishing fixed delivery schedules (SP); (iv)

and selecting part bins for loading on tow train tours posing a Loading Problem (LP). Additionally, (v) selecting a vehicle mix can be considered a Vehicle Selection Problem (VSP).

As per the earlier definitions, some supermarket-related decision problems lean towards ALFP, while others align more with SLP. In that sense the ultimate objective would be a model that combines all, while solving the computational bottleneck associated with such a complex model.

As per the earlier definition, the ALFP considers the sub-problems (ii, iii and iv). The attention to the ALFP was first raised by Bozer and McGinnis (1992) through their descriptive model, aiding the decision between two feeding policies, namely line-stocking, travelling- and stationary-kitting, whilst considering handling costs, space-limitations and work-in-progress (WIP) . Major advances and extensions over the years were made by Battini et al. (2009, 2010b) which added storage and feeding costs. Faccio (2014) considered mixed-model assembly lines, different “bill-of-materials” (BOM) and the additional feeding policy boxed-supply. The classification of the research-body by Schmid and Limère (2019) revealed that logistics design had been largely disregarded.

On part of the SLP, which deals exclusively with logistics design in its puristic form Zhou and Tan (2020) developed a model for the choosing supermarket locations based on deterministic demands and distance-dependent transportation costs from a set of pre-defined supermarket locations. Nourmohammadi et al. (2018) addressed uncertain part-demands by using stochastic demands.

2.2 Combining the Assembly Line Feeding Problem with factory-layout aspects

Closely related to layouts is vehicle selection, due to the different infrastructure needed to move parts. Battini et al. (2015) and Nourmohammadi et al. (2021) developed models which combine the SLP with the Transport Vehicle Selection Problem (TSVP or VSP). This integration allows for a more comprehensive and realistic optimization of logistics and transportation in assembly lines, taking into account

the availability of different types of transport vehicles and their associated costs and efficiencies. However they do not address the ALFP.

Our seed paper (Adenipekun et al. 2022), integrates these aspects of vehicle efficiencies into the ALFP, by considering tow-trains, forklifts and AGVs, in addition to feeding policies, route determination, scheduling and importantly layout-aspects in form of travel-distances; this is a non-exhaustive listing. Adenipekun et al. (2022) the most comprehensive model to date, using branch and cut algorithm to solve it. However, in contrast to SLPs only one supermarket is considered.

Showcased in Table 1 is the research gap in regard to the ALFP with multiple supermarkets.

Table 1: Summary of the literature analysis

| Literature | ALFP | feeding policies | vehicle types | SLP | multiple supermarkets |
|-------------------------------|------|---------------------|------------------|-----|--------------------------|
| Adenipekun et al., 2021 ★ | x | x | x | | |
| Limère et al., 2012 | x | x | | | |
| Schmid et al., 2020 | x | x | | | |
| Bozer and McGinnis, 1992 | x | x | | | |
| Baller et al., 2020 | x | x | | | |
| Schmid and Limère, 2019 | x | x | | | |
| Battini et al., 2010 | x | x | | | |
| Battini et al., 2013 | x | x | | | |
| Müllerklein et al., 2022 | x | x | | | |
| Emde and Boysen, 2011 | x | | | | |
| Emde and Boysen, 2012 | x | | | | |
| Battini et al., 2009 | x | | | | |
| Golz et al., 2012 | x | | | | |
| Emde et al., 2012 | x | | x | | |
| Sali and Sahin, 2016 | | x | | | |
| Sternatz, 2015 | | x | | | |
| Fathi et al., 2020 | | (x) | x | x | x |
| Alnahhal and Noche, 2015 | (x) | | x | x | x |
| Nourmohammadi et al., 2021 | | | x | x | (x) |
| Nourmohammadi et al., 2018 | (x) | (x) | | x | x |
| Zhou and Tan, 2019 | | | | x | x |
| Reibold and Voigtländer, 2024 | x | x | x | x | x |

x: core aspect, implemented | (x): mentioned descriptively, not implemented | ★: our seed paper

3 Problem Description

The aim of this work is to extend an existing mathematical model by integrating additional functions. The publication by Adenipekun et al. (2022), which deals with the assembly line feeding problem (ALFP), serves as the starting point.

Adenipekun et al. (2022) have further developed the ALFP already known from the literature by adding some extensions. The ALFP is about the way in which the parts are brought to the production line and the respective station. A number of decisions can be made in this process, which are presented in more detail below. The initial model makes the following decisions:

1. Each part number is assigned to one of five feeding policies (line stocking, box stocking, sequencing, stationary and moving kits),
2. One of three vehicle types (VT) (forklift, tow train or automated guided vehicle (AGV)),
3. The exact route the vehicle will take to reach the station where the part will be installed,
4. The frequency / number of cycles in which the stations are approached are calculated.

The overall goal is to minimize all costs incurred, including transportation, replenishment, preparation, usage and procurement costs. The initial model is based on a number of assumptions, such as that demand is deterministic and not stochastic. These are adopted and are not discussed in detail. The research conducted by Adenipekun et al. (2022) looks exclusively at operational decisions that are only ever made for a short planning horizon. Figure 1 showcases certain layout, routing and vehicle selection considerations.

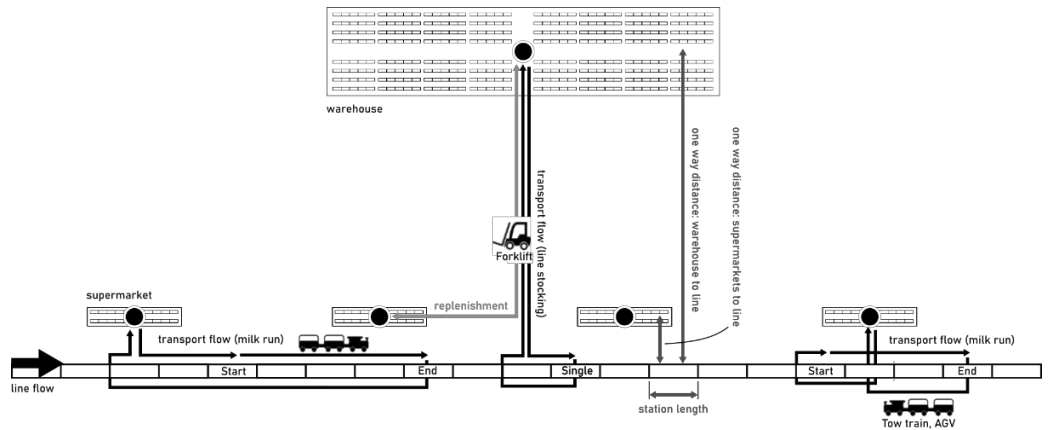


Figure 1: Layout, routing, and vehicle selection considerations

For a more detailed description of the problem and the decision features, we recommend the publication by Adenipekun et al. (2022).

Our approach extends the existing model by not only limiting the allocation of parts to feeding policies, vehicle types and routes, but also including the allocation and selection of supermarkets in the equations. In industry, supermarkets are buffer areas or small storage areas in which parts are stored within the production hall before they are transported to the respective station where they are assembled to the final product. This extension reflects reality, as it is common to have more than one supermarket on a production line. This means that each part is not only assigned a feeding policy, a vehicle type and a route, but also a specific supermarket.

This extension turns the originally purely operational decision problem into a tactical problem by integrating tactical decisions about the selection and location of supermarkets into the holistic design of a production chain and parts supply.

The extension aims to clarify under which conditions additional supermarkets make sense and can therefore contribute to a reduction in costs. Criteria such as the location of the potential supermarkets, the distance to the stations, the size of the production line, the capacity of the supermarkets and the costs are taken into account. The latter are made up of both personnel costs and size-dependent costs.

Table 2 summarizes and compares the decisions of the initial model and the extended model.

Table 2: Comparison of the model by Adenipekun et al. (2022) and the extended model

| Decisions | Adenipekun et al. (2022) | Reibold & Voigtländer (2024) |
|--------------------------|--------------------------|------------------------------|
| Feeding policies | x | x |
| Vehicle Types | x | x |
| Acquisition costs for VT | x | |
| Fleet size determination | x | |
| Route determination | x | x |
| Multiple supermarkets | | x |
| Supermarket capacities | | x |
| Multiple travelling kits | | x |

4 Methodology

As mentioned at the beginning, in this section, a mixed integer linear problem (MILP) is introduced. Due to a large number of indices, variables, parameters and equations, the developed model is introduced in individual subsections. The model is based on the model of Adenipekun et al. (2022) and has been supplemented by the extensions presented in the previous section.

4.1 Mixed Integer Linear Problem (MILP)

First, the objective function and its components are explained. Section 4.1.2 presents the equations for the assembly line feeding problem (ALFP) and the assignment of parts and stations to supermarkets. Then the constraints for determining the routes and the space requirements for each part on the assembly line are explained. Finally, the various cost parameters are calculated in individual subsections (chapter 4.2).

4.1.1 Objective function

The objective function attempts to minimize various costs and consists of six parts.

$$\begin{aligned}
 \text{Minimize } z = & \sum_{i \in \mathcal{I}} \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}} \sum_{b \in \mathcal{B}} c_{ipmb} x_{ipmb} + \\
 & \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}'} \sum_{t \in \mathcal{T}} \sum_{r \in \mathcal{R}} \sum_{w \in \mathcal{W}} \sum_{b \in \mathcal{B}} c_{pmtrwb} v_{pmtrwb} + \\
 & \sum_{f \in \mathcal{F}} \sum_{i \in \mathcal{I}: f_{i-1} \neq f_i} \sum_{m \in \mathcal{M}} \sum_{b \in \mathcal{B}} c_{fmb} x_{i3mb} + \\
 & \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} \sum_{b \in \mathcal{B}} c_{msb} k_{msb} + \\
 & \sum_{m \in \mathcal{M}} \sum_{b \in \mathcal{B}} c_{mb} t_{mb} + \\
 & \sum_{b \in \mathcal{B}} (SCcap_b + Fix_S) y_b
 \end{aligned} \tag{1}$$

The objective function minimizes transportation and replenishment costs, which arise when the supermarkets are supplied with new parts from the warehouse or parts are transported from the warehouse or a supermarket to the BoL. Another component are the preparation costs incurred when deliveries from the

supermarkets to the border of line (BoL) are prepared in one of the supermarkets and the usage costs which occur directly at the BoL and include searching and picking costs. These costs are all calculated in part 1-5 of the objective function for all feeding policies and vehicle types. They vary depending on the selected feeding policy. In the sixth part, the operating costs for the supermarkets are calculated. These costs include the wages for the production employees in the individual supermarkets and capacity-related costs for the supermarkets, which can be interpreted as rent.

4.1.2 General Constraints

$$\sum_{m \in \mathcal{M}} \sum_{p \in \mathcal{P}} \sum_{b \in \mathcal{B}} x_{ipmb} = 1 \quad \forall i \in \mathcal{I} \quad (2)$$

$$x_{ipmb} = x_{jpmb} \quad \forall f \in \mathcal{F}, i, j \in \mathcal{I}_f: i + 1 = j, p \in \mathcal{P}, m \in \mathcal{M}, b \in \mathcal{B} \quad (3)$$

$$x_{i2mb} = 0 \quad \forall a \in \mathcal{A}, i \in \mathcal{I}: r_{ia} > r_{a2}, m \in \mathcal{M}, b \in \mathcal{B} \quad (4)$$

$$\frac{n_4^k}{r_{a4}} \sum_{i \in \mathcal{I}_s} \frac{r_{ia} m_i}{|\mathcal{V}_i|} x_{i4mb} \leq k_{msb} \quad \forall s \in \mathcal{S}, m \in \mathcal{M}, a \in \mathcal{A}, b \in \mathcal{B} \quad (5)$$

$$\sum_{m \in \mathcal{M}} \sum_{b \in \mathcal{B}} k_{msb} \leq 1 \quad \forall s \in \mathcal{S} \quad (6)$$

$$\frac{n_5^t}{r_{a5}} \sum_{i \in \mathcal{I}} \frac{r_{ia} m_i}{|\mathcal{V}_i|} x_{i5mb} \leq t_{mb} \quad \forall m \in \mathcal{M}, a \in \mathcal{A}, b \in \mathcal{B} \quad (7)$$

$$\sum_{m \in \mathcal{M}} t_{mb} \leq 1 \quad \forall b \in \mathcal{B} \quad (8)$$

$$q_{sb} \leq y_b \quad \forall b \in \mathcal{B}, s \in \mathcal{S} \quad (9)$$

$$x_{ipmb} \leq y_b \quad \forall i \in \mathcal{I}, p \in \mathcal{P}, m \in \mathcal{M}, b \in \mathcal{B} \quad (10)$$

$$\sum_{i \in \mathcal{I}} \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}} x_{ipmb} \leq cap_b \quad \forall b \in \mathcal{B} \quad (11)$$

Constraint 2 ensures that each part is assigned to a feeding policy p , a vehicle type m and a supermarket b . In the case of parts that belong to the same part family, the vehicle type, feeding policy and supermarket are assigned together (constraint 3). Constraint 4 acts as a filter that excludes parts that are unsuitable for box feeding due to their size or weight.

Constraints 5 to 8 determine if and which parts are assigned to stationary or traveling kits. They also calculate the quantity of parts that can be included in a traveling kit. Parts and stations are only assigned to a supermarket if it is open and if there is sufficient capacity for the additional part (constraints 9-11). A more detailed description of the constraints can be found in the publication by Adenipekun et al. (2022).

Equations 12 to 21 define the value ranges for binary and continuous decision variables.

$$x_{ipmb} \in \{0,1\} \quad \forall i \in \mathcal{I}, p \in \mathcal{P}, m \in \mathcal{M}, b \in \mathcal{B} \quad (12)$$

$$v_{pmtrwb} \in \{0,1\} \quad \forall p \in \mathcal{P}, m \in \mathcal{M}', t \in \mathcal{T}, r \in \mathcal{R}, w \in \mathcal{W}, b \in \mathcal{B} \quad (13)$$

$$k_{msb} \in \{0,1\} \quad \forall m \in \mathcal{M}, s \in \mathcal{S}, b \in \mathcal{B} \quad (14)$$

$$t_{mb} \in \{0,1\} \quad \forall m \in \mathcal{M}, b \in \mathcal{B} \quad (15)$$

$$f_{ips} \in \mathbb{N}_0 \quad \forall i \in \mathcal{I}, p \in \{1,2\}, s \in \mathcal{S} \quad (16)$$

$$f_{fs} \in \mathbb{N}_0 \quad \forall f \in \mathcal{F}, s \in \mathcal{S} \quad (17)$$

$$f_s \in \mathbb{N}_0 \quad \forall s \in \mathcal{S} \quad (18)$$

$$r_s \in \mathbb{N}_0 \quad \forall s \in \mathcal{S} \quad (19)$$

$$y_b \in \{0,1\} \quad \forall b \in \mathcal{B} \quad (20)$$

$$q_{sb} \in \{0,1\} \quad \forall b \in \mathcal{B}, s \in \mathcal{S} \quad (21)$$

4.1.3 Route determination

Constraints 20 and 21 determine the fixed routes that are traveled by the vehicle types to the stations. The stations can be approached either directly from the warehouse or from an active supermarket. They also define the extent to which the station is supplied with parts. The supply frequency is directly dependent on the available storage space at the station. The corresponding equations are presented in the next subsection.

$$\sum_{t \in \mathcal{T}} \sum_{r \in \mathcal{R}_s} v_{pmtrwb} \leq q_{sb} \quad \forall b \in \mathcal{B}, p \in \mathcal{P}, m \in \mathcal{M}', s \in \mathcal{S}, w \in \mathcal{W}_p \quad (22)$$

$$\sum_{i \in \mathcal{I}_s} x_{ipmb} \leq \mathcal{M} \sum_{t \in \mathcal{T}} \sum_{r \in \mathcal{R}_s} v_{pmtrwb} \quad \forall b \in \mathcal{B}, p \in \mathcal{P}, m \in \mathcal{M}', s \in \mathcal{S}, w \in \mathcal{W}_p \quad (23)$$

4.1.4 Line-sided storage

The following equations (24 - 31) determine the proportion of available space per station that each part receives directly at the BoL storage location. This depends on the dimensions and requirements of the respective part and therefore influences the routes and the supply cycles of the stations with parts. If a large storage bin is reserved for a particular part number, this station is supplied with a lower frequency. In contrast, a part with identical requirements and limited storage space is supplied more frequently. This ensures the constant availability of sufficient parts and thus prevents production downtimes.

$$\sum_{b \in \mathcal{B}} \left(\frac{\lambda_i t}{H n_{ip} d_p} x_{ipmb} - \mathcal{M} \left(1 - \sum_{r \in \mathcal{R}_s} v_{pmtrwb} \right) \right) \leq f_{ips} \quad (24)$$

$$\forall p \in \{1,2\}, s \in \mathcal{S}, i \in \mathcal{I}_s, t \in \mathcal{T}, m \in \mathcal{M}', w \in \mathcal{W}_p$$

$$\frac{1}{h_2 w_2} \sum_{i \in \mathcal{I}_s} f_{i2s} \leq r_s \quad \forall s \in \mathcal{S} \quad (25)$$

$$\sum_{b \in \mathcal{B}} \left(\frac{\lambda_f t}{H n_f} x_{i3mb} - \mathcal{M} \left(1 - \sum_{r \in \mathcal{R}_s} v_{3mtr2b} \right) \right) \leq f_{fs} \quad (26)$$

$$s \in \mathcal{S}, f \in \mathcal{F}_s, i \in \mathcal{I}_f: f_{i-1} \neq f_i, t \in \mathcal{T}, m \in \mathcal{M}'$$

$$\sum_{b \in \mathcal{B}} \left(\frac{t}{n_4^k} k_{msb} - \mathcal{M} \left(1 - \sum_{r \in \mathcal{R}_s} v_{4mtr2b} \right) \right) \leq f_s \quad \forall s \in \mathcal{S}, t \in \mathcal{T}, m \in \mathcal{M}' \quad (27)$$

$$q_p \sum_{m \in \mathcal{M}} \sum_{b \in \mathcal{B}} x_{ipmb} \leq f_{ips} \quad \forall p \in \{1,2\}, s \in \mathcal{S}, i \in \mathcal{I}_s \quad (28)$$

$$q_3 \sum_{m \in \mathcal{M}} \sum_{b \in \mathcal{B}} x_{i3mb} \leq f_{fs} \quad \forall s \in \mathcal{S}, f \in \mathcal{F}_s, i \in \mathcal{I}_f: f_{i-1} \neq f_i \quad (29)$$

$$q_4 \sum_{m \in \mathcal{M}} \sum_{b \in \mathcal{B}} k_{msb} \leq f_s \quad \forall s \in \mathcal{S} \quad (30)$$

$$\sum_{i \in \mathcal{I}_s} L_1 f_{i1s} + L_2 r_s + \sum_{f \in \mathcal{F}_s} L_3 f_{fs} + L_4 f_s \leq L_s \quad \forall s \in \mathcal{S} \quad (31)$$

4.1.5 Fleet size determination

Due to the considerable complexity of the problem, no equations are used to determine the number of individual vehicle types. This decision is entirely justified, as the number of vehicles is used solely to calculate the acquisition costs. These costs represent one-off expenses that are difficult to break down into costs per shift. Such

a breakdown could distort the result and the total costs. It is therefore assumed that a sufficient number of vehicles of each type is always available.

4.2 Cost parameter calculation

This chapter presents equations that determine the costs incurred when delivering parts to the assembly line. These can be broken down into replenishment, inventory, transportation and usage costs and are explained in more detail below.

$$c_{ipmb} = c_{ipFb}^R + c_{ipmb}^P + c_{ipFb}^T + c_{ipmb}^U \quad \forall i \in \mathcal{I}, p \in \mathcal{P}, m \in \mathcal{M}, b \in \mathcal{B} \quad (32)$$

$$c_{pmtrwb} = c_{pmtrwb}^T \quad \forall p \in \mathcal{P}, m \in \mathcal{M}', t \in \mathcal{T}, r \in \mathcal{R}, w \in \mathcal{W}, b \in \mathcal{B} \quad (33)$$

$$c_{fmb} = c_{fmb}^T \quad \forall f \in \mathcal{F}, m \in \mathcal{M}, b \in \mathcal{B} \quad (34)$$

$$c_{msb} = c_{msb}^T \quad \forall m \in \mathcal{M}, s \in \mathcal{S}, b \in \mathcal{B} \quad (35)$$

$$c_{mb} = c_{mb}^T \quad \forall m \in \mathcal{M}, b \in \mathcal{B} \quad (36)$$

4.2.1 Replenishment costs

Equation 37 calculates the replenishment costs. These costs arise when parts are delivered from one of the supermarkets to the production line. In this case, the parts must first be transported from the warehouse to the supermarkets. This is where the replenishment costs occur.

$$c_{ipFb}^R = co_F \left[\frac{\lambda_i}{n_{i1}} \left(\frac{d_b^{ws}}{n_{1F} \mu_F v_F} + l_{1F} \right) \right] \quad \forall i \in \mathcal{I}, p \in \mathcal{P}_2, b \in \mathcal{B} \quad (37)$$

4.2.2 Preparation costs

Preparation costs arise when parts are provided by feeding policy line stocking, sequencing, stationary or traveling kits. In this case, the deliveries must be prepared by an employee in a supermarket. Parts have to be repacked from large boxes into small boxes, sorted into the correct order for several variants or bundled into kits. Equations 38 to 41 calculate the respective costs.

$$c_{i2mb}^P = c^l \left[\frac{\lambda_i}{n_{i2}} \left(\frac{AL_{2b}}{n_{tr2} ov} \right) + (ht_{i2} + s_{i2}) \lambda_i \right] \quad \forall i \in \mathcal{I}, m \in \mathcal{M}, b \in \mathcal{B} \quad (38)$$

$$c_{i3mb}^P = c^l \left[\frac{\lambda_i}{n_{i3}} \left(\frac{AL_{3b} + mv}{ov} \right) + (ht_{i3} + s_{i3}) \lambda_i \right] \quad \forall i \in \mathcal{I}, m \in \mathcal{M}, b \in \mathcal{B} \quad (39)$$

$$c_{i4mb}^P = c^l \left[\left(\frac{AL'_{4b} d}{ovn_4^k} \right) + (ht_{i4} + s_{i4}) \lambda_i \right] \quad \forall i \in \mathcal{I}, m \in \mathcal{M}, b \in \mathcal{B} \quad (40)$$

$$c_{i5mb}^p = c^l \left[\left(\frac{AL'_{5b}d}{ovn_5^t} \right) + (ht_{i5} + s_{i5})\lambda_i \right] \quad \forall i \in \mathcal{I}, m \in \mathcal{M}, b \in \mathcal{B} \quad (41)$$

4.2.3 Transportation costs

Transportation costs, as the name suggests, include the cost of transporting parts from the warehouse or one of the supermarkets to the stations. The costs are subject to variations depending on the chosen feeding policy, vehicle type, route selection and frequency of deliveries (see equations 42-47).

$$c_{i1mb}^T = co_m \left[\frac{\lambda_i}{n_{i1}} \left(\frac{d_s^w}{n_{1m}\mu_m v_m} + l_{1m} \right) \right] \quad \forall s \in \mathcal{S}, i \in \mathcal{I}_s, m \in \mathcal{M}, b \in \mathcal{B} \quad (42)$$

$$c_{i2mb}^T = co_m \left[\frac{\lambda_i}{n_{i2}} \left(\frac{d_{bs}^s}{n_{2m}\mu_m v_m} + l_{2m} \right) \right] \quad \forall s \in \mathcal{S}, i \in \mathcal{I}_s, m \in \mathcal{M}, b \in \mathcal{B} \quad (43)$$

$$c_{fmb}^T = co_m \left[\frac{\lambda_f}{n_f} \left(\frac{d_{bs}^s}{n_{3m}\mu_m v_m} + l_{3m} \right) \right] \quad \forall s \in \mathcal{S}, f \in \mathcal{F}_s, m \in \mathcal{M}, b \in \mathcal{B} \quad (44)$$

$$c_{msb}^T = co_m \left[\frac{d}{n_4^k} \left(\frac{d_{bs}^s}{n_{4m}\mu_m v_m} + l_{4m} \right) \right] \quad \forall s \in \mathcal{S}, m \in \mathcal{M}, b \in \mathcal{B} \quad (45)$$

$$c_{mb}^T = co_m \left[\frac{d}{n_5^t} \left(\frac{d_{b1}^s}{n_{5m}\mu_m v_m} + l_{5m} \right) \right] \quad \forall m \in \mathcal{M}, b \in \mathcal{B} \quad (46)$$

$$c_{pmtwrb}^T = co_m \left[\frac{Hmr_{rw}b}{t\mu_m v_m} \right] \quad \forall p \in \mathcal{P}, m \in \mathcal{M}', t \in \mathcal{T}, r \in \mathcal{R}, w \in \mathcal{W}_p, b \in \mathcal{B} \quad (47)$$

4.2.4 Usage costs

Usage costs are costs that occur directly on the production line. These include walking distance and time as well as search time, which depends on how the parts are provided (which feeding policy was assigned) (constraint 48 – 49).

$$c_{ipm}^U = c^a \lambda_i \left(\frac{wd_p}{ov} + s_{ip} \right) \quad \forall i \in \mathcal{I}, p \in \{1,2\}, m \in \mathcal{M} \quad (48)$$

$$c_{ipm}^U = c^a \lambda_i \frac{wd_p}{ov} \quad \forall i \in \mathcal{I}, p \in \{3,4\}, m \in \mathcal{M} \quad (49)$$

5 Results

In order to evaluate the extended model, presented in the previous chapter, numerical experiments are carried out. These are intended to clarify the research questions of whether several supermarkets have cost advantages over one.

A series of experiments will demonstrate that choosing multiple supermarkets over a single one can help to reduce overall costs. However, it should be noted that this finding does not apply equally to all production lines. Chapter 5.2 provides a detailed analysis of the costs associated with varying numbers of stations and supermarkets. Chapter 5.3 deals with a comparison of having one or two supermarkets with different labor costs and locations. Given the size of the model and the large number of variables, Chapter 5.4 examines whether it makes sense to define the feed strategy for the part numbers in advance in order to shorten the calculation time.

All tests were performed on a macOS 12.7.2 based computer with 8 GB RAM and a 2.2 GHz dual-core Intel Core i7 processor, running Python 3.9.18 with the commercial solver Gurobi Optimizer 10.0.3.

5.1 Numerical Experiments Design – Input Values

The majority of the input values were taken from the publication by Adenipekun et al. (2022) and left unchanged. Table 3 defines the values of the additional parameters. Furthermore, the ranges of supermarkets, stations, capacities, costs and distances are given, which were used for the experiments. The distances are the distances between the warehouse, supermarkets and stations.

Table 3: Additional parameters

| Parameter | Range | Details |
|------------------------|----------------------------------|--|
| # Supermarket | 1-4 | Number of supermarkets next to the production line |
| # Stations | 10-40 | Number of stations at the production line |
| Capacities | Depending on the number of parts | Supermarket capacities |
| Fixed costs | 10 – 40[€] | Interpretable as hourly wage for logistic operator |
| Capacity related costs | 0.5 – 4 [€] | Interpretable as rent |
| Distances | 5-450[m] | Distances between warehouse, supermarkets and stations |
| Planning horizon | 20 | takt cycles |

As mentioned before, the large problem size results in the fact that no optimal solution can be found in a reasonable amount of time. For this reason, a time limit of 1000 seconds was set for all experiments. The objective function values are therefore always the best solution after 1000 seconds.

5.2 Advantage of several supermarkets

As part of the first experiment, the total costs were analyzed as a function of the number of stations and the number of active supermarkets. It was determined that with an increasing number of stations and thus an increased number of part numbers that have to be provided, the total costs increase. This increase can be attributed to all components of the objective function.

Interestingly, the analysis shows that with an increasing number of stations, the total costs are lower when activating several supermarkets compared to activating only one supermarket. With 10 stations it is most cost-effective to store parts in only one supermarket, with 20 stations in two supermarkets, with 30 stations in 3 supermarkets and with 40 stations in 4 supermarkets. Despite an increase in fixed costs due to the activation of several supermarkets, transportation costs decrease significantly. This results from the shorter distance between each supermarket and the respective station due to the strategic allocation of the stations to the nearest supermarket (see Figure 2).

These cost advantages are sometimes only minor, but this is due to the fact that the optimization was only carried out for a few takt cycles.

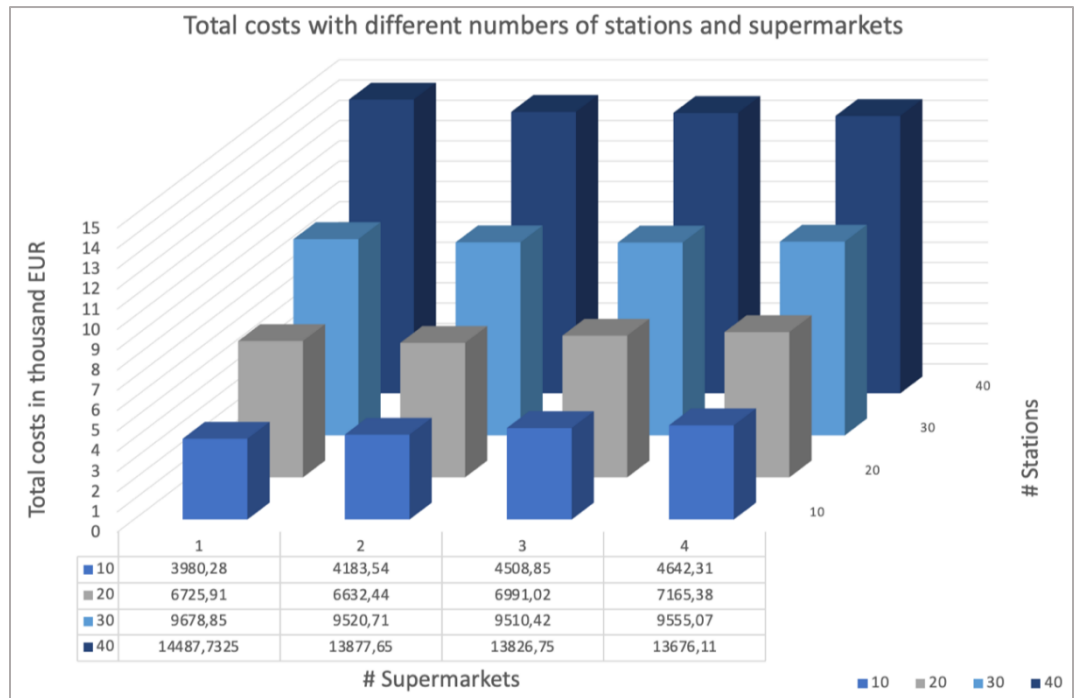


Figure 2: Total costs for different numbers of stations and supermarkets

5.3 Variable distances and supermarket fix costs

As explained in chapter 5.2, the fixed costs associated with the use of multiple supermarkets can be reduced by strategically minimizing the distance between the supermarkets and the outbound routes, which leads to cost advantages. The following section analyzes at which fixed costs (hourly labour cost) the benefits of shorter transport routes become negligible.

The fixed costs were systematically varied in the range from 20 € to 50 €. At labour costs between 40 and 50 €, it was found that the personnel costs reach a level at which the advantages of shorter transport routes are no longer recognizable (see Figure 3). This marks a critical threshold at which the trade-off process between fixed costs and personnel costs is particularly important for the precise optimization of cost-efficient logistics strategies.

This result suggests operating more supermarkets in regions and countries with lower labor costs and optimizing the storage of parts in supermarkets in countries with comparatively higher wages.

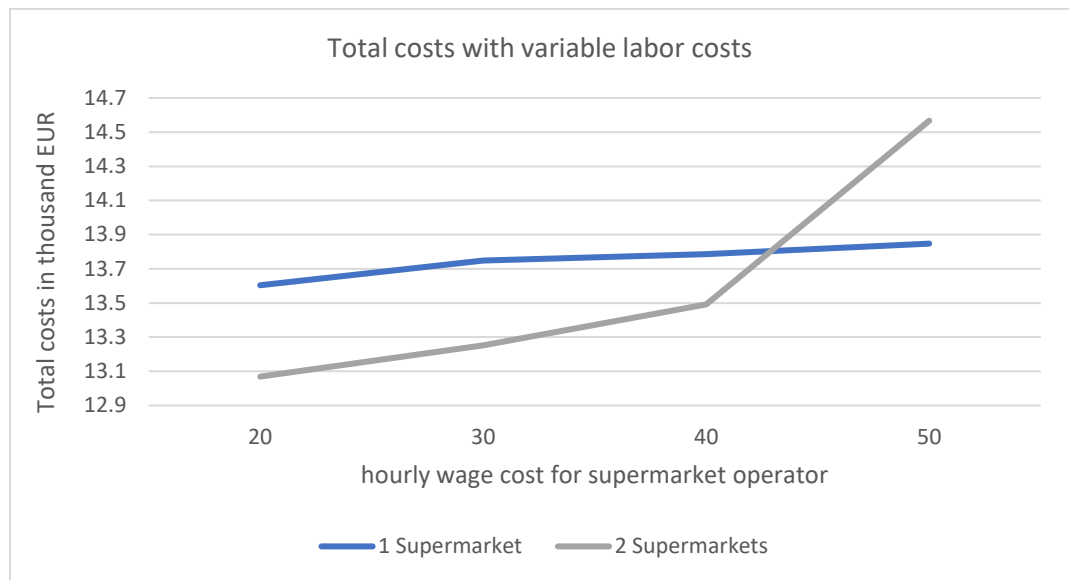


Figure 3: Total costs for different numbers of stations and supermarkets

A further insight was gained by changing the position of the supermarkets along the production line. Figure 4 shows four different location possibilities for one and two supermarkets. In the two options on the right, all supermarkets are located in the middle of the production line, while the supermarkets on the left are located at the beginning (and end) of the production line. A production line with 30 stations was considered for these experiments. It can be determined that the lowest costs (€8.698,41) are achieved when two supermarkets are used far apart from each other at the beginning and end of the line. In this scenario, the stations are assigned to the supermarkets in such a way that each station is always assigned to the nearest supermarket. This means that shorter distances have to be covered for partial delivery. Two supermarkets, both very centrally located in relation to the production line, generate only slightly higher costs of €8,704.25.

If there is only one supermarket, the correct location of the supermarket is even more important. Placing it at the beginning or end of the production line leads to costs of €9,834.96 in this example. If the supermarket is placed in the middle of the line, the costs can be reduced by approximately 7.7% to €9,079.32. This can also be explained by shorter transportation routes.

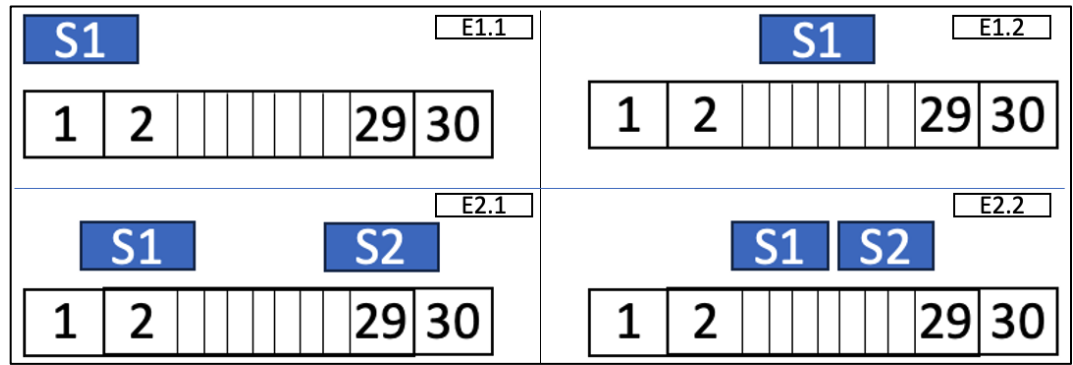


Figure 4: Different locations of one and two supermarkets at the production line

After carrying out these two tests, it can be concluded that the selection of several supermarkets is no longer advantageous if labor costs are too high. In addition, the location of the supermarkets also plays an important role, which can lead to significantly higher costs if the location is inappropriate.

5.4 Feeding policies pre-assigned vs. assigned during optimization

Given the substantial scale of the model and the extensive array of variables, the current investigation delves into whether the computational time can be diminished through the predefined feeding policy assigned to each part number. With this approach, each part number is only assigned to a specific vehicle type and a single supermarket. This methodology finds practical application, particularly due to the minimal alterations observed in parts over numerous life cycles, resulting in a stable supply structure within companies extending over decades.

The feeding policy is delineated by the following rules (see table 4):

Table 4: Rules for manual assignment of feeding policies in datasets

| Policy | Index | Rule |
|--------------------|---------|--|
| Line stocking | 1 (i) | For “singleton parts”: If a part exceeds a volume of 0.5 m ³ or a mass of 8kg the part will be assigned to policy 3 |
| Boxed supply | 2 (ii) | For “singleton parts”: If a part is not assigned a feeding policy by the previous rules it will be assigned to policy 2 |
| Sequencing | 3 (iii) | For “part-variants”: If multiple parts at a station exist, of the same part-family, those parts will be assigned policy 3. This leaves only parts, which are alone in their part family: “singleton parts”. |
| Stationary kitting | 4 (iv) | For “singleton parts”: If all parts at a station are “singleton parts” all parts at that station will be assigned to policy 4, randomly with $\frac{1}{3}$ probability. |

Due to the many correlations that exist in the allocation of parts to feeding policy 5, these have been omitted and no rules have been defined.

Even when conducting a relatively small experiment with only 10 stations, the advantages and disadvantages of pre-assigning of the feeding policy can be recognized.

Table 5 compares an overview of 3 optimization runs with the objective value, the gap to the best lower bound and the duration until a solution was found. Experiment 1 was the optimization with the model presented in chapter 4. No optimal solution could be found within the predefined time limit of 600 seconds. Nevertheless, a very good solution was found with a very small gap to the best lower bound and an objective value of €3,505.12. If feeding policies are assigned to the parts in advance according to the rules defined in Table 4, an optimal solution can be found after just 14.64 seconds. However, with an objective value of €9,932.93, this is almost three times as bad.

Due to the very poor value, the correctness of the code was checked by assuming that the solution from experiment 1 and the assigned feeding policies are known in advance. Here, exactly the same objective value of €3,505.32 was obtained. This confirms the correctness of the code.

Consequently, it can be inferred that the optimal assignment of supermarkets and vehicle types remains contingent on the stipulated feeding policy. In the course of further analysis, precise selection criteria can be refined and optimized to enhance the allocation of underfeeding strategies.

Table 5: Comparison of the performance between introduced model and model with pre-assigned feeding policies

| Model | Z(€) | GAP(%)⁽⁴⁾ | CPU(s)⁽⁵⁾ |
|---|-------------|-----------------------------|-----------------------------|
| Extended Model ⁽¹⁾ | 3.505,12 | 16,80 | Time limit reached |
| Feeding policy p pre-assigned ⁽²⁾ | 9.932,93 | Optimal solution | 14,64 |
| Feeding policy p pre-assigned V2 ⁽³⁾ | 3.505,12 | Optimal solution | 16,2 |

⁽¹⁾ Introduced in chapter 4

⁽²⁾ Assigning rule from table 4

⁽³⁾ Assignment of feeding policy p from solution of the extended model⁽¹⁾

⁽⁴⁾ Compared to best lower bound

⁽⁵⁾ Time limit set to 600 second

6 Conclusion

In this paper, we extend a mathematical model for the assembly line feeding problem. The existing model by Adenipekun et al. (2022) already considers several extensions in solving the problem. It assigns a specific feeding policy and vehicle type to each part. It also determines how often a station is supplied with new parts and which routes the vehicles take.

To add a decision on a tactical level, we expand the model to include several supermarkets. This means that the parts are also assigned to a set of supermarkets. The aim of the expansion is also to minimize the costs, which are made up of several components, including the costs of operating the supermarkets.

We implement the extended model and run several experiments to compare performance and to find the assignment combinations with the lowest cost. It was shown that the cost advantages of additional supermarkets come into play with an increasing number of station and thus the size of the production. For very small production lines, only one supermarket is still recommended.

In addition, it was found that with rising labor costs, the addition of a second or even more supermarkets is no longer economically viable. The limit is between €40 and €50. The location of the supermarkets also plays an important role and, regardless of the number of supermarkets, should always be positioned in such a way that the total distance to all stations is as short as possible.

The size of the model and the many options and parameters offer not only advantages, but also a major disadvantage. It is not possible to find an optimal solution in an acceptable amount of time. By defining precise rules for assigning feeding policies to parts in advance, it was possible to determine that an optimal solution can be found in a very short time. However, this solution is far from optimal and differs significantly from the solution that is found after 10 minutes without assigning the feeding policy in advance. Nevertheless, this is a very good approach to drastically reduce the optimization time. This should be pursued further, and the selection rules should be optimized in further analyses.

Furthermore, the following extensions should be considered in order to make the problem even closer to reality:

- More than one travelling kit per supermarket should be allowed. Currently, the number of traveling kits is limited to one per supermarket.
- The model is a deterministic model that does not allow for short-term fluctuations or product interruptions. Due to a very fast-moving world and production environment, the model should also be able to compensate for short-term changes and provide an optimal response to them.
- In addition, the route definition should be changed, and it should not be assumed that the production line is a straight line. The state of the art today are S- and U-shape production lines.

However, the critical point is still that the model already contains many aspects, and any additional extensions slow down the solution of the problem.

To overcome this challenge, a twofold approach is proposed. First, ways to decompose the model into different components need to be explored to enable multi-criteria optimization of each part. Secondly, the consideration of heuristic approaches is crucial. These heuristics can strategically optimize smaller sub-problems in isolation, whose optimized solutions can then be used as input variables in the higher-level overall model. This dual strategy aims to achieve a balance between realism and computational efficiency when tackling complex production and logistics planning tasks.

The extended model now reflects real-world scenarios and addresses challenges in tactical production and logistics planning, while retaining its abstract nature. When considering additional extensions to improve realism, it is important to recognize that the current model has many facets. Further extensions pose the challenge of potentially complicating problem solving and solution finding.

Reference List

- Adenipekun EO, Limère V, Schmid NA (2022) The impact of transportation optimisation on assembly line feeding. *Omega*. 107:102544.
- Alnahhal M, Noche B (2015) A genetic algorithm for supermarket location problem. *Assembly Automation*. 35(1):122–127.
- Baller R, Hage S, Fontaine P, Spinler S (2020) The assembly line feeding problem: An extended formulation with multiple line feeding policies and a case study. *International Journal of Production Economics*. 222:107489.
- Battini D, Faccio M, Persona A, Sgarbossa F (2009) Design of the optimal feeding policy in an assembly system. *International Journal of Production Economics*. 121(1):233–254.
- Battini D, Faccio M, Persona A, Sgarbossa F (2010a) Framework to optimise the inventory centralisation/ decentralisation degree and feeding policy in assembly systems. *IJSOM*. 6(2):184.
- Battini D, Faccio M, Persona A, Sgarbossa F (2010b) “Supermarket warehouses”: stocking policies optimization in an assembly-to-order environment. *Int J Adv Manuf Technol*. 50(5-8):775–788.
- Battini D, Gamberi M, Persona A, Sgarbossa F (2015) Part-feeding with supermarket in assembly systems: transportation mode selection model and multi-scenario analysis. *Assembly Automation*. 35(1):149–159.
- Bozer YA, McGinnis LF (1992) Kitting versus line stocking: A conceptual framework and a descriptive model. *International Journal of Production Economics*. 28(1):1–19.
- Emde S, Boysen N (2011) Optimally routing and scheduling tow trains for JIT-supply of mixed-model assembly lines. *European Journal of Operational Research*. Forthcoming.
- Emde S, Boysen N (2012) Optimally locating in-house logistics areas to facilitate JIT-supply of mixed-model assembly lines. *International Journal of Production Economics*. 135(1):393–402.
- Emde S, Fliedner M, Boysen N (2012) Optimally loading tow trains for just-in-time supply of mixed-model assembly lines. *IIE Transactions*. 44(2):121–135.
- Faccio M (2014) The impact of production mix variations and models varieties on the parts-feeding policy selection in a JIT assembly system. *Int J Adv Manuf Technol*. 72(1-4):543–560.

- Fathi M, Nourmohammadi A, Ghobakhloo M, Yousefi M (2020) Production Sustainability via Supermarket Location Optimization in Assembly Lines. *Sustainability*. 12(11):4728.
- Golz J, Gujjula R, Günther H-O, Rinderer S, Ziegler M (2012) Part feeding at high-variant mixed-model assembly lines. *Flex Serv Manuf J*. 24(2):119–141.
- Limère V, van Landeghem H, Goetschalckx M, Aghezzaf E-H, McGinnis LF (2012) Optimising part feeding in the automotive assembly industry: deciding between kitting and line stocking. *International Journal of Production Research*. 50(15):4046–4060.
- Müllerklein D, Fontaine P, Ostermeier F (2022) Integrated consideration of assembly line scheduling and feeding: A new model and case study from the automotive industry. *Computers & Industrial Engineering*. 170:108288.
- Nourmohammadi A, Eskandari H, Fathi M, Aghdasi M (2018) A mathematical model for supermarket location problem with stochastic station demands. *Procedia CIRP*. 72:444–449.
- Nourmohammadi A, Eskandari H, Fathi M, Ng AH (2021) Integrated locating in-house logistics areas and transport vehicles selection problem in assembly lines. *International Journal of Production Research*. 59(2):598–616.
- Sali M, Sahin E (2016) Line feeding optimization for Just in Time assembly lines: An application to the automotive industry. *International Journal of Production Economics*. 174:54–67.
- Schmid NA, Limère V (2019) A classification of tactical assembly line feeding problems. *International Journal of Production Research*. 57(24):7586–7609.
- Schmid NA, Limère V, Raa B (2021) Mixed model assembly line feeding with discrete location assignments and variable station space. *Omega*. 102:102286.
- Sternatz J (2015) The joint line balancing and material supply problem. *International Journal of Production Economics*. 159:304–318.
- Zhou B-H, Tan F (2020) A self-adaptive estimation of distribution algorithm with differential evolution strategy for supermarket location problem. *Neural Comput & Applic*. 32(10):5791–5804.

Appendices

Appendix I: Full space model code file (MILP.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jan 9 11:00:00 2024

@author: Felix Reibold, Steffen Voigtlaender
"""

import gurobipy as gp
import numpy as np
import pandas as pd
import openpyxl
import time

#####

# Create output
def Output(m):
    # Print the result
    status_code = {1: 'LOADED', 2: 'OPTIMAL', 3: 'INFEASIBLE', 4: 'INF_OR_UNBD',
5: 'UNBOUNDED', 9: 'TIME_LIMIT'}

    status = m.status

    print('The optimization status is ' + status_code[status])
    if status == 2 or status == 9:
        # Retrieve variables value
        print('Optimal solution:')
        for v in m.getVars():
            print(str(v.varName) + " = " + str(v.x))
        print('Optimal objective value: ' + str(m.objVal) + "\n")

# Define MILP Model
def Model_1_TC(H, muy_m, v_m, co_m, Cap_b, SC, mr_rb, r_a2, r_ia, nk_4, r_a4, m_i,
V_i, r_a5, nt_5, l_pF, n_ip, n_pm,
demand_i, dWS_b, l_pm, dW_s, dS_sb, station_families, n_f, de-
mand_f, ca, ov, s_ip, wd_p, cl, AL_pb,
ntr_2, ht_ip, mv, ALk_pb, q_p, families_parts, L_p, L_s, h_2, w_2,
BIG_M_route, mrl_r, mr2_rb,
BIG_M_storage, d_p):
    start_time = time.time()
    # Create the optimization model
    model = gp.Model()
    # Set parameters
    model.setParam('OutputFlag', True)
    # model.setParam('TimeLimit', 1000)

    # Parts
    I = len(assignment_parts_stations)
    # Supermarkets
    B = len(Cap_b)
    # Three vehicle types: forklift, tow train and AGV
    M = 3
    # Two attributes: Size(volume) and weight
    A = 2
    # Possible routes
    R = len(df)
    # Stations
    S = stations
    # Families
    F = int(assignment_parts_stations[-1, 3])
    # Feeding policies (line stocking, boxed supply, sequencing, stationary and
travelling kits)
    P = 5
    # Planning horizon
    H = 20
    T = H
    # product demand
    d = H
    # Transportation flows (warehousel - station & supermarket - station)
    W = 2
    W_p = {1: 1, 2: 2, 3: 2, 4: 2, 5: 2}
```

```

# Define decision variables
v_pmtrwb = model.addVars(P, M, T, R, W, B, lb=0, vtype=gp.GRB.BINARY,
name='decision_frequency')
q_sb = model.addVars(S, B, lb=0, vtype=gp.GRB.BINARY, name='assignment_super-
markets')
k_msb = model.addVars(M, S, B, lb=0, vtype=gp.GRB.BINARY, name='station-
ary_kit_vehicle')
t_mb = model.addVars(M, B, lb=0, vtype=gp.GRB.BINARY, name='travelling_kit_ve-
hicle')
f_ips = model.addVars(I, P, S, lb=0, vtype=gp.GRB.INTEGER, name='num-
ber_of_facings_f_ips')
f_fs = model.addVars(F, S, lb=0, vtype=gp.GRB.INTEGER, name='number_of_fac-
ings_f_fs')
f_s = model.addVars(S, lb=0, vtype=gp.GRB.INTEGER, name='number_of_fac-
ings_f_s')
r_s = model.addVars(S, lb=0, vtype=gp.GRB.INTEGER, name='number_of_fac-
ings_r_s')
c_ipmb = model.addVars(I, P, M, B, lb=0, vtype=gp.GRB.CONTINUOUS,
name='cost_general')
cR_ipmb = model.addVars(I, P, M, B, lb=0, vtype=gp.GRB.CONTINUOUS, name='re-
plenishment_cost')
cP_ipmb = model.addVars(I, P, M, B, lb=0, vtype=gp.GRB.CONTINUOUS, name='prep-
aration_cost')
cT_ipmb = model.addVars(I, P, M, B, lb=0, vtype=gp.GRB.CONTINUOUS,
name='transportation_cost')
cU_ipmb = model.addVars(I, P, M, B, lb=0, vtype=gp.GRB.CONTINUOUS, name='us-
age_cost')
c_pmtrwb = model.addVars(P, M, T, R, W, B, lb=0, vtype=gp.GRB.CONTINUOUS,
name='cost_route')
cT_pmtrwb = model.addVars(P, M, T, R, W, B, lb=0, vtype=gp.GRB.CONTINUOUS,
name='transportation_cost_route')
c_fmb = model.addVars(F, M, B, lb=0, vtype=gp.GRB.CONTINUOUS, name='cost_se-
quencing')
cT_fmb = model.addVars(F, M, B, lb=0, vtype=gp.GRB.CONTINUOUS, name='transpor-
tation_cost_sequencing')
c_msb = model.addVars(M, S, B, lb=0, vtype=gp.GRB.CONTINUOUS, name='cost_sta-
tionary')
cT_msb = model.addVars(M, S, B, lb=0, vtype=gp.GRB.CONTINUOUS, name='transpor-
tation_cost_stationary')
c_mb = model.addVars(M, B, lb=0, vtype=gp.GRB.CONTINUOUS, name='cost travel-
ling')
cT_mb = model.addVars(M, B, lb=0, vtype=gp.GRB.CONTINUOUS, name='transporta-
tion_cost_travelling')
y_b = model.addVars(B, lb=0, vtype=gp.GRB.BINARY, name='activation_supermar-
ket')
x_ipmb = model.addVars(I, P, M, B, lb=0, vtype=gp.GRB.BINARY, name='assign-
ment_parts')

# Constraints
# General constarints
model.addConstrs(
    gp.quicksum(x_ipmb[i, p, m, b] for p in range(P) for m in range(M) for b
in range(B)) == 1 for i in range(I))

'''
Constraint only active if feeding policy is pre-assigned
#model.addConstrs(gp.quicksum(x_ipmb[i, int(assignment_parts_stations[i, 12] -
1), m, b] for m in range(M) for b in range(B)) == 1 for i in range(I))
'''

model.addConstrs(
    x_ipmb[i, 1, m, b] == 0
    for a in range(A)
    for i in range(I)
    if r_ia[i, a] > r_a2[a]
    for m in range(M)
    for b in range(B))
model.addConstrs(
    (nk 4 / r_a4[a]) * gp.quicksum(
        ((r_ia[i, a] * m_i[i]) / v_i[i]) * x_ipmb[i, 3, m, b]
        for i, row in enumerate(map(tuple, assignment_parts_sta-
tions.tolist()))
        if row[1] == s + 1
    ) <= k_msb[m, s, b]
    for s in range(S)
    for m in range(M)
    for a in range(A)
    for b in range(B))
model.addConstrs(

```

```

gp.quicksum(k_msb[m, s, b] for m in range(M)) <= q_sb[s, b]
for b in range(B)
for s in range(S))
model.addConstrs(
    (nt_5 / r_a5[a]) * gp.quicksum(
        ((r_ia[i, a] * m_i[i]) / V_i[i]) * x_ipmb[i, 4, m, b]
        for i in range(I)
    ) <= t_mb[m, b]
    for s in range(S)
    for m in range(M)
    for a in range(A)
    for b in range(B))
model.addConstrs(gp.quicksum(t_mb[m, b] for m in range(M)) <= 1 for b in
range(B))
model.addConstrs(
    x_ipmb[i, p, m, b] <= y_b[b] for i in range(I) for b in range(B) for p in
range(P) for m in range(M))
model.addConstrs((x_ipmb[i, p, m, b] <= q_sb[s, b] for s in range(S) for i,
row in
    enumerate(map(tuple, assignment_parts_stations.tolist())) if
row[1] == s + 1 for b in range(B) for
    p in range(P) for m in range(M)))
model.addConstrs(q_sb[s, b] <= y_b[b] for s in range(S) for b in range(B))
model.addConstrs(
    gp.quicksum(x_ipmb[i, p, m, b] for i in range(I) for p in range(P) for m
in range(M)) <= Cap_b[b] for b in
    range(B))
model.addConstrs(
    x_ipmb[i, p, m, b] == x_ipmb[j, p, m, b]
    for f in range(F)
    for b in range(B)
    for p in range(P)
    for m in range(M)
    for i in range(I) if assignment_parts_stations[i, 3] == f + 1
    for j in range(i + 1, I)
    if assignment_parts_stations[i, 3] == assignment_parts_stations[j, 3])

# Route Determination
model.addConstrs(gp.quicksum(v_pmtrwb[p, m, t, r, w, b] for t in range(T) for
r in
    df[df['Route'].apply(lambda route: s + 1 in
route)].index) <= q_sb[s, b] for b in
    range(B) for s in range(S) for p in range(P) for m in [1, 2]
for w in range(W_p[p + 1]))
model.addConstrs(
    (gp.quicksum(
        x_ipmb[i, p, m, b]
        for i, row in enumerate(map(tuple, assignment_parts_sta-
tions.tolist()))
        if row[1] == s + 1
    ) <= BIG_M_route * (
        gp.quicksum(
            v_pmtrwb[p, m, t, r, w, b]
            for t in range(T)
            for r in df[df['Route'].apply(lambda route: s + 1 in route)].in-
dex))
        for p in range(P)
        for m in [1, 2]
        for s in range(S)
        for b in range(B)
        for w in range(W_p[p + 1]))

# Line-sided storage
model.addConstrs(
    (gp.quicksum(
        (((demand_i[i] * t) / (H * n_ip[i, p] * d_p[p])) * x_ipmb[i, p, m,
b])) - BIG_M_storage + (
            BIG_M_storage * (gp.quicksum(
                v_pmtrwb[p, m, t, r, w, b] for r in df[df['Route'].ap-
ply(lambda route: s + 1 in route)].index)))
            for b in range(B)
        ) <= f_ips[i, p, s])
    for p in [0, 1]
    for s in range(S)
    for i, row in enumerate(map(tuple, assignment_parts_stations.tolist())) if
row[1] == s + 1
    for t in range(T)
    for m in [1, 2]
    for w in range(W_p[p + 1]))
model.addConstrs(

```

```

        ((1 / (h_2 * w_2)) * gp.quicksum(
            f_ips[i, 1, s] for i, row in enumerate(map(tuple, assign-
ment_parts_stations.tolist())) if row[1] == s + 1)
            <= r_s[s]
            for s in range(S)))
    model.addConstrs(
        (gp.quicksum(
            (((demand_f[f] * t) / (H * n_f[f])) * x_ipmb[i, 2, m, b])) -
BIG_M_storage + (BIG_M_storage * (gp.quicksum(
            v_pmtrwb[2, m, t, r, 1, b] for r in df[df['Route']].apply(lambda
route: s + 1 in route).index)))
            for b in range(B)
            ) <= f_fs[f, s])
            for s in range(S)
            for f in station_families[s + 1]
            for i in families_parts[f + 1] if
            i > 0 and (i == 1 or assignment_parts_stations[i - 1, 3] != assign-
ment_parts_stations[i, 3]))
            for t in range(T)
            for m in [1, 2])
    model.addConstrs(
        (gp.quicksum(
            ((t / nk_4) * k_msb[m, s, b]) - BIG_M_storage + (BIG_M_storage *
gp.quicksum(
            v_pmtrwb[3, m, t, r, 1, b] for r in df[df['Route']].apply(lambda
route: s + 1 in route).index)))
            for b in range(B)
            ) <= f_s[s])
            for s in range(S)
            for t in range(T)
            for m in [1, 2])
    model.addConstrs(
        (q_p[p] * gp.quicksum(x_ipmb[i, p, m, b] for m in range(M) for b in
range(B)) <= f_ips[i, p, s]
            for p in [0, 1]
            for s in range(S)
            for i, row in enumerate(map(tuple, assignment_parts_stations.tolist()))
if row[1] == s + 1))
    model.addConstrs(
        (q_p[2] * gp.quicksum(x_ipmb[i, 2, m, b] for m in range(M) for b in
range(B)) <= f_fs[f, s]
            for s in range(S)
            for f in station_families[s + 1]
            for i in families_parts[f + 1] if
            i > 0 and (i == 1 or assignment_parts_stations[i - 1, 3] != assign-
ment_parts_stations[i, 3]))
    model.addConstrs(
        (q_p[3] * gp.quicksum(k_msb[m, s, b] for m in range(M) for b in range(B))
<= f_s[s]
            for s in range(S)))
    model.addConstrs(
        (gp.quicksum(L_p[0] * f_ips[i, 0, s] for i, row in enumerate(map(tuple,
assignment_parts_stations.tolist())) if
            row[1] == s + 1)
            + L_p[1] * r_s[s]
            + gp.quicksum(L_p[2] * f_fs[f, s] for f in station_families[s + 1])
            + L_p[3] * f_s[s]
            <= L_s
            for s in range(S)))

    # Cost Parameter caluclation - equations
    model.addConstrs(
        (c_ipmb[i, p, m, b] == (cR_ipmb[i, p, 0, b] + cP_ipmb[i, p, m, b] +
cT_ipmb[i, p, 0, b] + cU_ipmb[i, p, m, b])
            for i in range(I) for p in range(P) for m in range(M) for b in range(B))
    model.addConstrs(
        (c_pmtrwb[p, m, t, r, w, b] == cT_pmtrwb[p, m, t, r, w, b] for p in
range(P) for m in [1, 2] for t in range(T)
            for r in range(R) for b in range(B) for w in range(W))
    model.addConstrs(c_fmb[f, m, b] == cT_fmb[f, m, b] for f in range(F) for m in
range(M) for b in range(B))
    model.addConstrs(c_msb[m, s, b] == cT_msb[m, s, b] for m in range(M) for s in
range(S) for b in range(B))
    model.addConstrs(c_mb[m, b] == cT_mb[m, b] for m in range(M) for b in
range(B))

    # Replenishment costs
    model.addConstrs(
        (cR_ipmb[i, p, 0, b] == (
            co_m[0] * ((demand_i[i] / n_ip[i, 0]) * ((dws_b[b] / (n_pm[0,

```

```

0] * muy_m[0] * v_m[0])) + l_pF[p]))
    for i in range(I)
    for p in [1, 2, 3, 4]
    for b in range(B))

# Preparation costs
model.addConstrs(
    cP_ipmb[i, 1, m, b] == (
        cl * ((demand_i[i] / n_ip[i, 0]) * (AL_pb[1] / (ntr_2 * ov)) +
(ht_ip + s_ip) * demand_i[i]))
    for i in range(I)
    for m in range(M)
    for b in range(B))
model.addConstrs(
    cP_ipmb[i, 2, m, b] == (
        cl * ((demand_i[i] / n_ip[i, 1]) * ((AL_pb[2] + mv) / ov) +
(ht_ip + s_ip) * demand_i[i]))
    for i in range(I)
    for m in range(M)
    for b in range(B))
model.addConstrs(
    cP_ipmb[i, 3, m, b] == (cl * ((ALk_pb[3] * d) / (ov * nk_4)) + (ht_ip +
s_ip) * demand_i[i])) for i in range(I)
    for m in range(M) for b in range(B))
model.addConstrs(
    cP_ipmb[i, 4, m, b] == (cl * ((ALk_pb[4] * d) / (ov * nt_5)) + (ht_ip +
s_ip) * demand_i[i])) for i in range(I)
    for m in range(M) for b in range(B))

# Transportation costs
model.addConstrs(
    (cT_ipmb[i, 0, m, b] == co_m[m] * (
        demand_i[i] / n_ip[i, 0]) * ((dW_s[s] / (n_pm[0, m] *
muy_m[m] * v_m[m])) + l_pm[0, m]))
    for s in range(S)
    for i, row in enumerate(map(tuple, assignment_parts_stations.tolist()))
if row[1] == s + 1
    for m in range(M)
    for b in range(B))
model.addConstrs(
    (cT_ipmb[i, 1, m, b] == co_m[m] * (
        demand_i[i] / n_ip[i, 1]) * ((dS_sb[b, s] / (n_pm[1, m] *
muy_m[m] * v_m[m])) + l_pm[1, m]))
    for s in range(S)
    for i, row in enumerate(map(tuple, assignment_parts_stations.tolist()))
if row[1] == s + 1
    for m in range(M)
    for b in range(B))
model.addConstrs(
    (cT_fmb[f, m, b] == co_m[m] * (
        demand_f[f] / n_f[f]) * ((dS_sb[b, s] / (n_pm[2, m] *
muy_m[m] * v_m[m])) + l_pm[2, m]))
    for s in range(S)
    for f in station_families[s + 1]
    for m in range(M)
    for b in range(B))
model.addConstrs(
    (cT_msb[m, s, b] == co_m[m] * ((d / nk_4) * ((dS_sb[b, s] / (n_pm[3, m] *
muy_m[m] * v_m[m])) + l_pm[3, m]))
    for s in range(S)
    for m in range(M)
    for b in range(B))
model.addConstrs(
    (cT_mb[m, b] == co_m[m] * ((d / nt_5) * ((dS_sb[b, 1] / (n_pm[4, m] *
muy_m[m] * v_m[m])) + l_pm[4, m]))
    for m in range(M)
    for b in range(B))
model.addConstrs(
    (cT_pmtrwb[0, m, t, r, 0, 0] == (co_m[m] * ((H * mr1_r[r]) / ((t + 1) *
muy_m[m] * v_m[m]))
    for m in [1, 2]
    for t in range(T)
    for r in range(R))
model.addConstrs(
    (cT_pmtrwb[p, m, t, r, 1, b] == co_m[m] * ((H * mr2_rb[r, b]) / ((t + 1) *
muy_m[m] * v_m[m]))
    for p in [1, 2, 3, 4]
    for m in [1, 2]
    for t in range(T)
    for r in range(R))

```

```

        for b in range(B)))

    # Usage costs
    model.addConstrs(
        cU_ipmb[i, p, m, b] == (ca * demand_i[i] * ((wd_p / ov) + s_ip)) for i in
range(I) for p in [0, 1] for m in
range(M) for b in range(B))
    model.addConstrs(
        cU_ipmb[i, p, m, b] == (ca * demand_i[i] * (wd_p / ov)) for i in range(I)
for p in [2, 3] for m in range(M) for
        b in range(B))

    # Define objective function
    # Part 1: cost of feeding part i with feeding policy p through vehicle type m,
denoted by cipmb.
    # This cost covers all processes, namely replenishment, preparation, transpor-
tation, and usage cost.
    # However, this transportation cost only includes transportation cost for line
stocked and boxed supply parts delivered by forklift
    transportation_costs_forklift = gp.quicksum(
        c_ipmb[i, p, m, b] * x_ipmb[i, p, m, b] for i in range(I) for p in
range(P) for m in range(M) for b in range(B))
    # Part 2: cost of transporting all parts assigned to an AGV or a tow train be-
tween the warehouse or the supermarket and the line, denoted by cpmtrwb
    transportation_costs_TT_and_AGV = gp.quicksum(
        c_pmtrwb[p, m, t, r, w, b] * v_pmtrwb[p, m, t, r, w, b] for p in range(P)
for m in [1, 2] for t in range(T) for
        r in range(R) for w in range(W) for b in range(B))
    # Part 3: cost of transporting sequenced parts by forklift cfm
    transportation_costs_sequenced_parts = gp.quicksum(
        c_fm[b, f, m, b] * x_ipmb[i, 2, m, b] for f in range(F) for i in fami-
lies_parts.get(f, []) if
        i > 0 and (i == 1 or assignment_parts_stations[i - 1, 3] != assign-
ment_parts_stations[i, 3]) for m in range(M)
        for b in range(B))
    # Part 4: transportation cost of kitted parts by a forklift for stationary
kits cmsb
    transportation_costs_stationary_kits = gp.quicksum(
        c_msb[m, s, b] * k_msb[m, s, b] for m in range(M) for s in range(S) for b
in range(B))
    # Part 5: transportation cost of kitted parts by a forklift for travelling
kits cmb
    transportation_costs_travelling_kits = gp.quicksum(c_mb[m, b] * t_mb[m, b] for
m in range(M) for b in range(B))
    # Part 6: supermarket costs: activation and capacity related
    fix_cost_supermarket = gp.quicksum((SC * Cap_b[b]) + Fix_S) * y_b[b] for b in
range(B))

    model.setObjective(
        transportation_costs_forklift + transportation_costs_TT_and_AGV + trans-
portation_costs_sequenced_parts + transportation_costs_stationary_kits + transpor-
tation costs travelling kits + fix cost supermarket,
        sense=gp.GRB.MINIMIZE)
    obj_value = model.getObjective().getValue()

    # Optimize the model
    model.optimize()
    # model.write('mp.lp')
    model.write('MILP.sol')
    Output(model)
    # Record end time
    end_time = time.time()
    # Calculate and print the duration
    duration = end_time - start_time
    print("Optimization duration: {:.2f} seconds".format(duration))

    return obj_value

# Read Input and data preparation
excel_file_path = 'Input_Data.xlsx'
Input = openpyxl.load_workbook(excel_file_path)
sheet = Input['Tabelle1']
muy_m = np.array([cell.value for cell in sheet['B4:D4']][0])
v_m = np.array([cell.value for cell in sheet['B5:D5']][0])
H = sheet['B6'].value
co_m = np.array([cell.value for cell in sheet['B7:D7']][0])
Fix_S = sheet['B32'].value
Cap_b = [cell.value for cell in sheet['B8:C8']][0]]

```

```

SC = sheet['B9'].value
nk_4 = sheet['B12'].value
nt_5 = sheet['B14'].value
h_2 = sheet['B16'].value
w_2 = sheet['B17'].value
q_p = np.array([cell.value for cell in sheet['B18:F18'][0]])
L_s = sheet['B19'].value
L_p = np.array([cell.value for cell in sheet['B20:F20'][0]])
cl = sheet['B22'].value
AL_pb = np.array([cell.value for cell in sheet['B23:F23'][0]])
ntr_2 = sheet['B24'].value
ov = sheet['B25'].value
mv = sheet['B28'].value
Alk_pb = np.array([cell.value for cell in sheet['B29:F29'][0]])
wd_p = sheet['B30'].value
ca = sheet['B31'].value
BIG_M_route = sheet['B33'].value
BIG_M_storage = sheet['B34'].value
s_ip = sheet['B37'].value
ht_ip = sheet['B38'].value
l_pF = np.array([cell.value for cell in sheet['B21:F21'][0]])
d_p = np.array([cell.value for cell in sheet['B35:C35'][0]])
dWS_b = np.array([cell.value for cell in sheet['B36:C36'][0]])
excel_file_path_distance_matrix = 'distance_matrix_xy_storage_to_stations.xlsx'
distance_matrix = pd.read_excel(excel_file_path_distance_matrix).to_numpy()
distance_matrix_1 = distance_matrix[1:3, :]
station_length = sheet['B10'].value
stations = distance_matrix_1.shape[1]
r_a2 = np.array([cell.value for cell in sheet['B11:C11'][0]]).astype(float)
r_a4 = np.array([cell.value for cell in sheet['B13:C13'][0]]).astype(float)
r_a5 = np.array([cell.value for cell in sheet['B15:C15'][0]]).astype(float)

milk_run_lengths = []

# Route creation and definition
for i in range(stations):
    for j in range(i + 1, stations + 1):
        route = list(range(i + 1, j + 1))
        length = [0] * len(Cap_b)
        for b in range(len(Cap_b)):
            length[b] = distance_matrix_1[b][route[0] - 1]
            for k in range(len(route) - 1):
                length[b] += station_length
            length[b] += distance_matrix_1[b][route[-1] - 1]
        milk_run_lengths.append((route, length))

routes_array = np.array(milk_run_lengths, dtype=object)
split_lengths = [(route, length[0], length[1]) for route, length in routes_array]
df = pd.DataFrame(split_lengths, columns=['Route', 'Length_1', 'Length_2'])
dWS = distance_matrix[0, :] * 2
dS_sb = distance_matrix[1:3, :] * 2
excel_file_path_assignment_parts_stations = 'part_station_assignment_xy.xlsx'
assignment_parts_stations = pd.read_excel(excel_file_path_assignment_parts_stations)
assignment_parts_stations = assignment_parts_stations.to_numpy()
columns_5_6 = assignment_parts_stations[:, 5:7]
r_ia = np.array(columns_5_6)
m_i = np.array(assignment_parts_stations[:, 7])
V_i = np.array(assignment_parts_stations[:, 8])
n_ip = np.array(assignment_parts_stations[:, 10:12])
demand_i = np.array(assignment_parts_stations[:, 9])
mr_rb = np.array(df[[f'Length_{i + 1}' for i in range(len(Cap_b))]])

excel_file_path_n_pm = 'n_pm.xlsx'
Input_n_pm = openpyxl.load_workbook(excel_file_path_n_pm)
Input_n_pm = Input_n_pm['Tabelle1']
num_rows = 5
num_cols = 3
n_pm = [[Input_n_pm.cell(row=i, column=j).value for j in range(1, num_cols + 1)] for i in range(1, num_rows + 1)]
n_pm = np.array(n_pm)

# l_pm
excel_file_path_l_pm = 'l_pm.xlsx'
Input_l_pm = openpyxl.load_workbook(excel_file_path_l_pm)
Input_l_pm = Input_l_pm['Tabelle1']
l_pm = np.array(
    [[Input_l_pm.cell(row=i, column=j).value for j in range(1, num_cols + 1)] for i in range(1, num_rows + 1)])

```

```

station_families = {}
for row in assignment_parts_stations:
    station = int(row[1])
    family = int(row[3])
    family -= 1
    if station not in station_families:
        station_families[station] = set()
    station_families[station].add(family)

families_parts = {}
for row in assignment_parts_stations:
    part = int(row[0])
    family = int(row[3])
    part -= 1
    if family not in families_parts:
        families_parts[family] = set()
    families_parts[family].add(part)

# demand f
excel_file_path = 'n_f.xlsx'
Input_nf = openpyxl.load_workbook(excel_file_path)
Input_nf = Input_nf['Tabelle1']
last_row = Input_nf.max_row
while Input_nf.cell(row=last_row, column=1).value is None and last_row > 1:
    last_row -= 1
values_a = [Input_nf.cell(row=i, column=1).value for i in range(2, last_row + 1)]
values_b = [Input_nf.cell(row=i, column=2).value for i in range(2, last_row + 1)]

# family demand
demand_f = np.array(values_a)
n_f = np.array(values_b)

milk_run_lengths_w1 = []
distance_matrix_1 = distance_matrix[0, :]
for i in range(stations):
    for j in range(i + 1, stations + 1):
        route_w1 = list(range(i + 1, j + 1))
        route_length = [0] * 1
        route_length = distance_matrix_1[route_w1[0] - 1]
        for k in range(len(route_w1) - 1):
            route_length += station_length
        route_length += distance_matrix_1[route_w1[-1] - 1]
        milk_run_lengths_w1.append((route_w1, route_length))
routes_array_w1 = np.array(milk_run_lengths_w1, dtype=object)
df_w1 = pd.DataFrame(routes_array_w1, columns=['Route', 'Length'])
mr1_r = df_w1['Length'].to_numpy(dtype=float)
mr2_rb = df[['Length 1', 'Length 2']]
mr2_rb = mr2_rb.values

OptModel_low = Model_1_TC(H, muy_m, v_m, co_m, Cap_b, SC, mr_rb, r_a2, r_ia, nk_4,
r_a4, m_i, V_i, r_a5, nt_5, l_pF,
                        n_ip, n_pm, demand_i, dWS_b, l_pm, dW_s, dS_sb, sta-
tion_families, n_f, demand_f, ca, ov, s_ip,
                        wd_p, cl, AL_pb, ntr_2, ht_ip, mv, ALk_pb, q_p, fami-
lies_parts, L_p, L_s, h_2, w_2,
                        BIG_M_route, mr1_r, mr2_rb, BIG_M_storage, d_p)

```