MID-AMERICA TRANSPORTATION CENTER

Final Report
WBS: 25-1121-3002-104
Report # MATC: 104

A New Optimization Approach to Distributed Manufacturing System Design

Haitao Li, Ph.D.
Professor and Chair
Supply Chain & Analytics Department
University of Missouri-Saint Louis

Paula Penagos, M.S.
Graduate Research Assistant
Supply Chain & Analytics Department
University of Missouri-Saint Louis

2025

A Cooperative Research Project sponsored by the U.S. Department of Transportation-Office of the Assistant Secretary for Research and Technology.

A New Optimization Approach to Distributed Manufacturing System Design

Haitao Li, Ph.D.
Professor and Chair
Supply Chain & Analytics Department
University of Missouri – St. Louis

Paula Penagos, M.S.
Graduate Research Assistant
Supply Chain & Analytics Department
University of Missouri – St. Louis

A Report on Research Sponsored by

Mid-America Transportation Center

University of Nebraska–Lincoln

March 2025

Technical Report Documentation Page

1. Report No. 25-1121-3002-104	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle A New Optimization Approach to Distributed Manufacturing System Design		5. Report Date March 2025
		6. Performing Organization Code
7. Author(s) Haitao Li ORCID No. 0000-0001-7609-2819 Paula Penagos		8. Performing Organization Report No. 25-1121-3002-104
9. Performing Organization Name and Address Supply Chain & Analytics Department College of Business Administration University of Missouri – St. Louis St. Louis, MO 63121		10. Work Unit No. (TRAIS)
		11. Contract or Grant No. 69A3552348307
12. Sponsoring Agency Name and Address Office of the Assistant Secretary for Research and Technology 1200 New Jersey Ave., SE Washington, D.C. 20590		13. Type of Report and Period Covered June 2023 to Dec 2024
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract
This project studies the supply-production-networks (SPNs) in a distributed manufacturing system (DMS) and develops a novel modeling framework and solution procedure for optimizing the production planning and resource allocation decisions in DMS, called centralized-autonomous coordination scheme (CACS). The CACS is able to coordinate the decisions of distributed facilities by optimizing the system-level metrics and respecting the autonomous decisions of individual facilities. Our approach is applied for a case study of a generic drug manufacturer implementing the new continuous manufacturing (CM) technology for active pharmaceutical ingredient (API) production. The results show significant advantages of CACS over the existing approach without centralized coordination.

17. Key Words Distributed manufacturing, Optimization, Coordination, Production planning, Resource allocation	18. Distribution Statement

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 123	22. Price

Table of Contents

Disclaimer vi
Abstract vii
Executive Summary viii
Chapter 1 Introduction and Background 1
1.1 Motivation and Problem Setting 1
1.2 Applications 3
1.3 Project Overview 4
1.4 Main Contributions 5
Chapter 2 Related Literature 6
Chapter 3 Modeling Framework and Coordination Algorithm 9
3.1 Model Formulations of Distributed Facilities 11
3.2 Model Formulation of Parent Company's Centralized Problem 13
3.3 Centralized-Autonomous Coordination Scheme (CACS) 15
Chapter 4 Chapter 4 Case Study 18
4.1 Case Description 18
4.2 Case Results: No Centralized Coordination 22
4.3 Case Results: With CACS 25
4.4 Case Results: With Limited Supply of Raw Material 31
4.5 Discussions and Takeaways 32
Chapter 5 Conclusions and Future Study 34
References 36
Appendix A Detailed Steps of the CACS Procedure for the Scenario with Limited KSM Supply 39
Appendix B Python Code for the DMS-D Approach 46
Appendix C Python Code for the DMS-CACS Approach 64

List of Figures

Figure 3.1 A conceptual depiction of PPDM 10
Figure 3.2 Model sets and parameters 13
Figure 3.3 Model formulation of maximization and profit with decision variables 14
Figure 3.4 A conceptual depiction of a centralized-autonomous-coordination scheme (CACS). 16
Figure 3.5 A high-level flow chart of the sequential negotiation procedure in CACS 17
Figure 4.1 An example of distributed API manufacturing system 20
Figure 4.2 Final pure decentralized solution following A, B, C. 23
Figure 4.3 Facility A's proposed plan 26
Figure 4.4 Parent company's revised plan after Facility A submits its proposed plan 27
Figure 4.5 Facility B's proposed plan 28
Figure 4.6 Parent company's revised plan after Facility B submits its proposed plan 29
Figure 4.7 Facility C's proposed plan 30
Figure 4.8 Parent company's revised plan after Facility C submits its proposed plan 31
Figure A.1 Facility A's proposed plan 39
Figure A.2 Parent's revised plan after Facility A submits its proposed plan 41
Figure A.3 Facility B's proposed plan 42
Figure A.4 Parent's revised plan after Facility A submits its proposed plan 43
Figure A.5 Facility C's proposed plan 44
Figure A.6 Parent's revised plan after Facility C submits its proposed plan 45

List of Tables

Table 4.1 Comparison of DMS-D solutions with different orders of decision..... 24
Table 4.2 The DMS-CACS and DMS-D solutions for the case with limited KSM supply 32

Disclaimer

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, by a grant from the U.S. Department of Transportation's University Transportation Centers Program. However, the U.S. Government assumes no liability for the contents or use thereof.

Abstract

This project studies the supply-production-networks (SPNs) in a distributed manufacturing system (DMS) and develops a novel modeling framework and solution procedure for optimizing production planning and resource allocation decisions in the DMS, called centralized-autonomous coordination scheme (CACS). The CACS optimizes the system-level metrics and respects the autonomous decisions of distributed facilities. Our approach is applied to a case study of a generic drug manufacturer implementing the new continuous manufacturing (CM) technology for API production. The results show a significant advantage in using the CACS over the existing approach without centralized coordination.

Executive Summary

Driven by fasting growing technologies such as sensors, Internet-of-Things (IoT) and advanced manufacturing such as 3D printing, Distributed Manufacturing (DM) is an emerging paradigm with some known advantages of low production cost and flexibility for small-batch customized products. It has various real-world applications in API manufacturing, vertical farming and modular construction. Effectively and efficiently managing a distributed manufacturing system (DMS) requires a solution that addresses the following unique features: (i) localized sourcing and production; (ii) autonomous decision-making; (iii) centralized coordination; and (iv) dynamic operations and decision-support. The goal of this project aims to develop a novel modeling framework and solution procedure for the design of a distributed manufacturing system (DMS) in terms of production and resource allocation decisions, while considering autonomous distributed planning decisions of supply-production networks (SPNs) in the DMS.

The technical contributions of our work include: (i) optimization models of distributed facilities with heterogenous and conflicting objectives; (ii) a centralized model to coordinate the decisions of distributed facilities by optimizing the system-level metrics and minimizing deviation of the facilities' original proposed decisions; (iii) a centralized-autonomous coordination scheme (CACS) and a sequential negotiation procedure to implement CACS for the production planning and resource allocation decisions in the SPN of a DMS.

Our approach has been applied for a case study of a generic drug manufacturer who implements the new continuous manufacturing (CM) technology, which addresses the well-known national threat of public health in the United States, who relies heavily on outsourcing active pharmaceutical ingredients (APIs) from other countries. This case study shows that our CACS solutions have clear advantages over the benchmark solutions without coordination,

especially for the scenario with limited supply of raw material. The CACS is able to overcome the challenge and deficiency caused by each individual facility making autonomous decisions without centralized coordination, which leads to either local optima or infeasible, unfair plans.

Our work provides a data-driven game-decision-theoretic solution for implementing, managing and scaling a DMS by optimizing the production planning and resource allocation decisions. This approach can be applied, adapted and extended for various applications in other sectors such as agriculture/food, construction, and healthcare. With the profitability gain, cost saving, and other improved performance metrics such as carbon emission, our solution has the potential to make significant impacts on economic and workforce development.

Chapter 1 Introduction and Background

1.1 Motivation and Problem Setting

Driven by fast growing technologies such as sensors, Internet-of-Things (IoT) and advanced manufacturing such as 3D printing, Distributed Manufacturing (DM) is an emerging paradigm that has been gaining traction worldwide (Srai et al., 2016). The purpose of DM is to allow for small-scale networks with more localized sourcing and production, such that parts, components, and semi-finished goods can be manufactured and assembled in a distributed and flexible supply chain (Roscoe & Blome, 2016; Srai et al., 2020). Rather than taking advantage of economies-of-scale with large plants and production batches for make-to-stock (MTS) manufacturing, DM motivates and facilitates customized, flexible production in small batches. Once properly designed and scaled, DM can be part of an industry cluster (Porter, 2000) to achieve both high production volume and variety, which is the main characteristic of mass customization (Anderson, 2008). Existing studies show that DM can deliver on-demand personalized consumer products, facilitate mass customization, support demands in remote regions, and underpin the shared and circular economy (Durach, Kurpjuweit, & Wagner, 2017; Ratnayake, 2019).

While mass customization has clear advantages over traditional manufacturing paradigms of MTS and make-to-order (MTO), it is also well-known that mass customization can be more difficult and costly to manage. For instance, properly managing production planning of distributed facilities can be difficult when satisfying a common demand pool with limited total supply of parts and raw materials. There are difficulties when coordinating the operations of distributed facilities, i.e., production, distribution and transportation, such that demands are fulfilled efficiently. We also recognize the different, sometimes conflicting, objectives of

autonomous facilities can be difficult to reconcile while achieving global optimal performance at the corporate level.

We identify and focus on the following distinctive characteristics of a distribution manufacturing system (DMS) in this study.

- **Localized sourcing and production**: A DMS consists of multiple small-scale supply-production networks with shortened logistics distances compared to a regular supply chain network. This means it may not have the opportunity of low-cost outsourcing. However, a small-scale network incurs less transportation cost, has more agility when responding to customers, and reduces the impact of supply disruption.
- **Autonomous decision-making**: Each supply-production network (SPN) in a DMS is an autonomous entity who makes sourcing and production decision to achieve its own objective, e.g., profit maximization, cost minimization, and minimization of environmental impact. Sometimes these objectives can be conflicting to each other. For example, reducing environment impact may require the use of green manufacturing or transportation methods, which can be costly from the profitability perspective. In addition, depending on the degree of autonomy, information can be *asymmetric* among the SPNs. That is, a distributed facility in one SPN may not know the market demand or production plan of the other distributed facility in a different SPN, or it might be costly in time or money to obtain such information.
- **Centralized coordination**: As part of a DMS, SPNs also need to cooperate when fulfilling customer orders. Often, there is a parent firm who owns or manages the SPNs and has the authority to recommend or overwrite the production decision of an SPN. The

parent firm also has access to information on the SPNs including their market demand, resource requirements, and production decisions.

- **Dynamic operations and decision-support**: Operations in a DMS are often dynamic in that they change and adapt over time. For example, factories can be mobile to provide critical operations in a production process to certain SPNs they are assigned to. An SPN may serve varying customer demands that change over time. Effectively and efficiently managing autonomous SPNs in the DMS to make dynamic decisions when there is uncertainty can be complex and challenging.

1.2 Applications

The above scenario of a DMS consisting of a parent firm and multiple autonomous SPNs is a common set up in multiple industries.

- API and Generic Drug Manufacturing: The emerging advancement in flow chemistry and continuous manufacturing feature agile and small-batch production of active pharmaceutical ingredients (API; Algorri, Abernathy, Cauchon, Lamm, & Moore, 2022), which makes it possible to meet the point-of-care demands (Adamo et al., 2016). While the technology of continuous manufacturing is maturing, managing such a system for API manufacturing requires proper planning and coordinating distributed production facilities in a DMS.
- Vertical Farming: While indoor farming, also known as vertical farming or controlled environment agriculture (CEA), has been gaining traction worldwide (Artemis, 2020; Autogrow, 2020), properly managing an indoor farming system is a non-trivial task. A recent study by H. Li et al. (2023) showed that indoor farming start-ups face significant economic challenges, despite maturing technologies in plant science and engineering for

CEA. The authors proposed a centralized optimization model for the design and operations of an indoor farming supply chain. To properly capture the autonomous nature of multiple indoor farming facilities in the production system requires modeling it as a DMS.

- Modular Construction: Modular Construction is an alternative project paradigm to the traditional project management approach, where a building is constructed off-site in a controlled plant environment (Almashaqbeh & El-Rayes, 2021). Components of a building are produced in "modules", which are then put together on site. While modular construction has several perceived advantages—including reduced project schedule, less material waste, greater flexibility and reuse, better quality control, and less environmental impact—its adoption and success often varies significantly by geographical locations, economic conditions, regulatory policies, and industry practices. This calls for new approaches to manage project operations at multiple off-site locations to simultaneously optimize the sourcing, transportation, resource allocation, and scheduling decisions.

1.3 Project Overview

The goal of this project is to develop a new optimization approach for the design of a distributed manufacturing system (DMS) in terms of production and resource allocation decisions, while considering autonomous distributed planning decisions of supply-production networks (SPNs) in the DMS. The main challenge and technical advancement of the project is the design and implementation of a decision-game-theoretic model to capture the autonomous decision-making feature of each local SPN, while achieving the optimal system-wide global performance. Advanced computational algorithms will also be developed to obtain quality solutions efficiently. This project aligns with DOT's strategic goals of economic strength and

global competitiveness, safety, and supports MATS-TSE's themes on resilient supply chains and transportation systems of the future.

1.4 Main Contributions

We design and develop a decision-game-theoretic model to optimize the following decisions in a DMS: (i) allocation of raw material supply to distributed facilities; (ii) production planning decisions of distributed facilities in the SNPs; and (iii) centralized coordination to achieve the system-wide global performance metrics.

The model and solution algorithms developed in this project are applied for a real world case study on advanced manufacturing for generic drugs to demonstrate how our approach provides data-driven decision-support for the design of DMS, and the benefit of DMS compared to the existing approach. A case study will be conducted to examine the performance of our model and algorithms in different scenarios of raw material availability.

Chapter 2 Related Literature

The topic of distributed manufacturing has been gaining traction. Notably, Yu et al. (2020) provided an updated review of distributed/shared manufacturing in the context of shared economy, and identified three related components: shared manufacturing service modeling, shared manufacturing architecture formulation, and shared manufacturing service scheduling. Shahmoradi-Moghadam & Schonberger (2021) studied a DMS with mobile factories shared by multiple production sites, and proposed a mixed-integer programming (MIP) model for the addressed problem. Liu, Liu, & Wei (2021) addressed the allocation of shared manufacturing resources and developed a bilevel programming approach to optimize flexibility at the upper level and quality of service at the lower level. Wang, Zhang, Guo, & Zhang (2021) developed a digital twin model to allocate and coordinate a shared manufacturing resource for on-demand customization.

The other stream of research focuses on the design and planning of industry clusters. Notably, Chan, Swarnkar, & Tiwari (2007) present a conceptual framework of distributed artificial intelligence (AI) for information-based manufacturing. Each manufacturer is modeled as an agent making autonomous decisions coordinated by a centralized mechanism. Xue, Wei, & Liu (2012) presented a qualitative study to address the gaps in implementing cluster supply chain and discussed the specific conditions, advantages, and challenges in cluster supply chains. A case study was conducted with an emphasis on implementing service systems. J. Li et al. (2012) considered the design of multiple parallel supply chains in an industry cluster to allow for inter-chain cooperation. They devised a mixed-integer nonlinear programming (MINLP) model with a coefficient to indicate the proportion of vertical and horizontal cooperation. A hybrid integer programming and genetic algorithm (GA) method was applied to solve the model. Xiang, Song, & Ye (2014) considered a multi-sourcing supply-demand network with suppliers in an industry

cluster purchasing from manufacturers outside the cluster. Heuristic demand allocation decision rules, namely, production capacity-based and production load-balancing, were implemented. Discrete event simulation was performed to evaluate the heuristic rules under uncertain demand and production capacity. Their approach did not model the optimization problem of each entity. Renna & Perrone (2015) expanded on the work of Xiang et al. (2014) to study a multi-sourcing supply-demand network in a dynamic setting, where the order allocation decision effects the long-term supply chain partnership. A simulation was employed to evaluate the capacity-based and production load-based allocation strategies under different market conditions.

Yan & Liu (2018) studied the cluster supply chain where entities in an industry cluster formed a supply chain such that there exist both cooperation and competition among the entities. They proposed a decision rule for transshipment based on system dynamics, which is shown to improve the customer satisfaction level and reduce inventory compared to the two benchmark approaches. Haque, Paul, Sarker, & Essam (2020) studied a multi-tier supply chain with decentralized entities of manufacturers, distributors, and retailers who make autonomous decisions. To address the asymmetric information between decentralized entities, a bilevel modeling framework was developed with the upper-level problem aiming to coordinate the decentralized decisions at the lower-level via Nash equilibrium.

None of the studies addressed the situation of asymmetric information among SPNs, nor the different objectives of distributed facilities. A related topic on designing a complex products or systems while considering different design needs with heterogeneous objectives/metrics and requirements has been extensively studied through the so-called Analytical Target Cascading (ATC) Optimal System Design approach. The seminal work of Kim (2001) proposes the methodology of target cascading for hierarchical multilevel system design and multidisciplinary

design optimization (MDO), where multiple teams participate in the design of a system with multiple modules or subsystems with different performance metrics (targets) to meet and different constraints (requirements) to satisfy. Kim, Michelena, Papalambros, & Jiang (2003) present a case study of automobile chassis design. An application of target cascading for vehicle design is presented by Kim et al. (2002). Other works in this line of research include an augmented Lagrangian relaxation method to improve the nested iterative procedure of ATC (Tosserams, Etman, Papalambros, & Rooda, 2006), a coordination approach for ATC with linking variables as well as coupling objectives and constraints (Tosserams, Etman, & Rooda, 2008), and an empirical study of the convergence behavior of augmented Lagrangian coordination for solving multi-modal optimization problems in a distributed fashion (Tosserams, Etman, & Rooda, 2010). However, none of these works address the unique characteristics and decision needs in a DMS.

Our review of the related literature shows that although there is existing research on industry cluster design and multidisciplinary design optimization, there has been little work addressing the design of distributed manufacturing system.

Chapter 3 Modeling Framework and Coordination Algorithm

Consider a distributed manufacturing system (DMS) consisting of multiple flexible and small-scale manufacturing facilities owned by one parent company. Each facility plans for its own production and distribution to meet the demands of multiple customers though its own supply-production network (SPN). The parent company would like to optimally allocate and centralize the limited raw materials and supplies, while allowing each manufacturing facility to make its planning decision autonomously. We name the addressed problem Production Planning for Distributed Manufacturing (PPDM).

The PPDM is conceptually described in Figure 3.1. The parent company H owns three distributed manufacturing facilities—A, B, and C—to serve seven customers, which form a DMS. Each facility operates its own SPN. In Facility A's SPN, A receives a raw material supply from H, plans for its production to serve the demand of Customers 1, 2, and 3, with an objective of maximizing profit; Facility B also receives supply from H and serves Customers 2, 3, 4, and 5 to maximize its market share of Customers 2 and 3 among its entire customer pool of 2, 3, 4, and 5; Facility C's supply of raw material also comes from H, and it serves Customers 4, 5, 6, and 7 with an objective of minimizing carbon dioxide emission. There is a limited supply of raw materials from H which is shared among the three facilities. The parent company H would like to properly allocate the limited raw material supply and to coordinate distributed facilities' production for demand fulfillment, while optimizing the system-wide performance metric(s).

To capture an authentic decision environment and process, we make the following two assumptions.

Assumption 1: Production decisions of distributed facilities are made in a certain order, which may be predetermined by the decision-maker.

Assumption 2: Distributed facilities have asymmetric information about market demand, available raw material supply and other facilities' objectives.

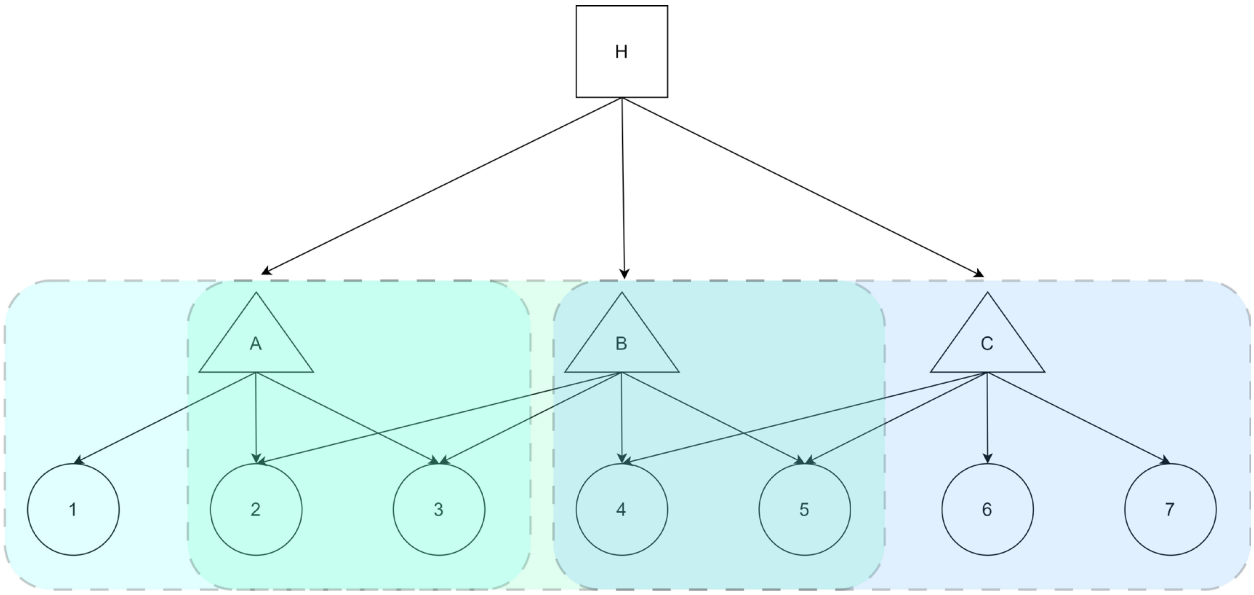

Figure 3.1 A conceptual depiction of PPDM

The most distinctive and unique feature of PPDM is that planning decisions are made in a distributed way by autonomous manufacturing facilities, which is coordinated by a centralized decision-maker. The relationship among the distributed manufacturing facilities is called "coopetitive". As shown in Figure 3.4, on one hand, the distributed facilities compete for a limited supply of raw materials; on the other hand, they cooperate to fulfill demand. The cooperation is also made possible by the centralized intervention and coordination of the parent company.

Our main technical development in this project is the design of a new modeling framework and algorithm to coordinate autonomous SPNs to achieve system-wide optimal performance.

3.1 Model Formulations of Distributed Facilities

In this section, we present the model formulation for the distributed facilities. It includes the following sets and parameters.

Sets

H: one parent company

F: set of facilities

C: set of customers

P: set of products

Parameters

K_h: raw material availability at parent $h \in H$

κ_{pf} : manufacturing capacity of product $p \in P$ at facility $f \in F$

d_{pc}: demand of product $p \in P$ by customer $c \in C$

c_{pf} : cost of manufacturing product $p \in P$ at facility $f \in F$ α_p: $KgCO_2$ emission per unit of product $p \in P$

π_p: unit of revenue of product $p \in P$

r_p: units of raw material needed to manufacture product $p \in P$

The following decision variables are defined.

$X_{pfc} \geqslant 0$: units of product $p \in P$ served to customer $c \in C$ by facility $f \in F$

$Y_f \geqslant 0$: units of raw material allocated to facility $f \in F$

Facility A solves the profit maximization problem with the equations below. $\bar{X}_{pfc}$ denotes the production decision of other facilities, i.e., $f \in \{B, C\}$ The first constraint satisfies the demand of product p at customer c, given the production from other facilities. The second

constraint ensures that the total required raw material does not exceed Y_A, determined by the parent company. The third constraint satisfies the available capacity at facility A.

$$
\begin{aligned}
\text{maximize} \quad & \sum_{p\in P}\sum_{c\in C}(\pi_p - c_{pA}) \cdot X_{pAc} \\
\text{subject to} \quad & \sum_{f\in\{B,C\}} \overline{X_{pfc}} + X_{pAc} \geq d_{pc} \qquad \forall \quad p \in P, c \in C, \\
& \sum_{p\in P}\sum_{c\in C} r_p \cdot X_{pAc} \leq Y_A, \\
& \sum_{c\in C} X_{pAc} \leq \kappa_{pA} \qquad \forall \quad p \in P
\end{aligned}
$$

Facility B maximizes its market share for all products and customers. The first constraint satisfies the demand of product p at customer c, given the production from other facilities. The second constraint ensures that the total required raw material does not exceed Y_B, determined by the parent company. The third constraint satisfies the available capacity at facility B.

$$
\begin{aligned}
\text{maximize} \quad & \frac{\sum_{p\in P}\sum_{c\in C} X_{pBc}}{\sum_{p\in P}\sum_{c\in C} d_{pc}} \\
\text{subject to} \quad & \sum_{f\in\{A,C\}} \overline{X_{pfc}} + X_{pBc} \geq d_{pc} \qquad \forall \quad p \in P, c \in C, \\
& \sum_{p\in P}\sum_{c\in C} r_p \cdot X_{pBc} \leq Y_B, \\
& \sum_{c\in C} X_{pBc} \leq \kappa_{pB} \qquad \forall \quad p \in P
\end{aligned}
$$

Facility C minimizes the total cost of its carbon emissions. The first constraint satisfies the demand of product p at customer c, given the production from other facilities. The second

constraint ensures that the total required raw material does not exceed Y_C, determined by the parent company. The third constraint satisfies the available capacity at facility C.

3.2 Model Formulation of Parent Company's Centralized Problem

We develop a model for the parent company's centralized problem to optimize the system-wide performance metrics, taking distributed facilities' decisions as inputs. The parent would also minimize the deviation of its recommendation from a facility's decision. The following sets and parameters are defined for the model.

- **Sets:**

  H: one parent company
  F: set of facilities
  C: set of customers
  P: set of products

- **Parameters:**

  K: raw material availability at parent H
  κ_{pf}: manufacturing capacity of product $p \in P$ at facility $f \in F$
  d_{pc}: demand of product $p \in P$ by customer $c \in C$
  c_{pf}: cost of manufacturing product $p \in P$ at facility $f \in F$
  α_p: cost of producing a $KgCO_2$ emissions per unit of product $p \in P$
  π_p: unit of revenue of product $p \in P$
  r_p: units of raw material needed to manufacture product $p \in P$
  c_{raw}: cost of unit of raw material
  CS_{fc}: 1 if customer $c \in C$ is served by facility $f \in F$; 0, o.w.

Figure 3.2 Model sets and parameters

The parent company's centralized model formulation can be written as:

- **Decision variables:**

  $X_{pfc} \geqslant 0$: units of product $p \in P$ served to customer $c \in C$ by facility $f \in F$
  $Y_f \geqslant 0$: units of raw material allocated to facility $f \in F$
  $DEV_f \geqslant 0$: Absolute value of deviation in units of product from facility $f \in F$ requirement to parent allocation.
  $D_{pc} \geqslant 0$: Absolute value of deviation in units of product $p \in P$ for customer $c \in C$

- **Parent H - Overall profit maximization problem**

  The parent company's goal is to optimize the system-wide performance metrics while considering the decisions of distributed facilities *a priori*.

  Profit = (Revenue - Raw material cost - Manufacturing cost - CO_2 cost) - Penalty for resource allocation deviation - Penalty for demand fulfillment deviation

$$
\begin{aligned}
\text{maximize} \quad & \sum_{p\in P}\sum_{f\in F}\sum_{c\in C}[\pi_p \cdot X_{pfc} - C_{raw} \cdot Y_f - c_{pf} \cdot X_{pfc} - \alpha_p \cdot X_{pfc}] - \sum_{f\in F} DEV_f - \sum_{c\in C}\sum_{p\in P} D_{pc} \\
\text{subject to} \quad & \sum_{p\in P}\sum_{c\in C} r_p \cdot X_{pfc} \leq Y_f \qquad \forall f \in F, \\
& \sum_{p\in P} X_{pfc} \leq CS_{fc} \cdot K_f \qquad \forall f \in F, c \in C, \\
& \sum_{f\in F} Y_f \leq K, \\
& DEV_f \geq Y_h - \sum_{p\in P}\sum_{c\in C}[r_p \cdot \overline{X_{pfc}}] \qquad \forall f \in F, \\
& DEV_f \geq -(Y_h - \sum_{p\in P}\sum_{c\in C}[r_p \cdot \overline{X_{pfc}}]) \qquad \forall f \in F, \\
& D_{pc} \geq d_{pc} - \sum_{f\in F} \overline{X_{pfc}}, \quad \forall p \in P, c \in C, \\
& D_{pc} \geq -(d_{pc} - \sum_{f\in F} \overline{X_{pfc}}), \quad \forall p \in P, c \in C
\end{aligned}
$$

Figure 3.3 Model formulation of maximization and profit with decision variables

The parent company determines how much products from each distributed facility go to serve customers (X_{pfc}) and the corresponding amount of raw materials allocated to each facility (Y_f), which will potentially deviate from the decision made by a facility (priori). A decision variable DEV_f is defined to capture the absolute deviation of the raw materials allocated by the

parent company from the amount requested by a facility. Another auxiliary decision variable D_{pc} is defined to quantify the absolute deviation of products sent to customers from the actual demand, since it is undesirable to have either an unfulfilled demand gap or an over-fulfillment surplus.

The parent company's objective function maximizes the system-wide total profitability of all distributed facilities for all products. The first triple-summation term computes the total profit as the total revenue subtracting costs of raw materials, production, and carbon dioxide emissions. The next term penalizes the absolute deviation from a facility's request of raw materials. The last term penalizes the absolute deviation from the actual demand.

The first constraint ensures that the production of all products at each distributed facility does not require more than the amount of raw materials allocated to the facility. The second constraint guarantees that the production of all products at each facility does not exceed the maximum production capacity of the facility. The third constraint limits the total allocated raw materials not to exceed the maximum availability from the parent company. The fourth and fifth constraints together compute the absolute value of deviation of raw material allocated to each facility from the facility's requested raw material amount. Note that some decision variables X_{pfc} are fixed as constants, denoted as $\overline{X_{pfc}}$, for those facilities that have already made their production decisions. The last two constraints calculate the absolute value of the deviation of product quantity sent to customers from the actual demand.

3.3 Centralized-Autonomous Coordination Scheme (CACS)

The parent company aims to coordinate the production of distributed facilities. The objective function of the parent company optimizes the global performance of the DMS, which includes the objective of all the distributed facilities, plus an additional term to minimize the

deviation of the parent firm's coordination decision from a facility's proposed decision. In this way, the parent firm is able to optimize the global performance metric of the entire DMS, while respecting each distributed facility's decision.

In this section, we present a centralized-autonomous coordination scheme (CACS) for the DMS as shown in Figure 3.4.

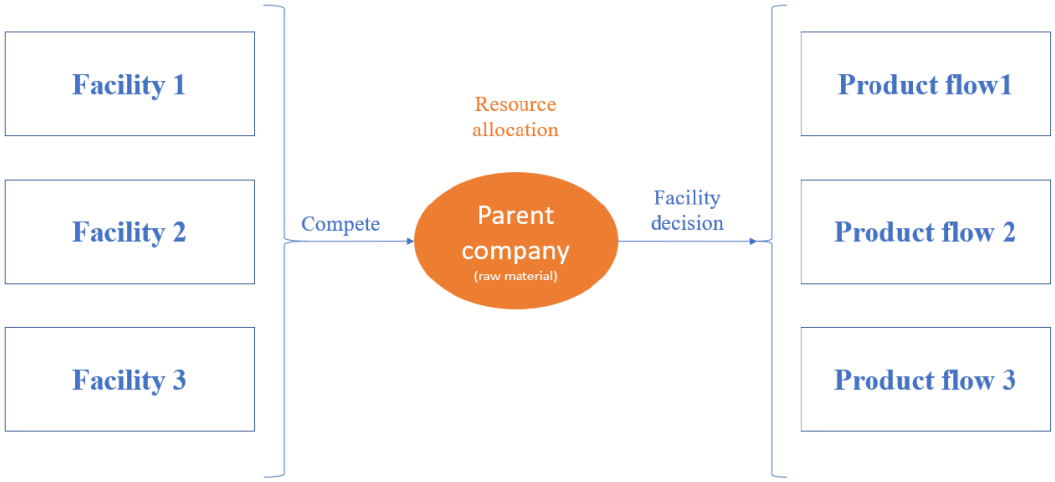

Figure 3.4 A conceptual depiction of a centralized-autonomous-coordination scheme (CACS)

The main features of CACS include: (1) Asymmetric Information: We consider a realistic situation where the information about each facility's objective function and the available resource is "asymmetric" among the distributed facilities, which is a key characteristic of autonomous facilities in the DMS; (2) Iterative Process: Following a certain order of distributed facilities, one facility makes a production planning decision and submits its proposal to the parent firm. Then the parent firm makes its global decision and adjusts the available resource for the next facility's decision in the iterative process. A flow chart of the iterative sequential negotiation procedure in the CACS is presented in Figure 3.5; (3) Global Metrics vs

Autonomous Decisions: The parent firm optimizes the global performance metrics while observing and respecting each distributed facility's autonomous decision. This is achieved by minimizing the deviation of parent firm's recommendation from a facility's decision.

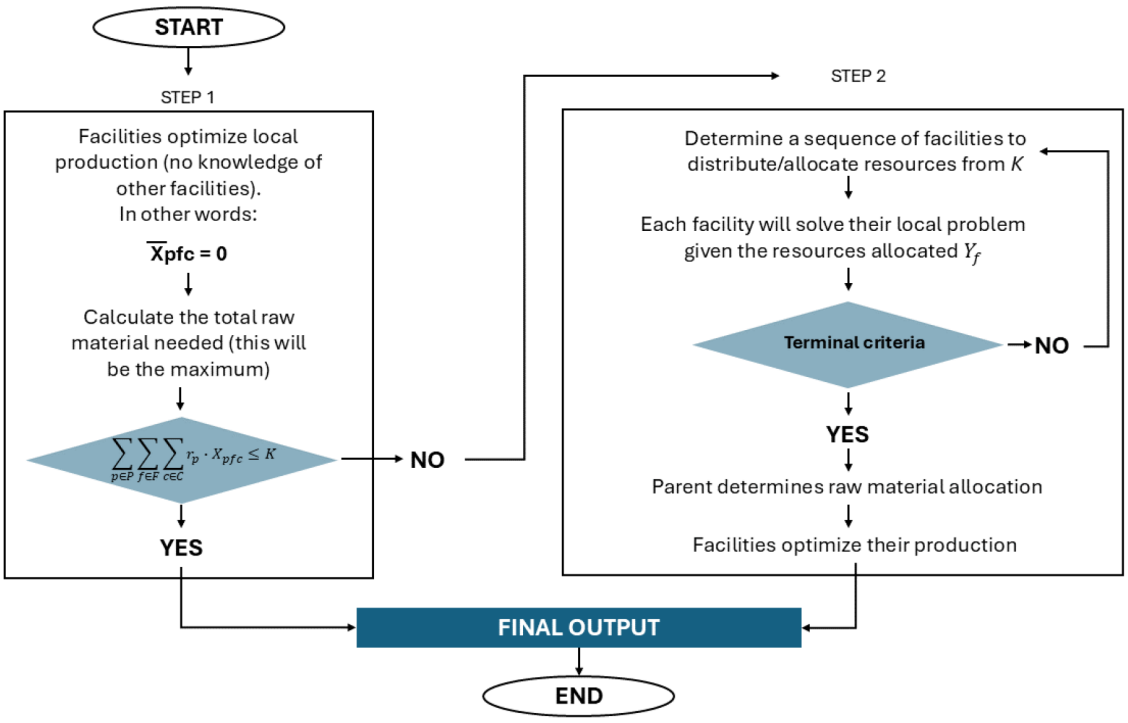

Figure 3.5 A high-level flow chart of the sequential negotiation procedure in CACS

Chapter 4 Chapter 4 Case Study

4.1 Case Description

The US is heavily reliant on other countries, particularly India and China, for the production of critical active pharmaceutical ingredients (APIs) and related starting materials. The nation's over-reliance on offshore pharmaceutical manufacturing poses significant threats to public health and national security as millions of Americans rely on undisrupted access to safe pharmaceuticals.

Less than 5% of large-scale global API sites are in the US; the majority of large-scale manufacturing sites are in China and India. The latest *Drug Shortages: Root Causes and Potential Solutions: A Report by the Drug Shortages Task Force,* issued in 2020 by the FDA, noted that 88% of the manufacturing sites for the US market's APIs were located outside the US in 2018.

The COVID-19 pandemic, the shifting geopolitical landscape, and extreme weather events have all exposed how reliance on outside-the-US (OUS) manufacturing of our essential and critical medicines has put the US pharmaceutical supply chain at risk. The majority of our nation's APIs are sourced from manufacturers located outside of the United States, which has created both public health and a national security risks. A September 2022 evaluation of the US pharmaceutical supply report by the Department of Defense Office of Inspector General concluded "*reliance on foreign suppliers for pharmaceuticals is a public health, readiness and national security risk*."

The application of new, continuous manufacturing (CM) into existing facilities offers a promising path to build resiliency (Domokos, Nagy, Szilagyi, Marosi, & Nagy, 2021; Aulakh, Settanni, & Srai, 2021). By condensing processes that used to take months of expensive work into a few days, the Cost of Goods Sold (COGS) can be reduced by as much as 30-50%. It also

promotes the use of automation to shrink labor costs while improving quality controls to minimize waste. Industry 4.0 proponents believe that automation of traditional manufacturing processes can make these gains even greater when the right technologies extend across the supply chain.

In addition to being less expensive, new technology such as continuous manufacturing can expedite regulatory checks without compromising oversight, improve manufacturing agility, and cause less of an environmental impact. The FDA also recognizes that the adoption of advanced manufacturing techniques would reduce costs and increase the resilience of US production, enabling a competitive advantage to ensure a stable supply of critical drugs. The placement of new, advanced technologies in idled sites would boost production, build emergency capacity, and help minimize the risk for existing manufacturers to upgrade equipment and expand production lines. Advanced manufacturing also creates new opportunities to develop a trained workforce through hands-on experience, with outreach concentrated in communities of color, to develop critical, transferable skills within the pharmaceutical supply chain.

Consider generic drug manufacturer $\mathcal{H}$ who is part of the initiative to bring active pharmaceutical ingredients (API) manufacturing to the U.S. $\mathcal{H}$ has equipment for the new production technology known as continuous manufacturing (Domokos et al., 2021; Aulakh et al., 2021). Continuous manufacturing (CM) features a new and flexible production paradigm, which can be operated in a distributed system with small batches, rather than the traditional API manufacturing process with discrete operations in large batches at one facility to achieve economies-of-scale. In this numerical example, $\mathcal{H}$ operates three distributed facilities A, B and C to serve seven customers. Facility A has an objective of maximizing profit. Facility B's objective is to maximize market share of Customers 2 and 3 among its customer pool of 2, 3, 4, 5. Facility

C's objective is to minimize carbon dioxide emission. There is a limited supply of raw materials that is shared among the three facilities. The distributed manufacturing system for APIs can be depicted by Figure 4.1.

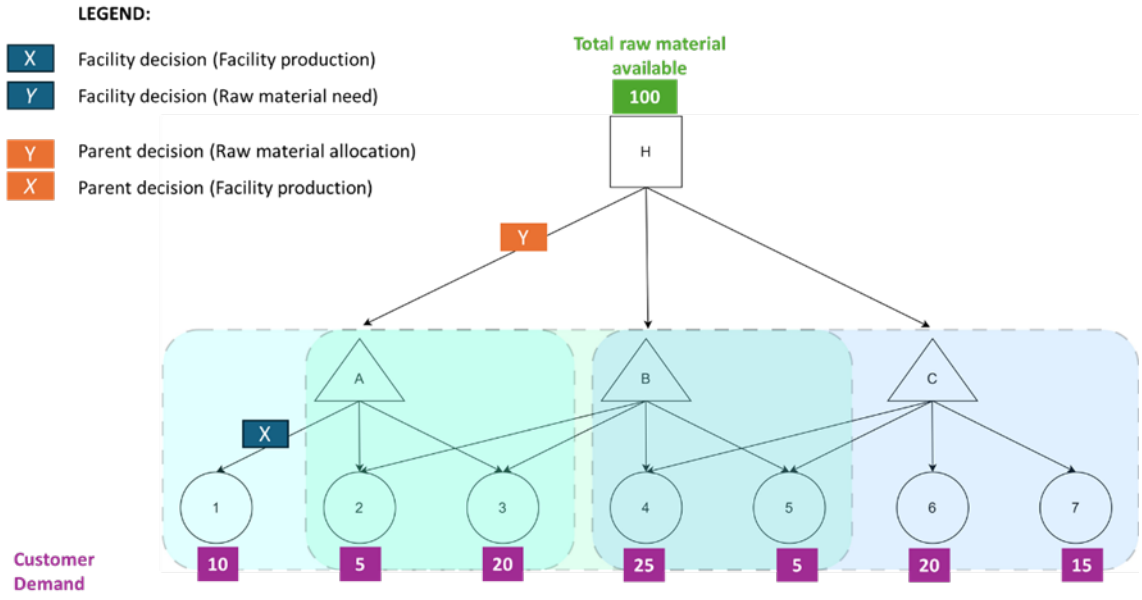

Figure 4.1 An example of distributed API manufacturing system

Without loss of generality, we assume the following input data in the case study:

- The total available supply of key starting materials (KSM) from $\mathcal{H}$ is 100 kgs.
- Assume that producing 1 kg of API requires 0.8 kgs of KSM.
- The production capacities (in kgs) at distributed facilities A, B, and C are 500, 100 and 100, respectively.
- The production costs ($/kg) at distributed facilities A, B, and C are 1, 0.5, and 0.8, respectively.

- The demands (in kg) at markets 1 – 9 are 10, 5, 20, 25, 5, 20, 15, respectively.
- The market price ($/kg) of the API is 5.
- The cost of carbon emission is $0.5 per kg of API production.

Although the distributed API manufacturing system enabled by the advanced CM technology has multiple perceived advantages over the traditional paradigm, the Senior VP of Operations and Supply Chain realizes there are several challenges in managing the new production and supply chain system:

- While distributed CM facilities has the capability and flexibility of meeting local demands, the API manufacturer $\mathcal{H}$ as the parent company would like to have certain centralized control to ensure company-wide global performance metrics are achieved.
- Distributed facilities have asymmetric information, meaning that they do not know each other's information about the corresponding local demands, cost of production, or objective metric.
- Newly established CM facilities operate along with other existing production facilities (rather than replacing them), which may have different and conflicting individual objective metrics to optimize.

To address these challenges and decision needs, the Senior VP calls for a team consisting of professionals in optimization modeling and subject matter experts for the company's purchasing, production and operations, to work on a solution. The team is charged to:

- Develop a data-driven decision-support solution to plan for the production and allocation of raw materials of KSM.

- The solution must allow each distributed facility to make their autonomous decisions under asymmetric information, while achieving the company-wide global performance metrics.
- Compare the new solution with the status quo approach and assess its performance. The company's existing approach asks each distributed facility to come up with their own production plan without any intervention or coordination.

4.2 Case Results: No Centralized Coordination

Without intervention or coordination, the company currently implements a simple approach where all the distributed facilities make their own decisions for a given order, which we call DMS-D for DMS-Decentralized. In this approach, the parent company only serves as a supplier of raw materials and does not intervene in the decision process, e.g., how resources are allocated to the distributed facilities.

Assuming the order of decisions is A,B, then C. Using the case study data, the procedure starts with Facility A making its own production decision first, operating independently without considering the actions or needs of Facility B or Facility C. Facility A determines to fully fulfill the demand of Customers 1, 2, and 3, while maximizing its own profit by using 28 units of the total available 100 units of raw material with an optimal objective function value of 140. This reduces the raw material available to 72 units, and some of the shared customer pool has already been satisfied.

Next, Facility B makes its own production decision with 72 units of raw material available, and Customers 2 and 3 already being fulfilled by Facility A. Given these constraints, Facility B determines to fulfill the demands of Customers 4 and 5 by requesting 24 units of raw material to maximize its market share for Customer 3. Recall that Customer 3 has already been

fulfilled by Facility A, thus Facility B must focus on other demands in its customer pool, although this is not ideal from Facility B's perspective. As a result, Facility B achieves an objective function value of 30, a lower value than if it was able to make the first decision and a reflection of how Facility B's performance is affected by Facility A's prior decision.

With Facilities A and B having completed their decisions, Facility C now proceeds with its production planning. At this point, only 48 units of raw material remain available, and Customers 4 and 5 have already been fully served by Facility B. Given these conditions (constraints), Facility C fulfills the demands of the remaining Customers 6 and 7, with an objective value of 17.5.

This decision by Facility C concludes the pure decentralized decision-making process for the DMS, with the final solution shown in Figure 4.2.

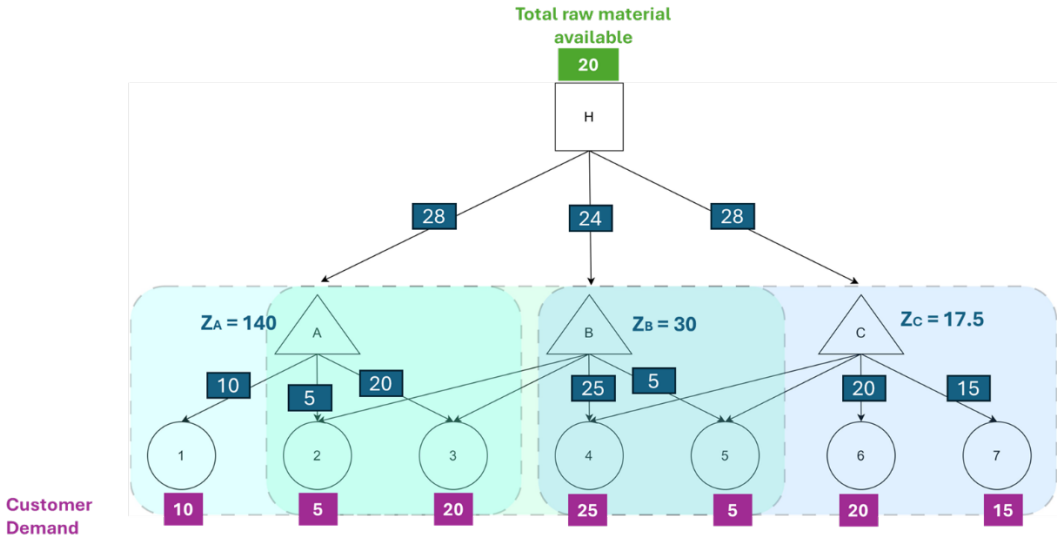

Figure 4.2 Final pure decentralized solution following A, B, C.

In this case, facilities earlier in the sequential decision process have the advantage in achieving the objective, creating intrinsic unfairness and lack of mechanism to achieve the global system-level performance.

We further examine two other orders of decision: B, A, C and C, B, A. Table 4.1 compares the complete results of the three orders in three performance metrics: profitability, market share, and carbon emission, both individually and globally. The bold numbers in the table indicate the best performance metric for the corresponding individual facility or global level.

Table 4.1 Comparison of DMS-D solutions with different orders of decision.

Performance Metrics		**A->B->C**	**B->A->C**	**C->B->A**
Profitability	A	**140**	40	40
	B	247.5	247.5	112.5
	C	273	147	273
	Global	**660.5**	434.5	425.5
Market Share	A	80	0	0
	B	90	90	90
	C	0	0	0
	Global	**170**	90	90
Carbon Emissions	A	17.5	5	5
	B	27.5	27.5	12.5
	C	16.25	**8.75**	16.25
	Global	61.25	41.25	**33.75**

The above results show that since there is no centralized coordination, the order for distributed facilities to make decision has significant effect on the performance of individual facilities and the overall system. Specifically, Facility A achieves most profitability when the order is A, B, C; Facility B is indifferent of the order for maximizing its market share; and the order B, A, C allows Facility C to achieve lowest carbon emissions (while sacrificing the other two metrics). For the overall system performance, the order of A, B, C generates the best

profitability and market share, but also has the highest carbon emissions. The lowest carbon emissions of the system are achieved with the order B, A, C.

Because each facility prioritizes its individual objectives—profit maximization for A, market share for B, and carbon emission reduction for C—it is often not possible for the system to achieve optimal global performance. This independent decision-making approach, without coordination, gives advantage to whoever makes the first move in the decision process, causing competition among distributed facilities with no cooperation. It misses the opportunity of better utilizing the limited resources and aligning facilities' metrics with the company's global objective.

4.3 Case Results: With CACS

The research team implements the sequential negotiation procedure in the centralized-autonomous-coordination scheme (CACS) in Figure 3.5 for the case study, which is called DMS-CACS in the sequel. In this approach, the parent company actively participates in the decision process by revising and adjusting the production plan of a facility after it submits its proposed decision to the parent company. The parent's goal is to: (i) coordinate distributed facilities' decisions to achieve better system-wide global performance; and (ii) respect each facility's decision by minimizing the deviation of the revised plan from the facility's proposed plan.

The DMS-CACS solutions have the following main features.

- **System Approach:** All decisions are evaluated and adjusted by the parent company to address the overall system-wide performance to better align distributed facilities' metrics with the corporate-level objective.
- **Sequential Coordination:** After each facility independently optimizes its production, the parent company reviews and revises the plan to align with the global objective while

minimizing the deviation from the facility's proposed decision. This achieves a better balance between the local and global objectives.

- **Progressive Symmetric Information:** As a facility's decision is reviewed and updated, other facilities gradually gain more information, leading to more informed and coordinated decisions.

The DMS-CACS solutions have the potential of enhancing coordination, reducing inefficiencies, and driving optimal global performance of the DMS. The detailed steps of DMS-CACS—assuming the order A, B, C—for the case study are elaborated below.

Step 1. Facility A begins by independently optimizing its production plan with the objective of maximizing its local profit. In doing so, Facility A requests 28 units of raw material from the parent company (H) and fully satisfies the demands of Customers 1, 2, and 3. This decision results in an objective function value of 140, which matches the outcome in the decentralized solution as shown in Figure 4.3.

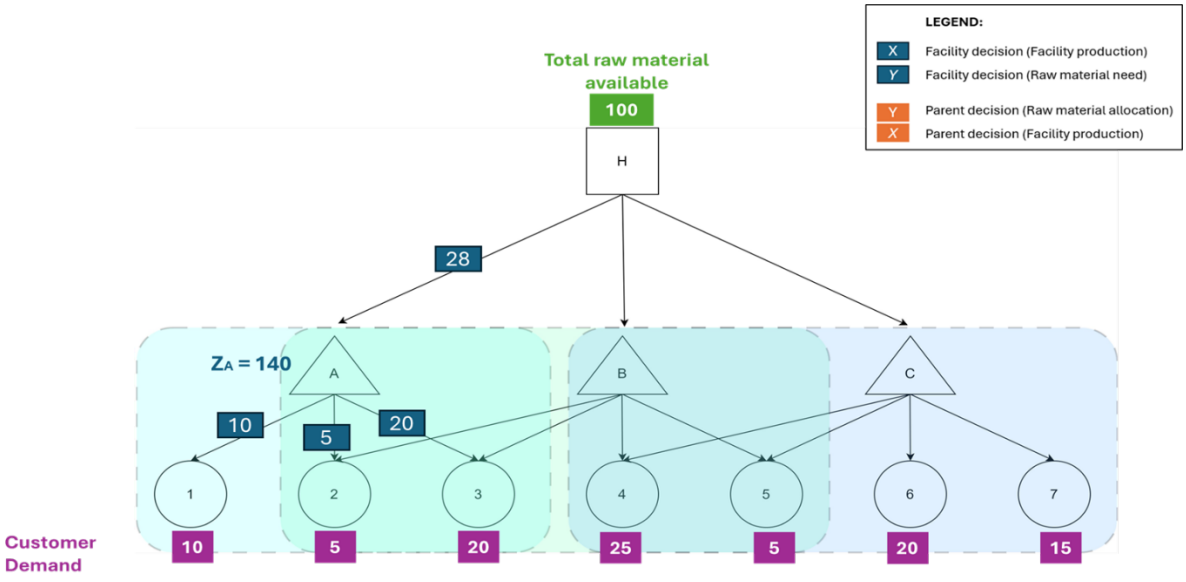

Figure 4.3 Facility A's proposed plan

Step 1R. The parent company reviews Facility A's proposed plan and solves the parent's centralized model in Section 3.2. As shown in Figure 4.4, in the parent company's revised plan, Facility A is assigned to serve Customers 1 and 3. Facility A's raw material allocation is reduced by four units (from 28 to 24), and the parent company allocates the total supply to optimize the decisions of the subsequent facilities. Facility B is now allocated 28 units of raw material, which allows it to fulfill the demands of Customers 2, 4, and 5. Facility C is allocated 28 units of raw material.

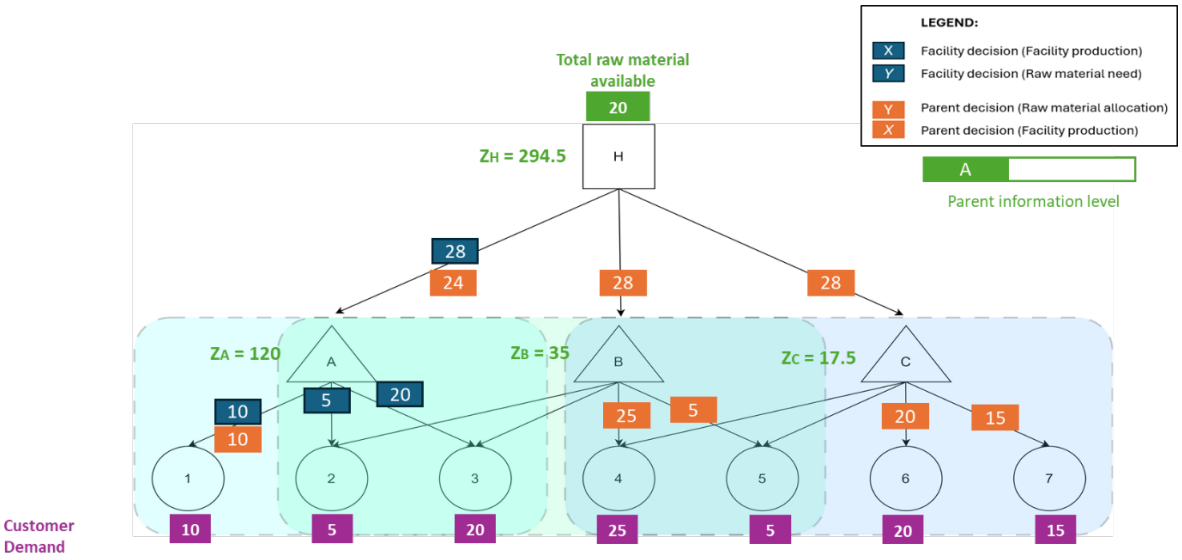

Figure 4.4 Parent company's revised plan after Facility A submits its proposed plan

Step 2. Facility B proceeds to make its independent production decision based on the updated conditions prescribed by the parent company's revision of Facility A's plan. Recall that the parent company has already allocated 28 units of raw material to Facility A and reassigned customer responsibilities, leaving Facility B with a maximum of 28 units of raw material.

Considering this constraint and the fact that Facility A has already fulfilled the demands of Customers 1 and 3, Facility B optimizes its production to focus on serving Customers 2, 4, and 5. To achieve this, Facility B utilizes all the raw material available (28 units available) for its production, resulting in an objective function value of 35. Facility B's proposed plan is shown in Figure 4.5.

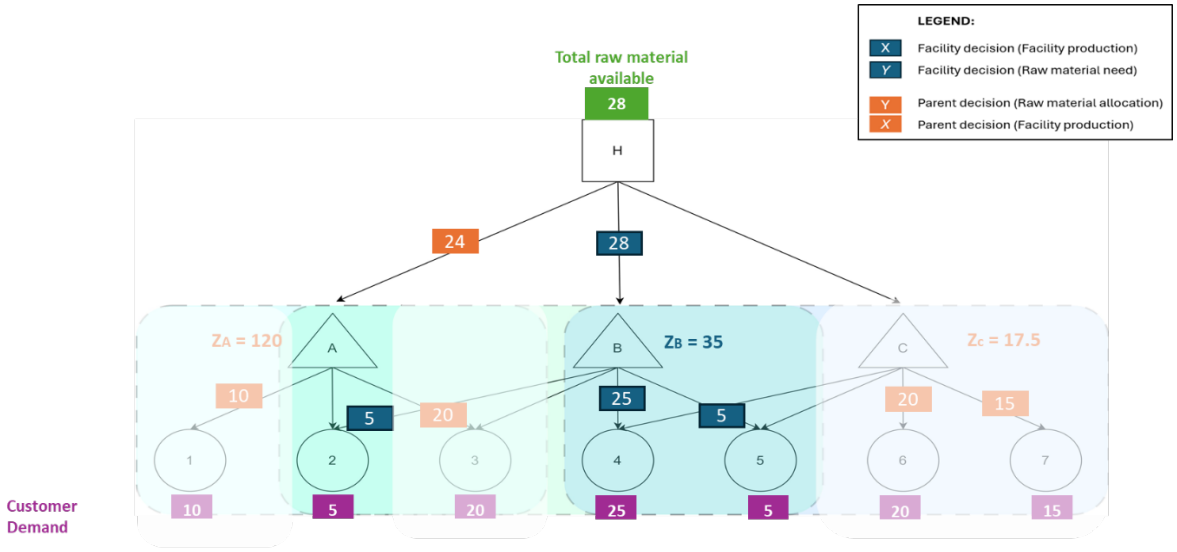

Figure 4.5 Facility B's proposed plan

Step 2R. After Facility B submits its proposed production plan: requesting 24 units of the available 28 units of raw material to fulfill Customers 2, 4, and 5, the parent company intervenes to revise B's proposal. Notably, the parent company has significantly revised B's proposal to allocate more raw material to B and allow it to produce more to serve Customers 2 and 3 (in addition to Customers 4 and 5 in B's original proposal), which takes advantage of B's lowest production cost among the three distributed facilities in the case study. The parent also updates the raw material availability for Facility C to be 28. Figure 4.6 shows the parent company's revision of Facility B's proposal.

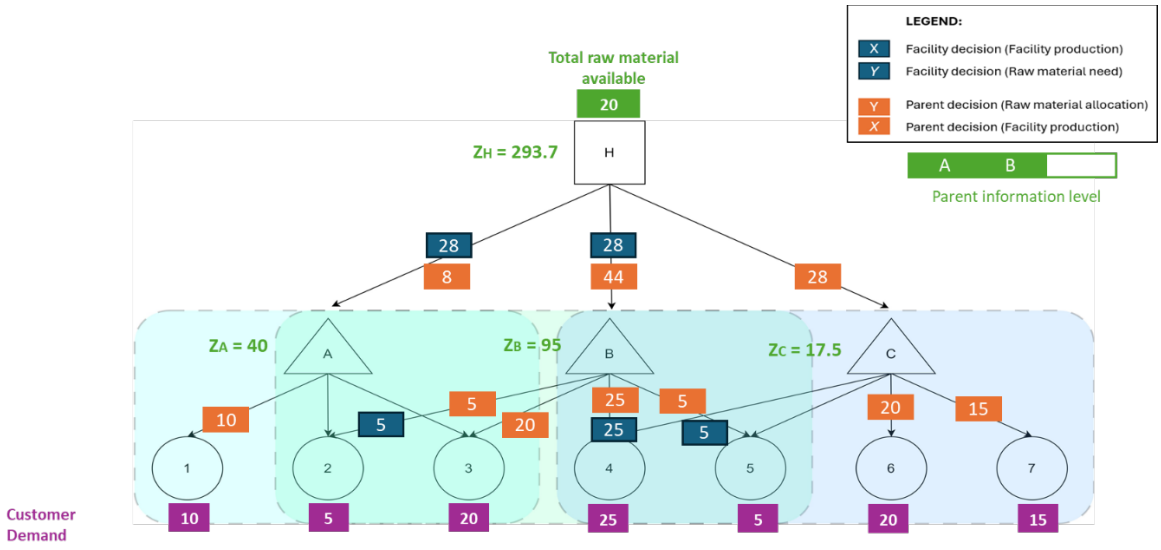

Figure 4.6 Parent company's revised plan after Facility B submits its proposed plan

Step 3. Facility C proceeds with its production decision, serving the set of customers prescribed by the parent's revision on the proposals of Facilities A and B. With 28 units of raw material available, Facility C utilizes all of it to fulfill the demand for Customers 6 and 7 exclusively, as shown in Figure 4.7. The orange numbers in the diagram indicate the adjustments made by the parent during its revision process, ensuring that resources are efficiently distributed and customer demands are met without duplication.

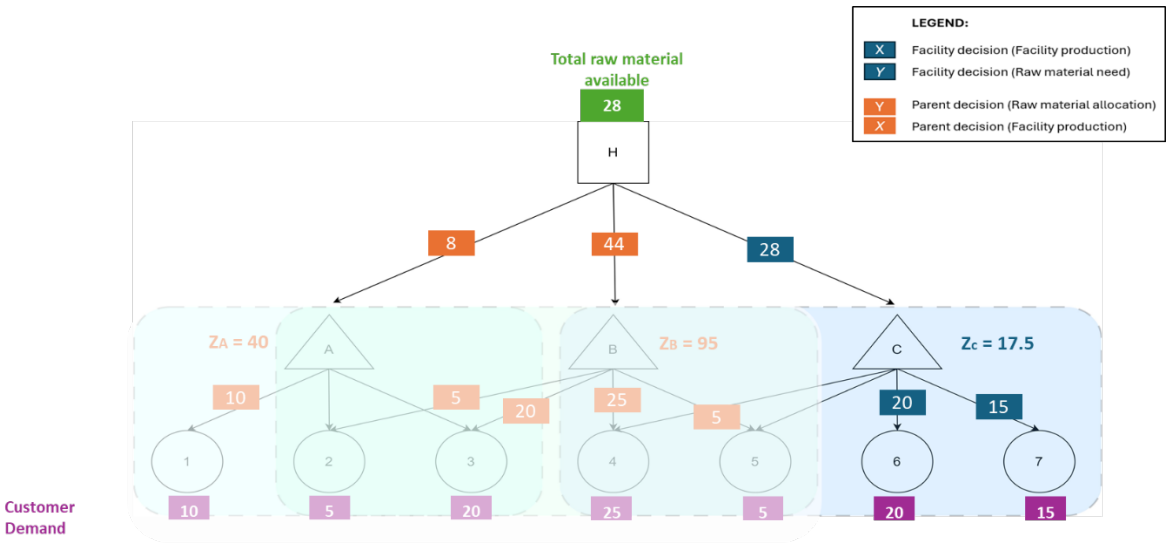

Figure 4.7 Facility C's proposed plan

At this point, all the distributed facilities have completed their production proposals, and all the customer demands are satisfied. The parent company will do a last round of revision.

Step 3R. In the final stage of the centralized decision process, the parent company performs a comprehensive revision of all facilities' production proposals. By this point, the parent company has gathered complete and symmetric information from each facility, enabling it to make informed and optimal decisions. As in Figure 4.8, the final allocation of resources, highlighted in the orange numbers, shows how the parent company strategically redistributes production responsibilities. This redistribution eliminates redundancies, aligns local facility decisions with system-wide objectives, and maximizes the overall performance.

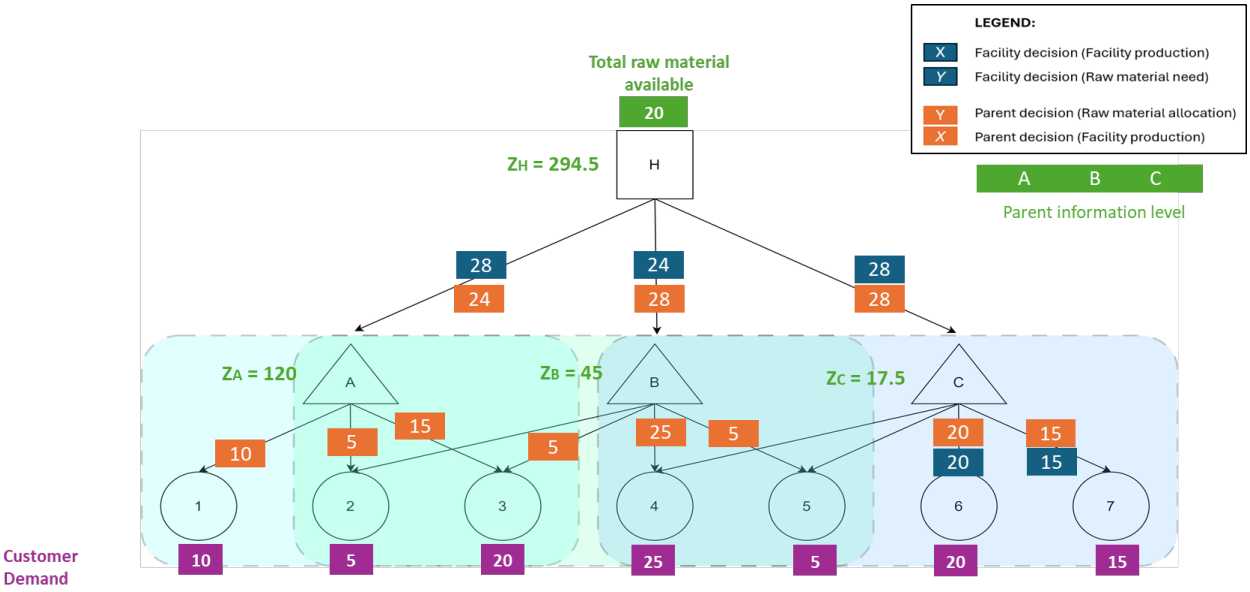

Figure 4.8 Parent company's revised plan after Facility C submits its proposed plan

The final DMS-CACS solution has a global objective of 294.5, which outperforms the global objective of 292 in the DMS-D solution (using the same objective function in the parent company's centralized model).

4.4 Case Results: With Limited Supply of Raw Material

The above scenario in the case study assumes ample capacity of raw material, i.e., key starting materials (KSMs). In real world operations, the capacity of KSM supply of the parent company is often limited. Comparison of DMS-CACS and DMS-D solutions in this case is provided in Table 4.2, assuming the total supply of KMS at the parent company is only 70 kgs. Detailed solution steps for this scenario are provided in Appendix A.

Table 4.2 The DMS-CACS and DMS-D solutions for the case with limited KSM supply

	ABC		CAB		BAC	
	Parent	**No parent**	**Parent**	**No parent**	**Parent**	**No parent**
Total Revenue	437.5	325.0	437.5	375.0	437.5	325.0
Total Raw Cost	70.0	52.0	70.0	60.0	70.0	52.0
Total Manufacturing Cost	67.1	50.0	53.5	62.0	53.5	37.5
Total CO2 Cost	43.8	32.5	43.8	37.5	43.8	32.5
Total Gap Cost	62.5	175.0	62.5	125.0	62.5	175.0
Total Global	**256.6**	**190.5**	**270.3**	**215.5**	**270.3**	**203.0**
Total Deviation (parent-facility)	12.4	-	21.0	-	0.0	-
Obj. fn. For Facility A (profit)	116.7	140.0	0.0	40.0	0.0	40.0
Obj. fn. For Facility B (market share)	40.8	30.0	95.0	**INFEASIBLE**	95.0	95.0
Obj. fn. For Facility C (Cost)	14.6	**INFEASIBLE**	16.3	32.5	16.3	**INFEASIBLE**

Note that the DMS-D approach without centralized coordination has difficulty coming up with feasible solutions for every distributed facility. This is not surprising because the supply can be depleted before the facilities that are last in order can make their decision, which creates infeasible or unrealistic production plans, unfairness, and misses the opportunity of achieving optimal global performance. A DMS with limited supply highlights the need for innovative and intelligent solutions, such as our CACS developed in this project, for effectively managing DMS.

4.5 Discussions and Takeaways

Our case study of a generic drug manufacturer with distributed CM facilities shows the benefit of using CACS for production planning in a DMS. It overcomes the challenge and deficiency caused by each individual facility making autonomous decisions, which leads to either local optima or infeasible, unfair plans. By properly formulating the optimization model of the parent company's centralized problem in Section 3.2, and following the sequential negotiation procedure to implement the CACS as in Figure 3.3, our DMS-CACS solutions are able to better allocate the limited supply of raw material, better prescribe and coordinate each

distributed facility's decision, and improves the system-wide global performance over the benchmark DMS-D solutions.

Chapter 5 Conclusions and Future Study

In this project, we have identified and defined an optimization problem to facilitate and take advantage of distributed manufacturing systems (DMSs) made possible by advancement in manufacturing and information technologies. The DMS has its unique features of localized sourcing and production, autonomous decision-making, and the need for centralized coordination and dynamic decision-support, with diverse applications including API manufacturing, vertical farming, and modular construction.

Our study focuses on the resource allocation and production planning decisions in a supply-production network (SPN) of a DMS, where multiple distributed facilities make their autonomous production decisions with their own (and sometimes conflicting) objectives, and a parent company allocates the supply need for production. The parent company has full visibility and access to all the facilities' information and data, but such information is asymmetric across distributed facilities. The parent company would like to coordinate the facilities' decisions in a centralized way to optimize the system-wide global performance metric(s), while respecting each individual facility's autonomous decision.

We first build an optimization model for the parent company's centralized problem, which takes the resource allocation, global performance metrics, along with each distributed facility's proposed production plan, into consideration. We then propose a novel centralized-autonomous coordination scheme (CACS) and a sequential negotiation procedure to implement CACS for the production planning and resource allocation decisions in the SPN of a DMS. Our approach iteratively allows each distributed facility to make its own production decision and to submit its proposal to the parent company. In each iteration, the parent company reviews the newly submitted proposal of a facility and revises it by solving the centralized coordination model. The approach handles asymmetric information among distributed facilities, a typical

situation in DMSs. It is also able to optimize the global performance metrics while respecting each distributed facility's autonomous decision.

A case study of API manufacturing using the new continuous manufacturing (CM) is performed to examine and assess the performance of our CACS approach, compared to the simple *status quo* approach without coordination. The results show that our CACS solutions have clear advantages over the benchmark solutions without coordination, especially for the scenario with limited supply of raw material. The CACS is able to overcome the challenge and deficiency caused by each individual facility making autonomous decisions, which leads to either local optima or infeasible, unfair plans.

Our work opens the door to future research opportunities. First, the basic CACS and sequential negotiation procedure can be improved to address production planning and resource decisions in multiple time periods, which explicitly handles the need for dynamic decision-support. Secondly, the SPN considered in this study can be extended to a more general supply chain network with multiple tiers and larger number of facilities and players. Last but not least, it will be fruitful to apply our modeling framework and CACS procedure for real world applications in other various sectors including manufacturing, agriculture/food, construction, and healthcare.

References

Adamo, A., Beingessner, R. L., Behnam, M., Chen, J., Jamison, T. F., Jensen, K. F., . . . Stelzer, T. (2016). On-demand continuous-flow production of pharmaceuticals in a compact, reconfigurable system. *Science, 352*(6281), 61-67.

Algorri, M., Abernathy, M. J., Cauchon, N. S., Lamm, T. R., & Moore, C. M. (2022). Re-envisioning pharmaceutical manufacturing: increasing agility for global patient access. *Journal of Pharmaceutical Sciences, 111*(3), 593-607.

Almashaqbeh, M., & El-Rayes, K. (2021). Optimizing the modularization of floor plans in modular construction projects. *Journal of Building Engineering, 39*, 102316.

Anderson, D. (2008). *Build-to-Order & Mass Customization*: CIM Press.

Artemis. (2020). *State of Indoor Farming 2020*. Retrieved from

Aulakh, P. K., Settanni, E., & Srai, J. S. (2021). Continuous manufacturing technologies in upstream pharmaceutical supply chains: Combining engineering and managerial criteria. *Journal of Multi-Criteria Decision Analysis, 29*(3-4), 298-312.

Autogrow. (2020). *Global CEA Census Report*. Retrieved from

Chan, F. T. S., Swarnkar, R., & Tiwari, M. K. (2007). Infrastructure for co-ordination of multi-agents in a network–based manufacturing system. *International Journal of Advanced Manufacturing Technology, 31*, 1028-1033.

Domokos, A., Nagy, B., Szilagyi, B., Marosi, G., & Nagy, Z. K. (2021). Integrated continuous pharmaceutical technologies - A review. *Organic Process Research & Development, 25*, 721-739.

Durach, C. F., Kurpjuweit, S., & Wagner, S. M. (2017). The impact of additive manufacturing on supply chains. *International Journal of Physical Distribution & Logistics Management, 47*(10), 954-971.

Haque, M., Paul, S. K., Sarker, R., & Essam, D. (2020). Managing decentralized supply chain using bilevel with Nash game approach. *Journal of Cleaner Production, 266*, 121865.

Kim, H. M. (2001). *Target Cascading in Optimal System Design.* (Ph.D.). The University of Michigan, Ann Arbor.

Kim, H. M., Kokkolaras, M., Louca, L. S., Delagrammatikas, G. J., Michelena, N. F., Filipi, Z. S., . . . Assanis, D. N. (2002). Target cascading in vehicle redesign: a class VI truck study. *International Journal of Vehicle Design, 29*(3), 199-225.

Kim, H. M., Michelena, N. F., Papalambros, P. Y., & Jiang, T. (2003). Target cascading in optimal system design. *Journal of Mechanical Design, 125*(3), 474-480.

Li, H., Parcell, J., Roach, A., Solomon, A., Cabrera-Garcia, J., Dietterle, L., & Spell, A. (2023). *The Economics and Optimal Design of Missouri Indoor Farming Supply Chains*. Retrieved from

Li, J., Xiong, N., Park, J. H., Liu, C., Ma, S., & Cho, S. (2012). Intelligent model design of cluster supply chain with horizontal cooperation. *Journal of Intelligent Manufacturing, 23*, 917-931.

Liu, P., Liu, C., & Wei, X. (2021). Optimal allocation of shared manufacturing resouces based on bilevel programming. *Discrete Dynamics in Nature and Society*, https://doi.org/10.1155/2021/6474241.

Porter, M. E. (2000). Location, competition, and economic development: Local clusters in a global economy. *Economic Development Quarterly, 14*(1), 15-34.

Ratnayake, R. M. C. (2019). Enabling RDM in challenging environments via additive layer manufacturing: enhancing offshore petroleum asset operations. *Production Planning & Control, 30*(7), 522-539.

Renna, P., & Perrone, G. (2015). Order allocation in a multiple suppliers-manufacturers environment within a dynamic cluster. *International Journal of Advanced Manufacturing Technology, 80*, 171-182.

Roscoe, S., & Blome, C. (2016). *A framework for the adoption of redistributed manufacturing strategies in pharmaceutical supply chains.* Paper presented at the EurOMA 2016.

Shahmoradi-Moghadam, H., & Schonberger, J. (2021). Coordinated allocation production routing problem for mobile supply chains with shared factories. *Computers & Chemical Engineering, 155*, 107501.

Srai, J. S., Graham, G., Hennelly, P., Phillips, W., Kapletia, D., & Lorentz, H. (2020). Distributed manufacturing: a new form of localized production? *International Journal of Operations & Production Management, 40*(6), 697-727.

Srai, J. S., Kumar, M., Graham, G., Phillips, W., Tooze, J., Ford, S., . . . Ravi, B. (2016). Distributed manufacturing: scope, challenges and opportunities. *International Journal of Production Research, 54*(23), 6917-6935.

Tosserams, S., Etman, L. F. P., Papalambros, P. Y., & Rooda, J. E. (2006). An augmented Lagrangian relaxation for analytical target cascading using the alternating direction method of multipliers. *Structural and Multidisciplinary Optimization, 31*, 176-189.

Tosserams, S., Etman, L. F. P., & Rooda, J. E. (2008). Augmented Lagrangian coordination for distributed optimal design in MDO. *International Journal for Numerical Methods in Engineering, 73*, 1885-1910.

Tosserams, S., Etman, L. F. P., & Rooda, J. E. (2010). Multi-modality in augmented Lagrangian coordination for distributed optimal design. *Structural and Multidisciplinary Optimization, 40*, 329-352.

Wang, G., Zhang, G., Guo, X., & Zhang, Y. (2021). Digital twin-driven service model and optimal allocation of manufacturing resources in shared manufacturing. *Journal of Manufacturing Systems, 59*, 165-179.

Xiang, W., Song, F., & Ye, F. (2014). Order allocation for multiple supply-demand networks within a cluster. *Journal of Intelligent Manufacturing, 25*, 1367-1376.

Xue, X., Wei, Z., & Liu, Z. (2012). The impact of service system on the implementation of cluster supply chain. *Service Oriented Computing and Applications, 6*, 215-230.

Yan, B., & Liu, L. (2018). A new transshipment policy in cluster supply chains based on system dynamics. *RAIRO Operations Research, 52*, 1-15.

Yu, C., Xu, X., Yu, S., Sang, Z., Yang, C., & Jiang, X. (2020). Shared manufacturing in the sharing economy: Concept, definition and service operations. *Computers & Industrial Engineering, 146*, 106602.

Appendix A Detailed Steps of the CACS Procedure for the Scenario with Limited KSM Supply

As shown in Figure A.1, Facility A begins by autonomously optimizing its production plan with the objective of maximizing its own profit. It requests 28 units of raw material from the parent company (H) and fully satisfies the demands of Customers 1, 2, and 3. This decision results in an objective function value of 140, which matches the outcome in the decentralized process. Since Facility A is the first facility to make its decision, its plan remains unaffected by the actions of the other facilities.

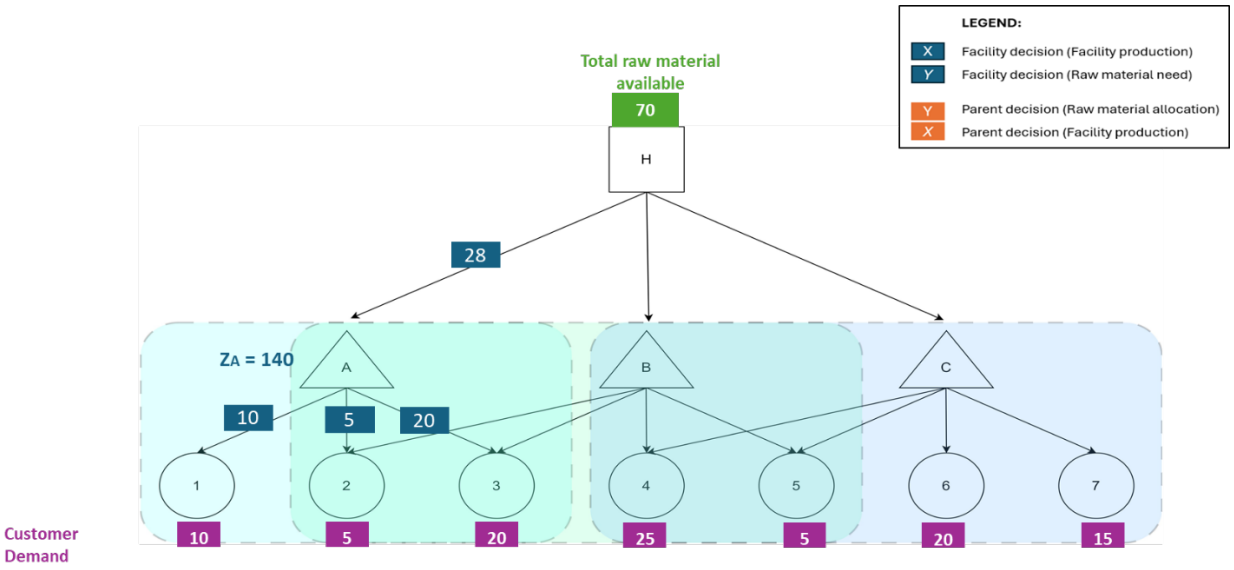

Figure A.1 Facility A's proposed plan

The parent company then reallocates customer fulfillment responsibilities as shown in Figure A.2. Facility A is now assigned to only serve Customer 3. Facility A's raw material allocation decreases to 14 units, but the parent company adjusts the remaining supply to optimize the decisions of the subsequent facilities. Facility B now has 28 units of raw material available, which allows it to fulfill the demands of Customers 2, 4, and 5. This facility also partially fulfills

the remaining demand of Customer 3. Facility B is prioritized, possibly due to the high relevance its production has on the global profit. Facility B is the least expensive in manufacturing and the one with the largest customer base. Facility C is left with 28 units of raw material.

Note that due to the raw material shortage, Customer 1 remains unserved, while Customer 2 is only partially served by Facility B. This outcome may be driven by Facility A's relatively lower profitability within the system, making it less favorable in the centralized allocation process. Additionally, since the model penalizes demand gaps, these two customers contribute to lower penalty costs compared to others with higher demand. As a result, the system prioritizes fulfilling higher-demand customers first, even if it leads to partial or unmet fulfillment for smaller-demand customers.

This revision directly constrains and influences the production decisions of Facilities B and C, guiding them toward solutions that contribute to system-wide optimization. Through this step-by-step revision, the parent company ensures that local facility objectives align with the overarching goals of the entire manufacturing network.

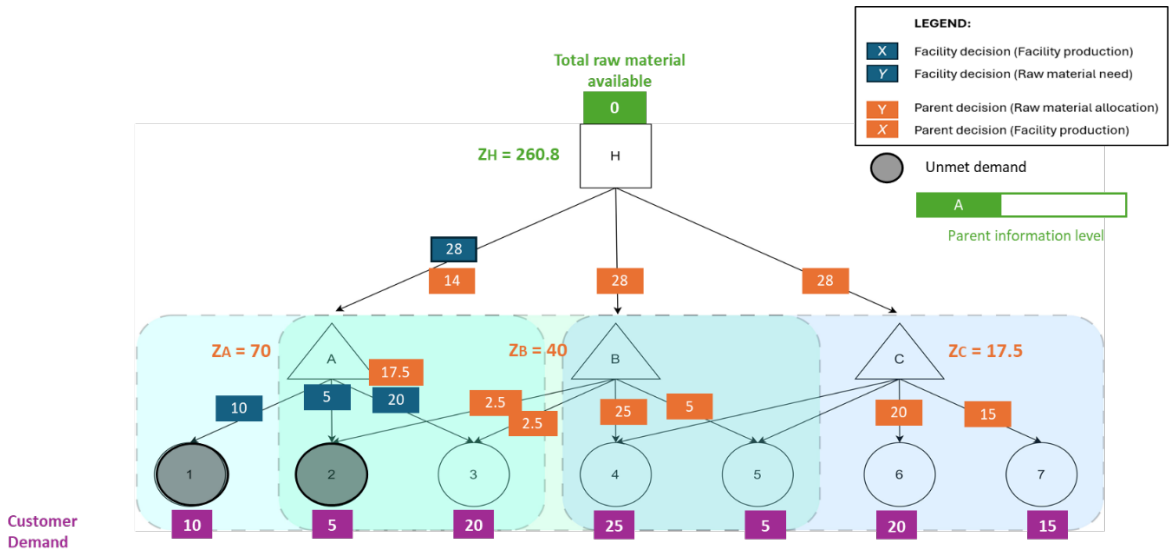

Figure A.2 Parent's revised plan after Facility A submits its proposed plan

Facility B now proceeds to make its autonomous production decision based on the updated conditions set by the parent company's prior revision of Facility A's plan, as shown in Figure A.3. The parent company has already allocated 14 units of raw material to Facility A and reassigned customer responsibilities, leaving Facility B with a maximum of 28 units of raw material. Considering this constraint and the fact that Facility A has partially fulfilled the demands of Customer 3, Facility B optimizes its production to focus on serving Customers 2, 4, and 5. Facility B can also partially fulfill the unmet demand of Customer 3. To meet this demand, Facility B requests 28 units of raw material for its production, resulting in an objective function value of 40.

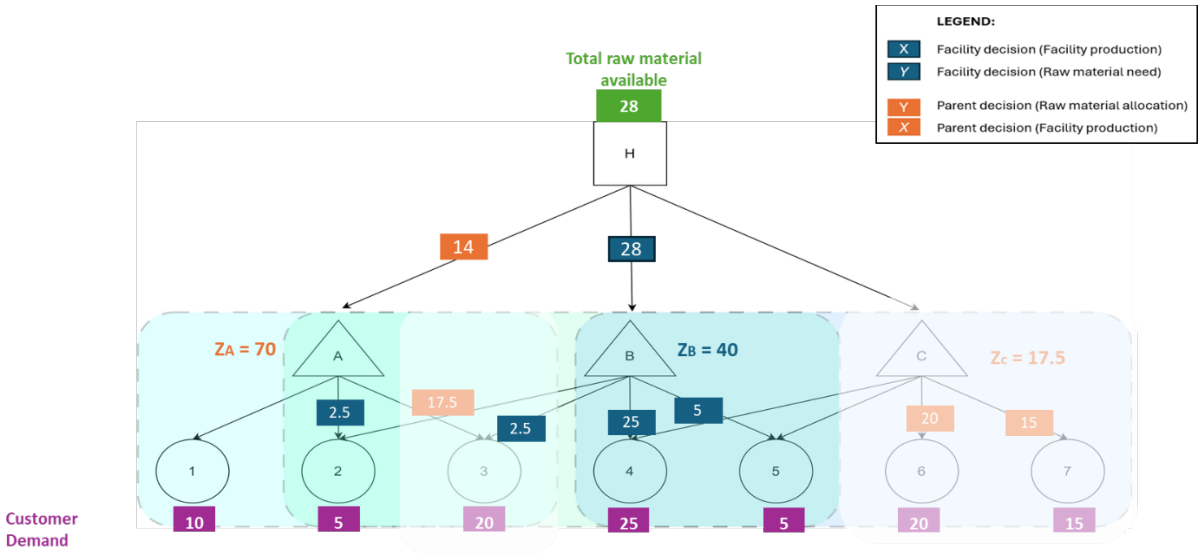

Figure A.3 Facility B's proposed plan

After Facility B submits its proposal: requesting 28 units of raw material to fulfill Customers 2, 3, 4, and 5, the parent company steps in to revise this proposal as shown in Figure A.4. At this point, the parent company holds the updated information from both Facility A and Facility B, allowing it to make more informed adjustments to align with system-wide objectives. Customers 2 and 6 remain partially unfulfilled because they are served by the most expensive facilities in the system (A and C). Since Facility B, the most cost-efficient facility, was prioritized in allocation, it received enough raw materials to fully meet the demands of Customers 3, 4, and 5. Given the supply constraints, the parent company prioritized fulfilling high-demand customers while minimizing cost inefficiencies. As a result, lower-priority demand from Customers 2 and 6 was left partially unfulfilled due to the limited raw materials available for Facilities A and C. The revision also updates the raw material availability for Facility C, leaving it with only 18 units due to the adjustments made for Facilities A and B. These changes will directly influence how Facility C approaches its production planning. By revising Facility

B's plan, the parent company ensures that each facility's operations contribute more effectively to system-wide goals, such as maximizing profit, minimizing carbon emissions, and reducing allocation deviations. This sequential adjustment process fosters a more balanced and optimized production strategy before Facility C proceeds with its decision-making.

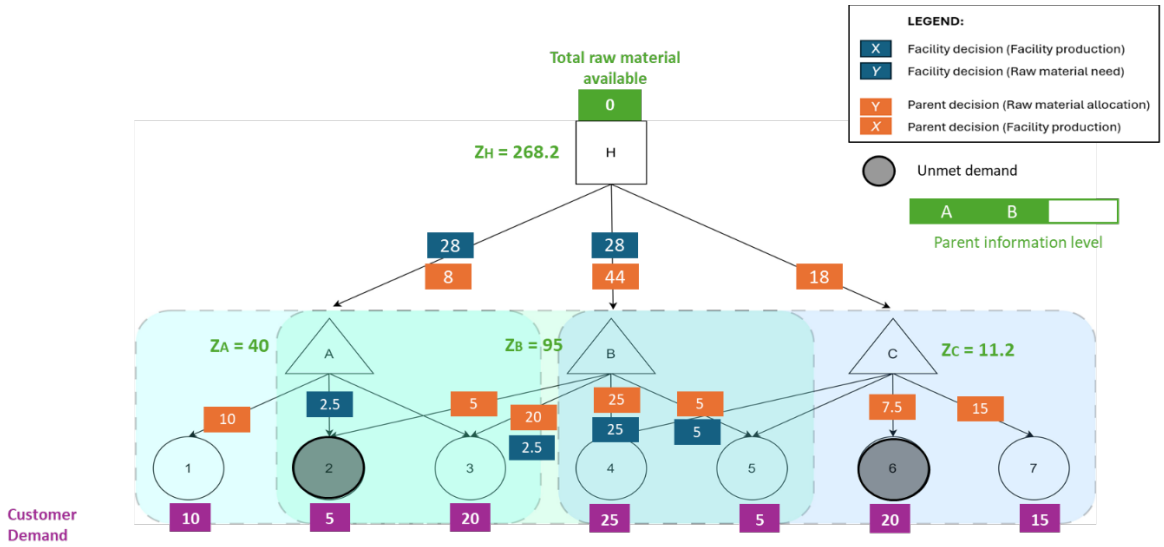

Figure A.4 Parent's revised plan after Facility A submits its proposed plan

Facility C then proceeds with its production decision, operating under the constraints set by the parent company after revising Facilities A and B as shown in Figure A.5. With only 18 units of raw material available, Facility C utilizes all of it to fulfill the demand for Customers 6 and 7 exclusively. However, demand for Customer 6 is only partially fulfilled. This allocation aligns with the parent company's system-wide optimization strategy after the last revision. The orange numbers in the diagram indicate the adjustments made by the parent during its revision process, ensuring that resources are efficiently distributed, and customer demands are met without duplication.

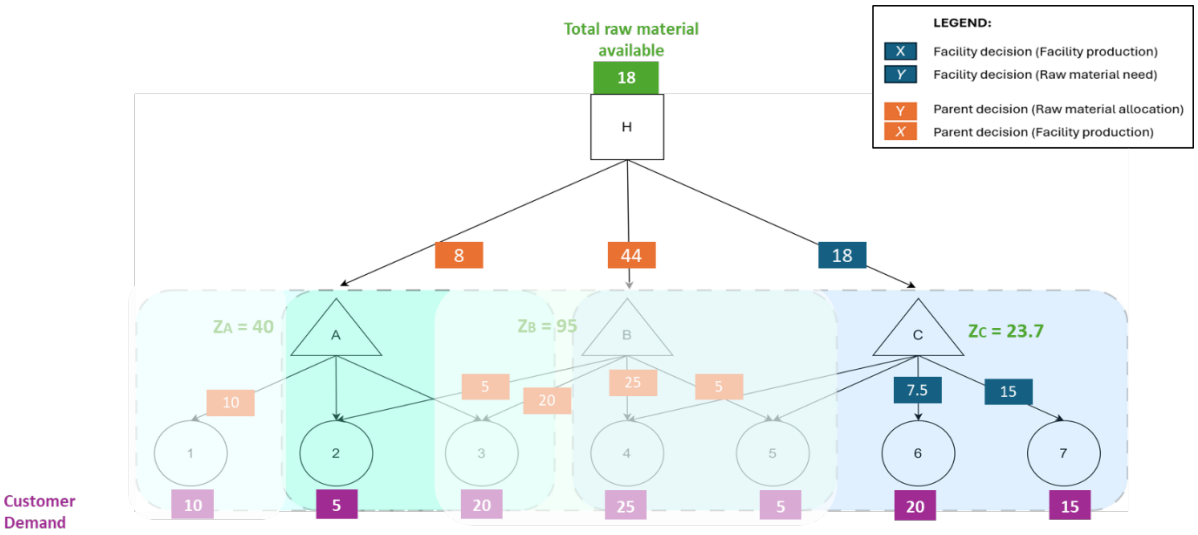

Figure A.5 Facility C's proposed plan

In the final stage of the centralized decision process, the parent company performs a thorough revision of all facilities' production plans. By this point, the parent company has gathered complete and symmetric information from each facility, enabling it to make informed and optimal decisions. As shown in Figure A.6, the final allocation of resources, highlighted in the orange numbers, shows how the parent company strategically redistributed production responsibilities. This redistribution eliminates redundancies, aligns local facility decisions with system-wide objectives, and maximizes overall performance while trying to better serve customers during a raw material supply shortage.

In this case, the parent company has distributed raw material equally among the three facilities: A, B, and C, each receiving 23.3 units. As a result, Customers 4 and 7 remain unfulfilled while other demands are met, due to a limited supply of raw material. The final round of solutions after the parent company revision prescribes the best way to both allocate the limited raw material supply and to determine which customers' demand to fulfill (and which to give up), while achieving the optimal global performance.

It is also interesting to note that a customer's demand can now be fulfilled by multiple distributed facilities, i.e., multiple sourcing. For example, Customer 3's total demand of 30 is met by both Facility A (14.2) and Facility B (5.8), which is also due to the limited total raw material capacity, such that there might exist some customer(s) whose demand cannot be fulfilled by a single facility (single sourcing).

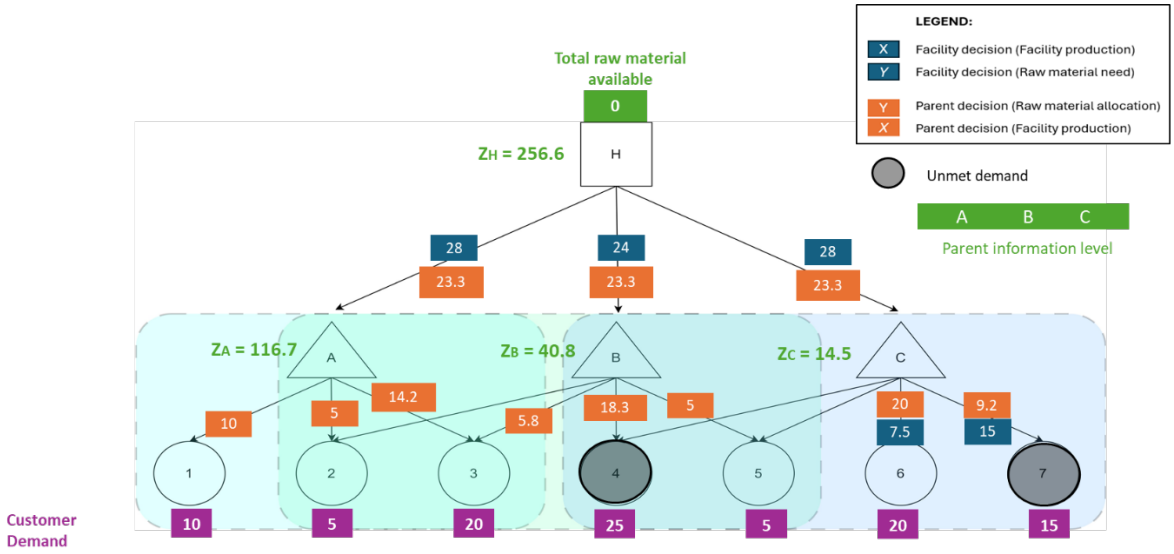

Figure A.6 Parent's revised plan after Facility C submits its proposed plan

Appendix B Python Code for the DMS-D Approach

```
# -*- coding: utf-8 -*-

import xlrd

import gurobipy as gp

from gurobipy import GRB

# %% Load the workbook, and map sheets to variables

dataset = r"C:\Users\paula\OneDrive - University of Missouri\PhD\Assistantship\Dr. Li - Distributed

Manufacturing\Code\PPMM - Small example - Paula Penagos.xls"

datafile = xlrd.open_workbook(dataset)

SetSheet = datafile.sheet_by_name("Sets")

FacCoverage = datafile.sheet_by_name("Facilities coverage")

RawAvail = datafile.sheet_by_name("Raw material avail.")

ManufCap = datafile.sheet_by_name("Manufacturing cap.")

Demand = datafile.sheet_by_name("Demand")

ManufCost = datafile.sheet_by_name("Cost of manufacturing")

Demand = datafile.sheet_by_name("Demand")

Costs = datafile.sheet_by_name("Costs")

# %% %Sets

parents = []

facilities = []

customers = []

products = []

nParent = 1

nFacilities = 3

nCustomers = 7

nProducts = 1


i=1
```

```
while (i < nParent +1):
    try:
        parent = SetSheet.cell_value(0,i).strip()
        parents.append(parent)
        i=i+1
    except IndexError:
        break
print("Set of parent companies: ", parents)
i=1
while (i < nFacilities +1):
    try:
        facility = SetSheet.cell_value(1,i).strip()
        facilities.append(facility)
        i=i+1
    except IndexError:
        break
print("Set of manufacuring facilities: ", facilities)
i=1
while (i < nCustomers +1):
    try:
        customer = int(SetSheet.cell_value(2,i))
        customers.append(customer)
        i=i+1
    except IndexError:
        break
print("Set of customers: ", customers)
i=1
while (i < nProducts +1):
```

```
    try:
        product = SetSheet.cell_value(3,i).strip()
        products.append(product)
        i=i+1
    except IndexError:
        break
print("Set of products: ", products)
# %% Parameters
cust_served = {}
raw_available = {}
manuf_cap = {}
demand_cust = {}
# manuf_cost = 1 #if it is constant
revenue = 5
cost_emission = 0.5
cost_emission_reduced = 0.25
raw_need = 0.8
i = 1
for customer in customers :
    j = 1
    for facility in facilities :
        cust_served[customer, facility] = FacCoverage.cell_value(i, j)
        j+=1
    i+=1
print ("Customers served by facility:", cust_served)


i = 1
for parent in parents :
```

```
    raw_available[parent] = RawAvail.cell_value(1,i)
    i = i+1
print ("Raw material available at parent company:", raw_available)
i = 1
for facility in facilities :
    manuf_cap[facility] = ManufCap.cell_value(i,1)
    i = i+1
print ("Manufacturing capacity at facility:", manuf_cap)
i = 1
for customer in customers :
    demand_cust[customer] = Demand.cell_value(i,1)
    i = i+1
print ("Demand of customer:", demand_cust)
manuf_cost = {}
i = 1
for facility in facilities :
    manuf_cost[facility] = ManufCost.cell_value(i,1)
    i = i+1
print ("Manufacturing cost at facility:", manuf_cost)
print ("Manufacturing cost =", manuf_cost)
print ("Product revenue =", revenue)
print ("Emission cost =", cost_emission)
print ("Raw material needed for manufacturing =", raw_need)
# %%Create Optimization Model for Facility A
Y_A = 0
Y_B = 0
Y_C = 0
```

```
# Create the model -------------------------------
m_A = gp.Model('Facility A')


# Decision Variables -----------------------------------------------
X_A = {}
for p in products:
    for f in facilities:
        for c in customers:
            X_A[p, f, c] = m_A.addVar(vtype=GRB.CONTINUOUS, name="X_A_%s_%s_%s" % (p, f, c))
# Add Constraints ----------------------------------------------------------
for f in facilities:
    if f != "A":
        m_A.addConstr(gp.quicksum(X_A[p, f, c] for p in products for c in customers) == 0, f"Production at {f} initial
stage")
Y_A = raw_need * gp.quicksum(X_A[p, "A", c] for p in products for c in customers) # Raw material needed
constraint (direct calculation without introducing a new decision variable)
for p in parents:
    m_A.addConstr(Y_A <= (raw_available[p] - Y_B - Y_C), "Raw material availability")


m_A.addConstr(gp.quicksum(X_A[p, "A", c] for p in products for c in customers) <= manuf_cap["A"],
"Manufacturing capacity at facility A")
for c in customers:
    m_A.addConstr(gp.quicksum(X_A[p, 'B', c] + X_A[p, 'C', c] for p in products for c in customers) +
                  gp.quicksum(X_A[p, 'A', c] for p in products) <= demand_cust[c] * cust_served[c, "A"], "Demand
customer")


# Objective function -----------------------------------------------
```

```
sum_Profit_A = gp.quicksum((revenue - manuf_cost[f]) * X_A[p, f, c]
                    for p in products
                    for f in facilities if f == "A"
                    for c in customers)
m_A.setObjective(sum_Profit_A, GRB.MAXIMIZE)


m_A.setParam('MIPGap', 0.001)
m_A.setParam('Timelimit', 36000)
m_A.optimize()
# %%
# Retrieve solution attributes for model A
X_A_out = m_A.getAttr('X', X_A)
# Print the solution attributes
print("Solution attributes for Model A:")
for key, value in X_A_out.items():
    print(f"{key}: {value}")
Y_values = {}
# Iterate over each facility
for facility in facilities:
    # Calculate Y for the current facility based on the optimized values of the decision variables
    Y_value = raw_need * gp.quicksum(X_A_out[p, facility, c] for p in products for c in customers)
    # Store the calculated Y value for the current facility in the dictionary
    Y_values[facility] = Y_value
# Access the Y values for each facility as needed
Y_A_value = Y_values["A"]
Y_B_value = Y_values["B"]
Y_C_value = Y_values["C"]
```

```
for facility, Y_value in Y_values.items():
    print(f"Y value for Facility {facility}: {Y_value}")


# Retrieve the solution from Model A
solution_A = {}
for v in m_A.getVars():
    solution_A[v.varName] = v.x
# Print the solution attributes from Model A
print("Solution from Model A:")
for var_name, var_value in solution_A.items():
    print(f"{var_name}: {var_value}")


obj_value_A = m_A.objVal
print("Objective function value for Model B:", obj_value_A)
#%% Create Optimization Model for Facility B


# # Create the model -------------------------------
m_B = gp.Model('Facility B')


# Decision Variables ------------------------------------------------
X_B = {}
for p in products:
    for f in facilities:
        for c in customers:
            X_B[p, f, c] = m_B.addVar(vtype=GRB.CONTINUOUS, name="X_B_%s_%s_%s" % (p, f, c))


# # Add Constraints ------------------------------------------------------------
```

```
Y_B = raw_need * gp.quicksum(X_B[p, "B", c] for p in products for c in customers) # Raw material needed
constraint (direct calculation without introducing a new decision variable)
for p in parents:
    m_B.addConstr(Y_B <= (raw_available[p] - Y_A_value - Y_C_value), "Raw material availability")

m_B.addConstr(gp.quicksum(X_B[p, "B", c] for p in products for c in customers) <= manuf_cap["B"],
"Manufacturing capacity at facility B")
for c in customers:
    m_B.addConstr(gp.quicksum(X_B[p, 'B', c] for p in products) <= demand_cust[c] * cust_served[c, "B"] -
gp.quicksum(X_B[p, 'A', c] + X_B[p, 'C', c] for p in products for c in customers), "Demand customer")

# Add constraint for Facility B: X_B + X_A_out <= demand
for c in customers:
    for p in products:
        # Add the constraint for each combination of customer and product
        m_B.addConstr(X_B[p, "B", c] + X_A_out[p, "A", c] <= demand_cust[c] , f"Constraint_demand_{p}_{c}")
# Objective function -----------------------------------------------
sum_Profit_B = gp.quicksum(X_B[p, f, c] * (3 if c == 3 else 1)
                           for p in products
                           for f in facilities if f == "B"
                           for c in customers)
m_B.setObjective(sum_Profit_B, GRB.MAXIMIZE)
m_B.setParam('MIPGap', 0.001)
m_B.setParam('Timelimit', 36000)
m_B.optimize()

# Retrieve solution attributes
X_B_out = m_B.getAttr('X', X_B)
```

```
print("SOLUTION FOR MODEL B ABOVE")
# Corrected part to retrieve solution attributes
if m_B.status == GRB.OPTIMAL:
    print("SOLUTION FOR MODEL B ABOVE")
    # Print the solution attributes directly
    for v in m_B.getVars():
        print(f'{v.varName} = {v.x}')
else:
    print("Model B optimization was not successful.")
Y_values_B = {}
# Iterate over each facility
for facility in facilities:
    # Calculate Y for the current facility based on the optimized values of the decision variables
    Y_value_B = raw_need * gp.quicksum(X_B_out[p, facility, c] for p in products for c in customers)
    # Store the calculated Y value for the current facility in the dictionary
    Y_values_B[facility] = Y_value_B

# Access the Y values for each facility as needed
Y_A_value_B = Y_values_B["A"]
Y_B_value_B = Y_values_B["B"]
Y_C_value_B = Y_values_B["C"]

for facility, Y_value_B in Y_values_B.items():
    print(f"Y value for Facility {facility}: {Y_value_B}")

obj_value_B = m_B.objVal
print("Objective function value for Model B:", obj_value_B)
```

```
# %%
# Retrieve solution attributes for model B
X_B_out = m_B.getAttr('X', X_B)
# Print the solution attributes
print("Solution attributes for Model B:")
for key, value in X_B_out.items():
    print(f"{key}: {value}")
# Retrieve the solution from Model A
solution_B = {}
for v in m_B.getVars():
    solution_B[v.varName] = v.x
# Print the solution attributes from Model A
print("Solution from Model B:")
for var_name, var_value in solution_B.items():
    print(f"{var_name}: {var_value}")


#%% Create Optimization Model for Facility C


# # Create the model -------------------------------
m_C = gp.Model('Facility C')


# Decision Variables -----------------------------------------------
X_C = {}
for p in products:
    for f in facilities:
        for c in customers:
            X_C[p, f, c] = m_C.addVar(vtype=GRB.CONTINUOUS, name="X_C_%s_%s_%s" % (p, f, c))
```

```
# # Add Constraints ----------------------------------------------------------

Y_C = raw_need * gp.quicksum(X_C[p, "C", c] for p in products for c in customers) # Raw material needed
constraint (direct calculation without introducing a new decision variable)
for p in parents:
    m_C.addConstr(Y_C <= (raw_available[p] - Y_A_value_B - Y_A_value - Y_C_value_B - Y_C_value), "Raw
material availability")

m_C.addConstr(gp.quicksum(X_C[p, "C", c] for p in products for c in customers) <= manuf_cap["C"],
"Manufacturing capacity at facility B")

for c in customers:
    m_C.addConstr(gp.quicksum(X_C[p, 'C', c] for p in products) <= demand_cust[c] * cust_served[c, "C"] -
gp.quicksum(X_C[p, 'A', c] + X_C[p, 'B', c] for p in products for c in customers), "Demand customer")

# Add constraint for Facility B: X_B + X_A_out <= demand
for c in customers:
    for p in products:
        # Add the constraint for each combination of customer and product
        m_C.addConstr(X_C[p, "C", c] + X_A_out[p, "A", c] + X_B_out[p, "B", c]>= demand_cust[c] ,
f"Constraint_demand_{p}_{c}")

# Objective function -----------------------------------------------
sum_Cost_C = gp.quicksum(cost_emission * X_C[p, f, c]
                         for p in products
                         for f in facilities if f == "C"
                         for c in customers)
```

```
m_C.setObjective(sum_Cost_C, GRB.MINIMIZE)
m_C.setParam('MIPGap', 0.001)
m_C.setParam('Timelimit', 36000)
m_C.optimize()
# Retrieve solution attributes
X_C_out = m_C.getAttr('X', X_C)
print("SOLUTION FOR MODEL C ABOVE")
# Corrected part to retrieve solution attributes
if m_C.status == GRB.OPTIMAL:
    print("SOLUTION FOR MODEL C ABOVE")
    # Print the solution attributes directly
    for v in m_C.getVars():
        print(f'{v.varName} = {v.x}')
else:
    print("Model C optimization was not successful.")


Y_values_C = {}
# Iterate over each facility
for facility in facilities:
    # Calculate Y for the current facility based on the optimized values of the decision variables
    Y_value_C = raw_need * gp.quicksum(X_C_out[p, facility, c] for p in products for c in customers)
    # Store the calculated Y value for the current facility in the dictionary
    Y_values_C[facility] = Y_value_C


# Access the Y values for each facility as needed
Y_A_value_C = Y_values_B["C"]
Y_B_value_C = Y_values_B["C"]
Y_C_value_C = Y_values_B["C"]
```

```
for facility, Y_value_C in Y_values_C.items():
    print(f"Y value for Facility {facility}: {Y_value_C}")

obj_value_C = m_C.objVal
print("Objective function value for Model C:", obj_value_C)

#%%  Output file

OutputFile = open(r"C:\Users\paula\OneDrive - University of Missouri\PhD\Assistantship\Dr. Li - Distributed
Manufacturing\Code\PPMM - Small example - output ABC.txt", "w")

# Production Summary for Model A
OutputFile.write('\n********Production Planning Results*********\n')
OutputFile.write('\nDataset: ' + dataset)
OutputFile.write('\n\n')
OutputFile.write('\n\n ***Production Summary for Model A:\n\n')

for f in facilities:
    OutputFile.write('\n\n Production at Facility %s:\n\n' % f)
    for p in products:
        for c in customers:
            OutputFile.write('  Customer %d: %d units\n' % (c, X_A_out[p, f, c]))

# Total Production for Model A at Each Facility
total_production_A = {f: gp.quicksum(X_A_out[p, f, c] for p in products for c in customers).getValue() for f in
facilities}
OutputFile.write('\n\nTotal Production for Model A at Each Facility:\n\n')
```

```
for f, value in total_production_A.items():
    OutputFile.write('Facility %s: %d units\n' % (f, value))

# Values of Y for Model A (Facility Assignment)
OutputFile.write('\n\nValues of Y for Model A (Raw Material Assignment):\n\n')
OutputFile.write(f'Facility A: {Y_A_value}\n')
OutputFile.write(f'Facility B: {Y_B_value}\n')
OutputFile.write(f'Facility C: {Y_C_value}\n')

OutputFile.write('\n\n \n\n')
# Production Summary for Model B
OutputFile.write('\n\n ****Production Summary for Model B:\n\n')

for f in facilities:
    OutputFile.write('\n\nProduction at Facility %s:\n\n' % f)
    for p in products:
        for c in customers:
            OutputFile.write('  Customer %d: %d units\n' % (c, X_B_out[p, f, c]))

# Total Production for Model B at Each Facility
total_production_B = {f: gp.quicksum(X_B_out[p, f, c] for p in products for c in customers).getValue() for f in
facilities}
OutputFile.write('\n\nTotal Production for Model B at Each Facility:\n\n')

for f, value in total_production_B.items():
    OutputFile.write('Facility %s: %d units\n' % (f, value))

# Values of Y for Model B (Facility Assignment)
```

```
OutputFile.write('\n\nValues of Y for Model B (Raw Material Assignment):\n\n')
OutputFile.write(f'Facility A: {Y_A_value_B}\n')
OutputFile.write(f'Facility B: {Y_B_value_B}\n')
OutputFile.write(f'Facility C: {Y_C_value_B}\n')

# Production Summary for Model C
OutputFile.write('\n\n ***Production Summary for Model C:\n\n')

for f in facilities:
    OutputFile.write('\n\n Production at Facility %s:\n\n' % f)
    for p in products:
        for c in customers:
            OutputFile.write('  Customer %d: %d units\n' % (c, X_C_out[p, f, c]))

# Total Production for Model A at Each Facility
total_production_C = {f: gp.quicksum(X_C_out[p, f, c] for p in products for c in customers).getValue() for f in
facilities}
OutputFile.write('\n\nTotal Production for Model C at Each Facility:\n\n')

for f, value in total_production_C.items():
    OutputFile.write('Facility %s: %d units\n' % (f, value))

# Values of Y for Model A (Facility Assignment)
OutputFile.write('\n\nValues of Y for Model C (Raw Material Assignment):\n\n')

for facility, Y_value_C in Y_values_C.items():
    OutputFile.write(f"Y value for Facility {facility}: {Y_value_C} \n")
```

```
OutputFile.write('\n\n' )
# Sum of X_B and X_A_out for Each Customer and Product
OutputFile.write('\n ****** Sum of X_B and X_A_out and X_C_out for Each Customer and Product:\n\n')
for c in customers:
    for p in products:
        sum_X_B_X_A_out_X_C_out = X_B_out[p, "B", c] + X_A_out[p, "A", c] + X_C_out[p, "C", c]
        OutputFile.write(f'Customer {c}, Product {p}: {sum_X_B_X_A_out_X_C_out} units\n')

# Sum of Y_value and Y_value_B for Each Customer and Product
OutputFile.write('\n\nSum of Raw Material Assigned for Each Customer and Product:\n\n')

for f in facilities:
        sum_Y_value_Y_value_B_Y_value_V = Y_values[f] + Y_values_B[f]  + Y_values_C[f]
        OutputFile.write(f'Facility {f}: {sum_Y_value_Y_value_B_Y_value_V} units\n')
# Total Production for Model A at Each Facility
total_production = {f: gp.quicksum(X_A_out[p, f, c] + X_B_out[p, f, c] +X_C_out[p, f, c] for p in products for c in
customers).getValue() for f in facilities}
OutputFile.write('\n\nTotal Production at Each Facility:\n\n')

for f, value in total_production.items():
    OutputFile.write('Facility %s: %d units\n' % (f, value))
# Objective function values
OutputFile.write('\n\n ***** Objective function values:')
OutputFile.write('\n\nObjective function value for Model A - Max. Profit: ' + str(obj_value_A))
OutputFile.write('\n\nObjective function value for Model B - Max. Market Share of a customer: ' +
str(obj_value_B))
OutputFile.write('\n\nObjective function value for Model C - Min. Emission Cost: ' + str(obj_value_C))
```

```
## Metrics ----------
OutputFile.write('\n\n *****General metrics:**** \n\n')
#Total profit
OutputFile.write("\n\nTotal profit")
total_production = {}
total_profit = 0
for f in facilities:
    total_production[f] = gp.quicksum(X_B_out[p, f, c] + X_A_out[p, f, c] + X_C_out[p, f, c] for p in products for c
in customers)
    facility_profit = total_production[f].getValue() * (revenue - manuf_cost[f])
    total_profit += facility_profit

    OutputFile.write("\n")
    OutputFile.write("Total production at Facility " + f + ": " + str(total_production[f].getValue()) + "\n")
    OutputFile.write("Total profit at Facility " + f + ": " + str(facility_profit) + "\n")

OutputFile.write("\n")
OutputFile.write("Total profit across all facilities: " + str(total_profit) + "\n")

#Customer 3 market sharet
OutputFile.write("\n\n\nMarket share - Customer 3 \n")
total_share=0
total_market= 0
for f in facilities:
    for p in products:
        production_ABC3 = X_A_out[p, f, 3] + X_B_out[p, f, 3] + X_C_out[p, f, 3]
        market_ABC3 = (revenue - manuf_cost[f]) * production_ABC3
        OutputFile.write('Customer %d at Facility %s: %d units\n' % (3, f, production_ABC3))
```

```
        OutputFile.write('Customer %d at Facility %s: %d profit units\n' % (3, f, market_ABC3)+'\n')
        total_share += production_ABC3
        total_market += market_ABC3

OutputFile.write("\n")
OutputFile.write("Total market share across all facilities: " + str(total_share) + "\n")
OutputFile.write("Total market share profit across all facilities: " + str(total_market) + "\n")

# Carbon emission cost
OutputFile.write("\n\n\nEmission costs:")
total_emission_cost = 0
for f in facilities:
    total_production[f] = gp.quicksum(X_B_out[p, f, c] + X_A_out[p, f, c] + X_C_out[p, f, c] for p in products for c
in customers)
    emission_cost = total_production[f].getValue() * (cost_emission_reduced if f == "C" else cost_emission)
    total_emission_cost += emission_cost

    OutputFile.write("\n")
    OutputFile.write("Total production at Facility " + f + ": " + str(total_production[f].getValue()) + "\n")
    OutputFile.write("Total emission cost at Facility " + f + ": " + str(emission_cost) + "\n")
OutputFile.write("\n")
OutputFile.write("Total emission cost across all facilities: " + str(total_emission_cost) + "\n")
OutputFile.close()
```

Appendix C Python Code for the DMS-CACS Approach

```
# -*- coding: utf-8 -*-

import xlrd

import gurobipy as gp

from gurobipy import GRB

import itertools

# import random

# %% Load the workbook, and map sheets to variables

dataset = r"C:\Users\paula\OneDrive - University of Missouri\PhD\Assistantship\Dr. Li\Distributed

Manufacturing\Code\PPMM - Small example - Paula Penagos.xls"

datafile = xlrd.open_workbook(dataset)

SetSheet = datafile.sheet_by_name("Sets")

FacCoverage = datafile.sheet_by_name("Facilities coverage")

RawAvail = datafile.sheet_by_name("Raw material avail.")

ManufCap = datafile.sheet_by_name("Manufacturing cap.")

Demand = datafile.sheet_by_name("Demand")

ManufCost = datafile.sheet_by_name("Cost of manufacturing")

Demand = datafile.sheet_by_name("Demand")

Costs = datafile.sheet_by_name("Costs")


# %% %Sets

parents = []

facilities = []

customers = []

products = []

nParent = 1

nFacilities = 3

nCustomers = 7
```

```
nProducts = 1
i=1
while (i < nParent +1):
    try:
        parent = SetSheet.cell_value(0,i).strip()
        parents.append(parent)
        i=i+1
    except IndexError:
        break
print("Set of parent companies: ", parents)
i=1
while (i < nFacilities +1):
    try:
        facility = SetSheet.cell_value(1,i).strip()
        facilities.append(facility)
        i=i+1
    except IndexError:
        break
print("Set of manufacuring facilities: ", facilities)
i=1
while (i < nCustomers +1):
    try:
        customer = int(SetSheet.cell_value(2,i))
        customers.append(customer)
        i=i+1
    except IndexError:
        break
```

```
print("Set of customers: ", customers)
i=1
while (i < nProducts +1):
    try:
        product = SetSheet.cell_value(3,i).strip()
        products.append(product)
        i=i+1
    except IndexError:
        break
print("Set of products: ", products)
# %% Parameters
cust_served = {}
raw_available = {}
manuf_cap = {}
demand_cust = {}
# manuf_cost = 1 #if it is constant
revenue = 5
cost_emission = 0.5
cost_emission_reduced = 0.25
raw_need = 0.8
cost_raw = 1
raw_available = 100
weight_deviation_raw = 0.3
weight_demand_fulfillment = 1
i = 1
for customer in customers :
    j = 1
    for facility in facilities :
```

```
        cust_served[customer, facility] = FacCoverage.cell_value(i, j)
        j+=1
    i+=1
print ("Customers served by facility:", cust_served)
# i = 1
# for parent in parents :
#     raw_available[parent] = RawAvail.cell_value(1,i)
#     i = i+1
# print ("Raw material available at parent company:", raw_available)
i = 1
for facility in facilities :
    manuf_cap[facility] = ManufCap.cell_value(i,1)
    i = i+1
print ("Manufacturing capacity at facility:", manuf_cap)
i = 1
for customer in customers :
    demand_cust[customer] = Demand.cell_value(i,1)
    i = i+1
print ("Demand of customer:", demand_cust)
manuf_cost = {}
i = 1
for facility in facilities :
    manuf_cost[facility] = ManufCost.cell_value(i,1)
    i = i+1
print ("Manufacturing cost at facility:", manuf_cost)
print ("Manufacturing cost =", manuf_cost)
print ("Product revenue =", revenue)
print ("Emission cost =", cost_emission)
```

```
print ("Raw material needed for manufacturing =", raw_need)

#%% MANUAL TRIAL WITH THE VARIABLE SAVE

Y_A = 0

Y_B = 0

Y_C = 0

Y_A_OUT = 0

Y_B_OUT = 0

Y_C_OUT = 0

# Initialize X_out to an empty dictionary

X_out = {}

for p in products:

    for f in facilities:

        for c in customers:

            # X_out[p, f, c] = random.randint(1, 20)

            X_out[p, f, c] = 0

print("X_out: ", X_out, "\n\n")

X_out_raw ={}

for p in products:

    for f in facilities:

        for c in customers:

            # X_out_raw[p, f, c] = random.randint(1, 20)

            X_out_raw[p, f, c] = 0

print("X_out: ", X_out_raw, "\n\n")

#%% facility A

print("----------------\n\n Facility A \n\n ---------------- \n\n")

Y_A = Y_A_OUT

Y_B = Y_B_OUT

Y_C = Y_C_OUT
```

```
print("\n\n ------------------------\n\n")

print("Y value for Facility A: " , Y_A)

print("Y value for Facility B: " , Y_B)

print("Y value for Facility C: " , Y_C)

print("----------------")

i = i + 1

f_a =1

f_b=0

f_c=0

print(f"Round {i} :")

# Create the model -------------------------------

m = gp.Model('Facility GENERALIZED')

# Decision Variables ----------------------------------------------

X = {}

for p in products:

    for f in facilities:

        for c in customers:

            X[p, f, c] = m.addVar(vtype=GRB.CONTINUOUS, name="X_%s_%s_%s" % (p, f, c))

# Add Constraints ------------------------------------------------------------


for f in facilities:

    if (f == "A" and f_a == 0) or (f == "B" and f_b == 0) or (f == "C" and f_c == 0):

        m.addConstr(gp.quicksum(X[p, f, c] for p in products for c in customers) == 0, f"Production at {f} initial

stage")

if f_a == 1:

    Y_A = raw_need * gp.quicksum(X[p, "A", c] for p in products for c in customers)  # Raw material needed

constraint
```

```
elif f_b == 1:
    Y_B =  raw_need * gp.quicksum(X[p, "B", c] for p in products for c in customers)  # Raw material needed
constraint
elif f_c == 1:
    Y_C = raw_need * gp.quicksum(X[p, "C", c] for p in products for c in customers)  # Raw material needed
constraint
for p in parents:
    m.addConstr(f_a * Y_A + f_b * Y_B + f_c * Y_C <= (raw_available - (f_a - 1) * Y_A + (f_b - 1) * Y_B + (f_c -
1) * Y_C), "Raw material availability")
if f_a == 1:
    m.addConstr(gp.quicksum(X[p, "A", c] for p in products for c in customers) <= manuf_cap["A"], "Manufacturing
capacity at facility A")
elif f_b == 1:
    m.addConstr(gp.quicksum(X[p, "B", c] for p in products for c in customers) <= manuf_cap["B"], "Manufacturing
capacity at facility B")
elif f_c == 1:
    m.addConstr(gp.quicksum(X[p, "C", c] for p in products for c in customers) <= manuf_cap["C"], "Manufacturing
capacity at facility C")
for c in customers:
    if f_a == 1:
        if cust_served[c, "A"] == 0:
            m.addConstr(
                gp.quicksum(X[p, 'A', c] for p in products) == demand_cust[c] * cust_served[c, "A"],
                "Demand customer"
                )
        if cust_served[c, "A"] == 1:
            m.addConstr(
                gp.quicksum(X[p, 'A', c] for p in products) +
```

```
            gp.quicksum(X_out[p,f,c] for p in products for f in facilities if f != "A") <= demand_cust[c] *
cust_served[c, "A"],
            "Demand customer"
            )
    elif f_b == 1:
        if cust_served[c, "B"] == 0:
          m.addConstr(
            gp.quicksum(X[p, 'B', c] for p in products) == demand_cust[c] * cust_served[c, "B"],
            "Demand customer"
            )
        if cust_served[c, "B"] == 1:
           m.addConstr(
            gp.quicksum(X[p, 'B', c] for p in products) +
            gp.quicksum(X_out[p,f,c] for p in products for f in facilities if f != "B") <= demand_cust[c] *
cust_served[c, "B"],
            "Demand customer"
          )

    elif f_c == 1:
        if cust_served[c, "C"] == 0:
          m.addConstr(
            gp.quicksum(X[p, 'C', c] for p in products) == demand_cust[c] * cust_served[c, "C"],
            "Demand customer"
            )
        if cust_served[c, "C"] == 1:
              m.addConstr(
            gp.quicksum(X[p, 'C', c] for p in products) +
```

```
            gp.quicksum(X_out[p,f,c] for p in products for f in facilities if f != "C") >= demand_cust[c] *
cust_served[c, "C"],
            "Demand customer"
        )
# Objective function -----------------------------------------------
objective = 0
# Sum profit if a == 1 (for facility A)
if f_a == 1:
    sum_Profit_A = gp.quicksum((revenue - manuf_cost[f]) * X[p, f, c]
                               for p in products
                               for f in facilities if f == "A"
                               for c in customers)
    objective += sum_Profit_A  # Maximize profit for A

# Sum profit if b == 1 (for facility B)
elif f_b == 1:
    sum_Profit_B = gp.quicksum(X[p, f, c] * (3 if c == 3 else 1)
                               for p in products
                               for f in facilities if f == "B"
                               for c in customers)
    objective += sum_Profit_B  # Maximize profit for B

# Sum cost if c == 1 (for facility C)
if f_c == 1:
    sum_Cost_C = gp.quicksum(cost_emission * X[p, f, c]
                             for p in products
                             for f in facilities if f == "C"
                             for c in customers)
```

```
    objective -= sum_Cost_C  # Minimize cost for C
# Set the objective for the model
m.setObjective(objective, GRB.MAXIMIZE)
m.write(r"C:\Users\paula\OneDrive - University of Missouri\PhD\Assistantship\Dr. Li\Distributed
Manufacturing\Code\FACILITY problems.lp")

# Set parameters and optimize the model
m.setParam('MIPGap', 0.001)
m.setParam('Timelimit', 36000)
m.optimize()
print("Round: ", i, "\n")
if m.status == GRB.OPTIMAL:
    print("Optimal objective value:", m.ObjVal)
    # Retrieve solution attributes from the model
    X_out_raw = m.getAttr('X', X)

    # Update global_X_out with the current X_out values
    for key, value in X_out_raw.items():
        X_out_raw[key] = value

    print("X_out_raw updated: ", X_out_raw, "\n\n")
    # print("X_out updated: ", X_out, "\n\n")

    for p in products:
        for f in facilities:
            for c in customers:
                # X_out[p, f, c] = random.randint(1, 20)
                X_out[p, f, c] = X_out[p, f, c]  + X_out_raw[p, f, c]
```

```
print("X_out updated: ", X_out, "\n\n")

print("\n\n ------------------------\n\nSUMMARY OF RESULTS: \n")

for p in products:
    for c in customers:
        for f in facilities:
            value = X_out[p, f, c]
            if value != 0:
                print(f"The value of X[{p}, {f}, {c}] is: {value}")

Y_values = {}

# Iterate over each facility

for facility in facilities:
    # Calculate Y for the current facility based on the optimized values of the decision variables
    Y_value = raw_need * gp.quicksum(X_out_raw[p, facility, c] for p in products for c in customers)
    # Store the calculated Y value for the current facility in the dictionary
    Y_values[facility] = Y_value

# Access the Y values for each facility as needed

Y_A_value = Y_values["A"]

Y_B_value = Y_values["B"]

Y_C_value = Y_values["C"]

print("\n\n ------------------------\n\n")

print("Y value for Facility A: " , Y_A_value)

print("Y value for Facility B: " , Y_B_value)

print("Y value for Facility C: " , Y_C_value)


Y_A_OUT = Y_A_OUT* f_a + Y_A_value

Y_B_OUT = Y_B_OUT* f_b + Y_B_value

Y_C_OUT = Y_C_OUT* f_c + Y_C_value
```

```
    print("\n\n ------------------------\n\n")
    print("Y value for Facility A: " , Y_A_OUT)
    print("Y value for Facility B: " , Y_B_OUT)
    print("Y value for Facility C: " , Y_C_OUT)

    Y_available = raw_available - Y_A_OUT - Y_B_OUT - Y_C_OUT
    print("\n\n ----- RAW available after all: ", Y_available)
    # Retrieve the solution from Model A
    solution = {}
    for v in m.getVars():
        solution[v.varName] = v.x
# Reset the previously run model
m.reset()
#%% Parent company problem --> FAC A REVISION
print("----------------\n\n Parent revision after Facility A \n\n ---------- \n\n")

# Create the model -------------------------------
m_P = gp.Model('Parent P')
# Decision Variables ---------------------------------------------
Y = {}
for f in facilities:
    Y[f] = m_P.addVar(vtype=GRB.CONTINUOUS, name="Y_%s" % (f))
 X_parent = {}
for p in products:
    for f in facilities:
        for c in customers:
            X_parent[p, f, c] = m_P.addVar(vtype=GRB.CONTINUOUS, name="X_parent_%s_%s_%s" % (p, f, c))
```

```
deviation = {}
for f in facilities:
    deviation[f] = m_P.addVar(vtype=GRB.CONTINUOUS, name="deviation_%s" % (f))
D = {} #Demand gap
for c in customers:
    for p in products:
        D[p, c] = m_P.addVar(vtype=GRB.CONTINUOUS, name="D_%s_%s" % (p, c))
# Add Constraints ----------------------------------------------------------------
for f in facilities:
    m_P.addConstr(gp.quicksum(raw_need *  X_parent[(p, f, c)]  for p in products for c in customers) <= Y[f] ,
"Supply availability")
m_P.addConstr(gp.quicksum(Y[f] for f in facilities) <= raw_available, "Supply availability")
for f in facilities:
    for p in products:
        for c in customers:
            m_P.addConstr(X_parent[(p, f, c)] <= raw_available *  cust_served[c,f])
for f in facilities:
    m_P.addConstr(deviation[f] >= (Y[f] - gp.quicksum(raw_need * X_out[(p, f, c)] for p in products for c in
customers)), name="Deviation_Constr_Pos")
for f in facilities:
    m_P.addConstr(deviation[f] >= (-1) * (Y[f] - gp.quicksum(raw_need * X_out[(p, f, c)] for p in products for c in
customers)), name="Deviation_Constr_Neg")
for c in customers:
    for p in products:
        m_P.addConstr((gp.quicksum(X_parent[p, f, c] for f in facilities) + D[p,c] ) >= demand_cust[c] , f"Demand
constraint {c}")
        m_P.addConstr(((-1)*gp.quicksum(X_parent[p, f, c] for f in facilities) + D[p,c] ) >= (-1)*demand_cust[c] ,
f"Demand constraint {c}")
```

```
# Objective function -----------------------------------------------
total_revenue = 0
for p in products:
    for f in facilities:
        for c in customers:
            total_revenue += revenue * X_parent[p, f, c]
total_raw_cost = 0
for f in facilities:
    total_raw_cost += cost_raw * Y[f]
total_manuf_cost = 0
for p in products:
    for f in facilities:
        for c in customers:
            total_manuf_cost += manuf_cost[f] * X_parent[p, f, c]
total_co2_cost =0
for p in products:
    for f in facilities:
        for c in customers:
            total_co2_cost += cost_emission * X_parent[p, f, c]
total_deviation = 0
for f in facilities:
    total_deviation += weight_deviation_raw*deviation[f]
total_gap = 0
gap_cost= 5
for c in customers:
    for p in products:
        total_gap += weight_demand_fulfillment*gap_cost*D[p,c]
```

```
total_global = total_revenue - total_raw_cost - total_manuf_cost - total_co2_cost

objective =  total_global - total_deviation - total_gap

profit =  total_global - total_gap

m_P.setObjective(objective, GRB.MAXIMIZE)

m_P.write(r"C:\Users\paula\OneDrive - University of Missouri\PhD\Assistantship\Dr. Li\Distributed

Manufacturing\Code\PARENT problems.lp")

m_P.setParam('MIPGap', 0.001)

m_P.setParam('Timelimit', 36000)

m_P.optimize()

# Print Y and D variables

if m_P.status == GRB.OPTIMAL:

    print("Optimized values for Y and Deviation:")

    for f in facilities:

        print(f"Y[{f}] = {Y[f].X}")

        print(f"deviation[{f}] = {deviation[f].X}")

    for p in products:

        for f in facilities:

            for c in customers:

                print(f"X_parent[{p}, {f}, {c}] = {X_parent[p, f, c].X}")

    print("\n\n ---------- \n\nObjective function values:\n")

    # Calculate total_revenue

    total_revenue = 0

    for p in products:

        for f in facilities:

            for c in customers:

                # total_revenue += revenue * X_parent[p, f, c].X  ##This is actually not all what they produced but what

they sell.
```

```
            if X_parent[p, f, c].X > demand_cust[c]:
                total_revenue += revenue * demand_cust[c]
            else:
                total_revenue += revenue * X_parent[p, f, c].X

# Calculate total_raw_cost
total_raw_cost = 0
for f in facilities:
    total_raw_cost += cost_raw * Y[f].X

# Calculate total_manuf_cost
total_manuf_cost = 0
for p in products:
    for f in facilities:
        for c in customers:
            total_manuf_cost += manuf_cost[f] * X_parent[p, f, c].X

# Calculate total_co2_cost
total_co2_cost = 0
for p in products:
    for f in facilities:
        for c in customers:
            total_co2_cost += cost_emission * X_parent[p, f, c].X
# Calculate total_deviation
total_deviation = 0
for f in facilities:
    total_deviation += deviation[f].X
```

```
total_gap = 0
gap_cost= 5
for c in customers:
    for p in products:
        total_gap += gap_cost*D[p,c].X
# Calculate total_global
total_global = total_revenue - total_raw_cost - total_manuf_cost - total_co2_cost
# Print the extracted values
print(f"Total Revenue: {total_revenue}")
print(f"Total Raw Cost: {total_raw_cost}")
print(f"Total Manufacturing Cost: {total_manuf_cost}")
print(f"Total CO2 Cost: {total_co2_cost}")
print(f"\nTotal Global: {total_global}")
print(f"\nTotal Global: {total_global}")
print(f"Total Deviation (without weight): {total_deviation}")
print(f"Total Gap cost (without weight): {total_gap}")
print("\n\n ---------- \n\nFacility objective values for global optima:\n")
sum_Profit_A_parent = 0
for p in products:
    for f in facilities:
        if f == 'A':
            for c in customers:
                sum_Profit_A_parent += ((revenue - manuf_cost[f]) * X_parent[p, f, c].X)
print(f"Obj. fn. For Facility A (profit): {sum_Profit_A_parent}")
```

```
sum_Profit_B_parent = 0
for p in products:
    for f in facilities:
        if f == 'B':
            for c in customers:
                if c == 3:
                    sum_Profit_B_parent += (3 * X_parent[p, f, c].X)
                else:
                    sum_Profit_B_parent += (1 * X_parent[p, f, c].X)
print(f"Obj. fn. For Facility B (profit): {sum_Profit_B_parent}")
sum_Profit_B_parent = 0
for p in products:
    for f in facilities:
        if f == 'B':
            for c in customers:
                if c == 3:
                    sum_Profit_B_parent += (3 * X_parent[p, f, c].X)


                else:
                    sum_Profit_B_parent += (1 * X_parent[p, f, c].X)
print(f"Obj. fn. For Facility B (profit): {sum_Profit_B_parent}")


sum_Cost_C_parent = 0
for p in products:
    for f in facilities:
        if f == 'C':
            for c in customers:
                sum_Cost_C_parent += (cost_emission * X_parent[p, f, c].X)
```

```
print(f'Obj. fn. For Facility C (Cost): {sum_Cost_C_parent}')
print("\n\n ---------- \n\nDifferences between X_parent and X_out variables:\n")
differences_found = False  # To track if any differences are found
for p in products:
    for f in facilities:
        for c in customers:
            # Get the optimized value from the model and the corresponding value from X_out
            X_parent_val = X_parent[p, f, c].X
            X_out_val = X_out[p, f, c]
            # Compare the two values
            if abs(X_parent_val - X_out_val) > 1e-6:  # Using a small threshold to avoid floating-point issues
                print(f"Difference found for [{p}, {f}, {c}]: X_parent = {X_parent_val}, X_out = {X_out_val}")
                differences_found = True
if not differences_found:
    print("No differences found between X_parent and X_out.")


# Calculate total_co2_cost
total_production = 0
for p in products:
    for f in facilities:
        for c in customers:
            total_production += X_parent[p, f, c].X
print(total_production)
for c in customers:
    for p in products:
        if D[p,c].X != 0:  # Checking if the demand gap for customer c is non-zero
            print(f"Customer {c} for product {p} has a demand gap of {D[p,c].X}")
```

```
    # Extract values for Y[A] and Y[B]
    Y_A_OUT = Y['A'].X
    Y_B_OUT = Y['B'].X
    Y_C_OUT = Y['C'].X

    print(f"Y_A_OUT = {Y_A_OUT}")
    print(f"Y_B_OUT = {Y_B_OUT}")
    print(f"Y_C_OUT = {Y_C_OUT}")

    Y_available = raw_available - Y_A_OUT - Y_B_OUT - Y_C_OUT
    print("\n\n ----- RAW available after all: ", Y_available)
else:
    print("Optimization was not successful.")
m_P.reset()
#%% facility b
print("----------------\n\n Facility B \n\n ---------- \n\n")
Y_A = Y_A_OUT
Y_B = Y_B_OUT
Y_C = Y_C_OUT
print("\n\n ----------\n\n")
print("Y value for Facility A: " , Y_A)
print("Y value for Facility B: " , Y_B)
print("Y value for Facility C: " , Y_C)
print("----------")
i = i + 1
```

```
f_a =0
f_b=1
f_c=0
print(f"Round {i} :")
# Create the model -------------------------------
m = gp.Model('Facility GENERALIZED')
# Decision Variables ----------------------------------------------
X = {}
for p in products:
    for f in facilities:
        for c in customers:
            X[p, f, c] = m.addVar(vtype=GRB.CONTINUOUS, name="X_%s_%s_%s" % (p, f, c))
# Add Constraints ----------------------------------------------------------------
for f in facilities:
    if (f == "A" and f_a == 0) or (f == "B" and f_b == 0) or (f == "C" and f_c == 0):
        m.addConstr(gp.quicksum(X[p, f, c] for p in products for c in customers) == 0, f"Production at {f} initial
stage")
if f_a == 1:
    Y_A = raw_need * gp.quicksum(X[p, "A", c] for p in products for c in customers)  # Raw material needed
constraint
elif f_b == 1:
    Y_B =  raw_need * gp.quicksum(X[p, "B", c] for p in products for c in customers)  # Raw material needed
constraint
elif f_c == 1:
    Y_C = raw_need * gp.quicksum(X[p, "C", c] for p in products for c in customers)  # Raw material needed
constraint
```

```
for p in parents:
    m.addConstr(f_a * Y_A + f_b * Y_B + f_c * Y_C <= 44, "Raw material availability")
if f_a == 1:
    m.addConstr(gp.quicksum(X[p, "A", c] for p in products for c in customers) <= manuf_cap["A"], "Manufacturing
capacity at facility A")
elif f_b == 1:
    m.addConstr(gp.quicksum(X[p, "B", c] for p in products for c in customers) <= manuf_cap["B"], "Manufacturing
capacity at facility B")
elif f_c == 1:
    m.addConstr(gp.quicksum(X[p, "C", c] for p in products for c in customers) <= manuf_cap["C"], "Manufacturing
capacity at facility C")
for c in customers:
    if f_a == 1:
        if cust_served[c, "A"] == 0:
            m.addConstr(
                gp.quicksum(X[p, 'A', c] for p in products) == demand_cust[c] * cust_served[c, "A"],
                "Demand customer"
                )
        if cust_served[c, "A"] == 1:
             m.addConstr(
                 gp.quicksum(X[p, 'A', c] for p in products) +
                 gp.quicksum(X_out[p,f,c] for p in products for f in facilities if f != "A") <= demand_cust[c] *
cust_served[c, "A"],
                 "Demand customer"
                 )
    elif f_b == 1:
        if cust_served[c, "B"] == 0:
            m.addConstr(
```

```
            gp.quicksum(X[p, 'B', c] for p in products) == demand_cust[c] * cust_served[c, "B"],
            "Demand customer"
            )
        if cust_served[c, "B"] == 1:
           m.addConstr(
            gp.quicksum(X[p, 'B', c] for p in products) +
            gp.quicksum(X_out[p,f,c] for p in products for f in facilities if f != "B") <= demand_cust[c] *
cust_served[c, "B"],
            "Demand customer"
           )

    elif f_c == 1:
        if cust_served[c, "C"] == 0:
          m.addConstr(
            gp.quicksum(X[p, 'C', c] for p in products) == demand_cust[c] * cust_served[c, "C"],
            "Demand customer"
            )
        if cust_served[c, "C"] == 1:
                m.addConstr(
            gp.quicksum(X[p, 'C', c] for p in products) +
            gp.quicksum(X_out[p,f,c] for p in products for f in facilities if f != "C") >= demand_cust[c] *
cust_served[c, "C"],
            "Demand customer"
           )
```

```
# Objective function -----------------------------------------------

objective = 0


# Sum profit if a == 1 (for facility A)

if f_a == 1:

    sum_Profit_A = gp.quicksum((revenue - manuf_cost[f]) * X[p, f, c]

                               for p in products

                               for f in facilities if f == "A"

                               for c in customers)

    objective += sum_Profit_A  # Maximize profit for A


# Sum profit if b == 1 (for facility B)

elif f_b == 1:

    sum_Profit_B = gp.quicksum(X[p, f, c] * (3 if c == 3 else 1)

                               for p in products

                               for f in facilities if f == "B"

                               for c in customers)

    objective += sum_Profit_B  # Maximize profit for B


# Sum cost if c == 1 (for facility C)

if f_c == 1:

    sum_Cost_C = gp.quicksum(cost_emission * X[p, f, c]

                             for p in products

                             for f in facilities if f == "C"

                             for c in customers)

    objective -= sum_Cost_C  # Minimize cost for C


# Set the objective for the model
```

```
m.setObjective(objective, GRB.MAXIMIZE)
m.write(r"C:\Users\paula\OneDrive - University of Missouri\PhD\Assistantship\Dr. Li\Distributed
Manufacturing\Code\FACILITY problems.lp")
# Set parameters and optimize the model
m.setParam('MIPGap', 0.001)
m.setParam('Timelimit', 36000)
m.optimize()
print("Round: ", i, "\n")

if m.status == GRB.OPTIMAL:
    print("Optimal objective value:", m.ObjVal)

    # Retrieve solution attributes from the model
    X_out_raw = m.getAttr('X', X)

    # Update global_X_out with the current X_out values
    for key, value in X_out_raw.items():
        X_out_raw[key] = value

    print("X_out_raw updated: ", X_out_raw, "\n\n")

    # print("X_out updated: ", X_out, "\n\n")

    for p in products:
        for f in facilities:
            for c in customers:
                # X_out[p, f, c] = random.randint(1, 20)
                X_out[p, f, c] = X_out[p, f, c]  + X_out_raw[p, f, c]
```

```
print("X_out updated: ", X_out, "\n\n")

print("\n\n ------------------------\n\nSUMMARY OF RESULTS: \n")

for p in products:

    for c in customers:

        for f in facilities:

            value = X_out[p, f, c]


            if value != 0:

                print(f"The value of X[{p}, {f}, {c}] is: {value}")

Y_values = {}

# Iterate over each facility

for facility in facilities:

    # Calculate Y for the current facility based on the optimized values of the decision variables

    Y_value = raw_need * gp.quicksum(X_out_raw[p, facility, c] for p in products for c in customers)

    # Store the calculated Y value for the current facility in the dictionary

    Y_values[facility] = Y_value

# Access the Y values for each facility as needed

Y_A_value = Y_values["A"]

Y_B_value = Y_values["B"]

Y_C_value = Y_values["C"]

print("\n\n ------------------------\n\n")

print("Y value for Facility A: " , Y_A_value)

print("Y value for Facility B: " , Y_B_value)

print("Y value for Facility C: " , Y_C_value)


Y_A_OUT =  Y_A_value

Y_B_OUT =  Y_B_value

Y_C_OUT =  Y_C_value
```

```
    print("\n\n ------------------------\n\n")
    print("Y value for Facility A: " , Y_A_OUT)
    print("Y value for Facility B: " , Y_B_OUT)
    print("Y value for Facility C: " , Y_C_OUT)

    # Y_available = raw_available - Y_A_OUT - Y_B_OUT - Y_C_OUT
    # print("\n\n ----- RAW available after all: ", Y_available)

    # Retrieve the solution from Model A
    solution = {}
    for v in m.getVars():
        solution[v.varName] = v.x
# Reset the previously run model
m.reset()
#%% Parent company problem --> FAC B REVISION
print("----------------\n\n Parent revision after Facility B \n\n ---------- \n\n")
# Create the model -------------------------------
m_P = gp.Model('Parent P')
# Decision Variables ----------------------------------------------
Y = {}
for f in facilities:
    Y[f] = m_P.addVar(vtype=GRB.CONTINUOUS, name="Y_%s" % (f))
X_parent = {}
for p in products:
    for f in facilities:
        for c in customers:
            X_parent[p, f, c] = m_P.addVar(vtype=GRB.CONTINUOUS, name="X_parent_%s_%s_%s" % (p, f, c))
```

```
deviation = {}
for f in facilities:
    deviation[f] = m_P.addVar(vtype=GRB.CONTINUOUS, name="deviation_%s" % (f))
D = {} #Demand gap
for c in customers:
    for p in products:
        D[p, c] = m_P.addVar(vtype=GRB.CONTINUOUS, name="D_%s_%s" % (p, c))
# Add Constraints ----------------------------------------------------------------
for f in facilities:
    m_P.addConstr(gp.quicksum(raw_need *  X_parent[(p, f, c)]  for p in products for c in customers) <= Y[f] ,
"Supply availability")

# m_P.addConstr(gp.quicksum(Y[f] for f in facilities) <= raw_available, "Supply availability")
for f in facilities:
    if f == "A":
        m_P.addConstr(Y[f] == 28)
m_P.addConstr(gp.quicksum(Y[f] for f in facilities) <= raw_available, "Supply availability")
for f in facilities:
    for p in products:
        for c in customers:
            m_P.addConstr(X_parent[(p, f, c)] <= raw_available *  cust_served[c,f])
for f in facilities:
    m_P.addConstr(deviation[f] >= (Y[f] - gp.quicksum(raw_need * X_out[(p, f, c)] for p in products for c in
customers)), name="Deviation_Constr_Pos")
for f in facilities:
    m_P.addConstr(deviation[f] >= (-1) * (Y[f] - gp.quicksum(raw_need * X_out[(p, f, c)] for p in products for c in
customers)), name="Deviation_Constr_Neg")
```

```
for c in customers:
    for p in products:
        m_P.addConstr((gp.quicksum(X_parent[p, f, c] for f in facilities) + D[p,c] ) >= demand_cust[c] , f"Demand
constraint {c}")
        m_P.addConstr(((-1)*gp.quicksum(X_parent[p, f, c] for f in facilities) + D[p,c] ) >= (-1)*demand_cust[c] ,
f"Demand constraint {c}")
# Objective function -----------------------------------------------
total_revenue = 0
for p in products:
    for f in facilities:
        for c in customers:
            total_revenue += revenue * X_parent[p, f, c]
total_raw_cost = 0
for f in facilities:
    total_raw_cost += cost_raw * Y[f]
total_manuf_cost = 0
for p in products:
    for f in facilities:
        for c in customers:
            total_manuf_cost += manuf_cost[f] * X_parent[p, f, c]
total_co2_cost =0
for p in products:
    for f in facilities:
        for c in customers:
            total_co2_cost += cost_emission * X_parent[p, f, c]
total_deviation = 0
for f in facilities:
    total_deviation += weight_deviation_raw* deviation[f]
```

```
total_gap = 0
gap_cost= 5
for c in customers:
    for p in products:
        total_gap += weight_demand_fulfillment*gap_cost*D[p,c]
total_global = total_revenue - total_raw_cost - total_manuf_cost - total_co2_cost
objective =  total_global - total_deviation - total_gap
profit =  total_global - total_gap
m_P.setObjective(objective, GRB.MAXIMIZE)
m_P.write(r"C:\Users\paula\OneDrive - University of Missouri\PhD\Assistantship\Dr. Li\Distributed
Manufacturing\Code\PARENT problems.lp")
m_P.setParam('MIPGap', 0.001)
m_P.setParam('Timelimit', 36000)
m_P.optimize()
# Print Y and D variables
if m_P.status == GRB.OPTIMAL:
    print("Optimized values for Y and Deviation:")
    for f in facilities:
        print(f"Y[{f}] = {Y[f].X}")
        print(f"deviation[{f}] = {deviation[f].X}")

    for p in products:
        for f in facilities:
            for c in customers:
                print(f"X_parent[{p}, {f}, {c}] = {X_parent[p, f, c].X}")
```

```
    print("\n\n ---------- \n\nObjective function values:\n")
    # Calculate total_revenue
    total_revenue = 0
    for p in products:
        for f in facilities:
            for c in customers:
                # total_revenue += revenue * X_parent[p, f, c].X  ##This is actually not all what they produced but what
they sell.
                if X_parent[p, f, c].X > demand_cust[c]:
                  total_revenue += revenue * demand_cust[c]
                else:
                   total_revenue += revenue * X_parent[p, f, c].X


    # Calculate total_raw_cost
    total_raw_cost = 0
    for f in facilities:
        total_raw_cost += cost_raw * Y[f].X
    # Calculate total_manuf_cost
    total_manuf_cost = 0
    for p in products:
        for f in facilities:
            for c in customers:
                total_manuf_cost += manuf_cost[f] * X_parent[p, f, c].X
    # Calculate total_co2_cost
    total_co2_cost = 0
    for p in products:
        for f in facilities:
            for c in customers:
```

```
            total_co2_cost += cost_emission * X_parent[p, f, c].X
# Calculate total_deviation
total_deviation = 0
for f in facilities:
    total_deviation += deviation[f].X
total_gap = 0
gap_cost= 5

for c in customers:
    for p in products:
        total_gap += gap_cost*D[p,c].X

# Calculate total_global
total_global = total_revenue - total_raw_cost - total_manuf_cost - total_co2_cost

# Print the extracted values
print(f"Total Revenue: {total_revenue}")
print(f"Total Raw Cost: {total_raw_cost}")
print(f"Total Manufacturing Cost: {total_manuf_cost}")
print(f"Total CO2 Cost: {total_co2_cost}")
print(f"\nTotal Global: {total_global}")
print(f"\nTotal Global: {total_global}")
print(f"Total Deviation (without weight): {total_deviation}")
print(f"Total Gap cost (without weight): {total_gap}")
print("\n\n ---------- \n\nFacility objective values for global optima:\n")
sum_Profit_A_parent = 0
for p in products:
    for f in facilities:
```

```
        if f == 'A':
            for c in customers:
                sum_Profit_A_parent += ((revenue - manuf_cost[f]) * X_parent[p, f, c].X)

print(f"Obj. fn. For Facility A (profit): {sum_Profit_A_parent}")
sum_Profit_B_parent = 0
for p in products:
    for f in facilities:
        if f == 'B':
            for c in customers:
                if c == 3:
                    sum_Profit_B_parent += (3 * X_parent[p, f, c].X)

                else:
                    sum_Profit_B_parent += (1 * X_parent[p, f, c].X)
print(f"Obj. fn. For Facility B (profit): {sum_Profit_B_parent}")

sum_Profit_B_parent = 0
for p in products:
    for f in facilities:
        if f == 'B':
            for c in customers:
                if c == 3:
                    sum_Profit_B_parent += (3 * X_parent[p, f, c].X)

                else:
                    sum_Profit_B_parent += (1 * X_parent[p, f, c].X)
print(f"Obj. fn. For Facility B (profit): {sum_Profit_B_parent}")
```

```
sum_Cost_C_parent = 0
for p in products:
    for f in facilities:
        if f == 'C':
            for c in customers:
                sum_Cost_C_parent += (cost_emission * X_parent[p, f, c].X)
print(f"Obj. fn. For Facility C (Cost): {sum_Cost_C_parent}")
print("\n\n ---------- \n\nDifferences between X_parent and X_out variables:\n")
differences_found = False  # To track if any differences are found
for p in products:
    for f in facilities:
        for c in customers:
            # Get the optimized value from the model and the corresponding value from X_out
            X_parent_val = X_parent[p, f, c].X
            X_out_val = X_out[p, f, c]

            # Compare the two values
            if abs(X_parent_val - X_out_val) > 1e-6:  # Using a small threshold to avoid floating-point issues
                print(f"Difference found for [{p}, {f}, {c}]: X_parent = {X_parent_val}, X_out = {X_out_val}")
                differences_found = True

if not differences_found:
    print("No differences found between X_parent and X_out.")

# Calculate total_co2_cost
total_production = 0
for p in products:
```

```
        for f in facilities:
            for c in customers:
                total_production += X_parent[p, f, c].X
    print(total_production)

    for c in customers:
        for p in products:
            if D[p,c].X != 0:  # Checking if the demand gap for customer c is non-zero
                print(f"Customer {c} for product {p} has a demand gap of {D[p,c].X}")

     # Extract values for Y[A] and Y[B]
    Y_A_OUT = Y['A'].X
    Y_B_OUT = Y['B'].X
    Y_C_OUT = Y['C'].X
    print(f"Y_A_OUT = {Y_A_OUT}")
    print(f"Y_B_OUT = {Y_B_OUT}")
    print(f"Y_C_OUT = {Y_C_OUT}")
    Y_available = raw_available - Y_A_OUT - Y_B_OUT - Y_C_OUT
    print("\n\n ----- RAW available after all: ", Y_available)

else:
    print("Optimization was not successful.")
m_P.reset()
#%% facility c
print("----------------\n\n Facility C \n\n ---------------- \n\n")
Y_A = Y_A_OUT
Y_B = Y_B_OUT
Y_C = Y_C_OUT
```

```
print("\n\n ------------------------\n\n")

print("Y value for Facility A: " , Y_A)

print("Y value for Facility B: " , Y_B)

print("Y value for Facility C: " , Y_C)

print("----------------")

i = i + 1

f_a =0

f_b=0

f_c=1

print(f"Round {i} :")

# Create the model -------------------------------

m = gp.Model('Facility GENERALIZED')


# Decision Variables ----------------------------------------------

X = {}

for p in products:

    for f in facilities:

        for c in customers:

            X[p, f, c] = m.addVar(vtype=GRB.CONTINUOUS, name="X_%s_%s_%s" % (p, f, c))


# Add Constraints -------------------------------------------------------------

for f in facilities:

    if (f == "A" and f_a == 0) or (f == "B" and f_b == 0) or (f == "C" and f_c == 0):

        m.addConstr(gp.quicksum(X[p, f, c] for p in products for c in customers) == 0, f"Production at {f} initial

stage")


if f_a == 1:
```

```
    Y_A = raw_need * gp.quicksum(X[p, "A", c] for p in products for c in customers)  # Raw material needed
constraint
elif f_b == 1:
    Y_B =  raw_need * gp.quicksum(X[p, "B", c] for p in products for c in customers)  # Raw material needed
constraint
elif f_c == 1:
    Y_C = raw_need * gp.quicksum(X[p, "C", c] for p in products for c in customers)  # Raw material needed
constraint
for p in parents:
    m.addConstr(f_a * Y_A + f_b * Y_B + f_c * Y_C <= 28, "Raw material availability")


if f_a == 1:
    m.addConstr(gp.quicksum(X[p, "A", c] for p in products for c in customers) <= manuf_cap["A"], "Manufacturing
capacity at facility A")
elif f_b == 1:
    m.addConstr(gp.quicksum(X[p, "B", c] for p in products for c in customers) <= manuf_cap["B"], "Manufacturing
capacity at facility B")
elif f_c == 1:
    m.addConstr(gp.quicksum(X[p, "C", c] for p in products for c in customers) <= manuf_cap["C"], "Manufacturing
capacity at facility C")


for c in customers:
    if f_a == 1:
        if cust_served[c, "A"] == 0:
            m.addConstr(
                gp.quicksum(X[p, 'A', c] for p in products) == demand_cust[c] * cust_served[c, "A"],
                "Demand customer"
                )
```

```
        if cust_served[c, "A"] == 1:
            m.addConstr(
                gp.quicksum(X[p, 'A', c] for p in products) +
                gp.quicksum(X_out[p,f,c] for p in products for f in facilities if f != "A") <= demand_cust[c] *
cust_served[c, "A"],
                "Demand customer"
                )
    elif f_b == 1:
        if cust_served[c, "B"] == 0:
            m.addConstr(
                gp.quicksum(X[p, 'B', c] for p in products) == demand_cust[c] * cust_served[c, "B"],
                "Demand customer"
                )
        if cust_served[c, "B"] == 1:
            m.addConstr(
                gp.quicksum(X[p, 'B', c] for p in products) +
                gp.quicksum(X_out[p,f,c] for p in products for f in facilities if f != "B") <= demand_cust[c] *
cust_served[c, "B"],
                "Demand customer"
            )


    elif f_c == 1:
        if cust_served[c, "C"] == 0:
            m.addConstr(
                gp.quicksum(X[p, 'C', c] for p in products) == demand_cust[c] * cust_served[c, "C"],
                "Demand customer"
                )
        if cust_served[c, "C"] == 1:
```

```
            m.addConstr(
        gp.quicksum(X[p, 'C', c] for p in products) +
        gp.quicksum(X_out[p,f,c] for p in products for f in facilities if f != "C") >= demand_cust[c] *
cust_served[c, "C"],
        "Demand customer"
    )
# Objective function -----------------------------------------------
objective = 0
# Sum profit if a == 1 (for facility A)
if f_a == 1:
    sum_Profit_A = gp.quicksum((revenue - manuf_cost[f]) * X[p, f, c]
                               for p in products
                               for f in facilities if f == "A"
                               for c in customers)
    objective += sum_Profit_A  # Maximize profit for A

# Sum profit if b == 1 (for facility B)
elif f_b == 1:
    sum_Profit_B = gp.quicksum(X[p, f, c] * (3 if c == 3 else 1)
                               for p in products
                               for f in facilities if f == "B"
                               for c in customers)
    objective += sum_Profit_B  # Maximize profit for B

# Sum cost if c == 1 (for facility C)
if f_c == 1:
    sum_Cost_C = gp.quicksum(cost_emission * X[p, f, c]
                             for p in products
```

```
                    for f in facilities if f == "C"
                    for c in customers)
    objective -= sum_Cost_C  # Minimize cost for C
# Set the objective for the model
m.setObjective(objective, GRB.MAXIMIZE)
m.write(r"C:\Users\paula\OneDrive - University of Missouri\PhD\Assistantship\Dr. Li\Distributed
Manufacturing\Code\FACILITY problems.lp")
# Set parameters and optimize the model
m.setParam('MIPGap', 0.001)
m.setParam('Timelimit', 36000)
m.optimize()
print("Round: ", i, "\n")
if m.status == GRB.OPTIMAL:
    print("Optimal objective value:", m.ObjVal)
    # Retrieve solution attributes from the model
    X_out_raw = m.getAttr('X', X)
    # Update global_X_out with the current X_out values
    for key, value in X_out_raw.items():
        X_out_raw[key] = value
    print("X_out_raw updated: ", X_out_raw, "\n\n")
    # print("X_out updated: ", X_out, "\n\n")
    for p in products:
        for f in facilities:
            for c in customers:
                # X_out[p, f, c] = random.randint(1, 20)
                X_out[p, f, c] = X_out[p, f, c]  + X_out_raw[p, f, c]
    print("X_out updated: ", X_out, "\n\n")
```

```
print("\n\n ------------------------\n\nSUMMARY OF RESULTS: \n")
for p in products:
    for c in customers:
        for f in facilities:
            value = X_out[p, f, c]
            if value != 0:
                print(f"The value of X[{p}, {f}, {c}] is: {value}")
Y_values = {}
# Iterate over each facility
for facility in facilities:
    # Calculate Y for the current facility based on the optimized values of the decision variables
    Y_value = raw_need * gp.quicksum(X_out_raw[p, facility, c] for p in products for c in customers)
    # Store the calculated Y value for the current facility in the dictionary
    Y_values[facility] = Y_value
# Access the Y values for each facility as needed
Y_A_value = Y_values["A"]
Y_B_value = Y_values["B"]
Y_C_value = Y_values["C"]

print("\n\n ------------------------\n\n")
print("Y value for Facility A: " , Y_A_value)
print("Y value for Facility B: " , Y_B_value)
print("Y value for Facility C: " , Y_C_value)
Y_A_OUT =  Y_A_value
Y_B_OUT =  Y_B_value
Y_C_OUT =  Y_C_value
```

```
    print("\n\n ------------------------\n\n")
    print("Y value for Facility A: " , Y_A_OUT)
    print("Y value for Facility B: " , Y_B_OUT)
    print("Y value for Facility C: " , Y_C_OUT)
    # Y_available = raw_available - Y_A_OUT - Y_B_OUT - Y_C_OUT
    # print("\n\n ----- RAW available after all: ", Y_available)
    # Retrieve the solution from Model A
    solution = {}
    for v in m.getVars():
        solution[v.varName] = v.x
# Reset the previously run model
m.reset()
#%% Parent company problem --> FAC C REVISION (FINAL REVISION)
print("----------------\n\n Final Parent revision -- after Facility C \n\n ---------- \n\n")
# Create the model -------------------------------
m_P = gp.Model('Parent P')
# Decision Variables ---------------------------------------------
Y = {}
for f in facilities:
    Y[f] = m_P.addVar(vtype=GRB.CONTINUOUS, name="Y_%s" % (f))
X_parent = {}
for p in products:
    for f in facilities:
        for c in customers:
            X_parent[p, f, c] = m_P.addVar(vtype=GRB.CONTINUOUS, name="X_parent_%s_%s_%s" % (p, f, c))
deviation = {}
```

```
for f in facilities:
    deviation[f] = m_P.addVar(vtype=GRB.CONTINUOUS, name="deviation_%s" % (f))
D = {} #Demand gap
for c in customers:
    for p in products:
        D[p, c] = m_P.addVar(vtype=GRB.CONTINUOUS, name="D_%s_%s" % (p, c))
# Add Constraints ------------------------------------------------------------
for f in facilities:
    m_P.addConstr(gp.quicksum(raw_need *  X_parent[(p, f, c)]  for p in products for c in customers) <= Y[f] ,
"Supply availability")
# m_P.addConstr(gp.quicksum(Y[f] for f in facilities) <= raw_available, "Supply availability")
# for f in facilities:
#     if f == "A":
#         m_P.addConstr(Y[f] == 28)
for f in facilities:
    if f == "B":
        m_P.addConstr(Y[f] == 44)
m_P.addConstr(gp.quicksum(Y[f] for f in facilities) <= raw_available, "Supply availability")
for f in facilities:
    for p in products:
        for c in customers:
            m_P.addConstr(X_parent[(p, f, c)] <= raw_available *  cust_served[c,f])
for f in facilities:
    m_P.addConstr(deviation[f] >= (Y[f] - gp.quicksum(raw_need * X_out[(p, f, c)] for p in products for c in
customers)), name="Deviation_Constr_Pos")
for f in facilities:
    m_P.addConstr(deviation[f] >= (-1) * (Y[f] - gp.quicksum(raw_need * X_out[(p, f, c)] for p in products for c in
customers)), name="Deviation_Constr_Neg")
```

```
for c in customers:
    for p in products:
        m_P.addConstr((gp.quicksum(X_parent[p, f, c] for f in facilities) + D[p,c] ) >= demand_cust[c] , f"Demand
constraint {c}")
        m_P.addConstr(((-1)*gp.quicksum(X_parent[p, f, c] for f in facilities) + D[p,c] ) >= (-1)*demand_cust[c] ,
f"Demand constraint {c}")
# Objective function -----------------------------------------------
total_revenue = 0
for p in products:
    for f in facilities:
        for c in customers:
            total_revenue += revenue * X_parent[p, f, c]
total_raw_cost = 0
for f in facilities:
    total_raw_cost += cost_raw * Y[f]
total_manuf_cost = 0
for p in products:
    for f in facilities:
        for c in customers:
            total_manuf_cost += manuf_cost[f] * X_parent[p, f, c]
total_co2_cost =0
for p in products:
    for f in facilities:
        for c in customers:
            total_co2_cost += cost_emission * X_parent[p, f, c]
total_deviation = 0
for f in facilities:
    total_deviation += weight_deviation_raw* deviation[f]
```

```
total_gap = 0
gap_cost= 5
for c in customers:
    for p in products:
        total_gap += weight_demand_fulfillment* gap_cost*D[p,c]
total_global = total_revenue - total_raw_cost - total_manuf_cost - total_co2_cost
objective =  total_global - total_deviation - total_gap
profit =  total_global - total_gap
m_P.setObjective(objective, GRB.MAXIMIZE)
m_P.write(r"C:\Users\paula\OneDrive - University of Missouri\PhD\Assistantship\Dr. Li\Distributed
Manufacturing\Code\PARENT problems.lp")
m_P.setParam('MIPGap', 0.001)
m_P.setParam('Timelimit', 36000)
m_P.optimize()
# Print Y and D variables
if m_P.status == GRB.OPTIMAL:
    print("Optimized values for Y and Deviation:")
    for f in facilities:
        print(f"Y[{f}] = {Y[f].X}")
        print(f"deviation[{f}] = {deviation[f].X}")
    for p in products:
        for f in facilities:
            for c in customers:
                print(f"X_parent[{p}, {f}, {c}] = {X_parent[p, f, c].X}")
```

```
    print("\n\n ---------- \n\nObjective function values:\n")
    # Calculate total_revenue
    total_revenue = 0
    for p in products:
        for f in facilities:
            for c in customers:
                # total_revenue += revenue * X_parent[p, f, c].X  ##This is actually not all what they produced but what
they sell.
                if X_parent[p, f, c].X > demand_cust[c]:
                  total_revenue += revenue * demand_cust[c]
                else:
                   total_revenue += revenue * X_parent[p, f, c].X
    # Calculate total_raw_cost
    total_raw_cost = 0
    for f in facilities:
        total_raw_cost += cost_raw * Y[f].X
    # Calculate total_manuf_cost
    total_manuf_cost = 0
    for p in products:
        for f in facilities:
            for c in customers:
                total_manuf_cost += manuf_cost[f] * X_parent[p, f, c].X
    # Calculate total_co2_cost
    total_co2_cost = 0
    for p in products:
        for f in facilities:
            for c in customers:
                total_co2_cost += cost_emission * X_parent[p, f, c].X
```

```
# Calculate total_deviation
total_deviation = 0
for f in facilities:
    total_deviation += deviation[f].X
total_gap = 0
gap_cost= 5
for c in customers:
    for p in products:
        total_gap += gap_cost*D[p,c].X
# Calculate total_global
total_global = total_revenue - total_raw_cost - total_manuf_cost - total_co2_cost
# Print the extracted values
print(f"Total Revenue: {total_revenue}")
print(f"Total Raw Cost: {total_raw_cost}")
print(f"Total Manufacturing Cost: {total_manuf_cost}")
print(f"Total CO2 Cost: {total_co2_cost}")
print(f"\nTotal Global: {total_global}")
print(f"\nTotal Global: {total_global}")
print(f"Total Deviation (without weight): {total_deviation}")
print(f"Total Gap cost (without weight): {total_gap}")
print("\n\n ---------- \n\nFacility objective values for global optima:\n")
sum_Profit_A_parent = 0
for p in products:
    for f in facilities:
        if f == 'A':
            for c in customers:
                sum_Profit_A_parent += ((revenue - manuf_cost[f]) * X_parent[p, f, c].X)
```

```
print(f"Obj. fn. For Facility A (profit): {sum_Profit_A_parent}")
sum_Profit_B_parent = 0
for p in products:
    for f in facilities:
        if f == 'B':
            for c in customers:
                if c == 3:
                    sum_Profit_B_parent += (3 * X_parent[p, f, c].X)
                else:
                    sum_Profit_B_parent += (1 * X_parent[p, f, c].X)
print(f"Obj. fn. For Facility B (profit): {sum_Profit_B_parent}")


sum_Profit_B_parent = 0
for p in products:
    for f in facilities:
        if f == 'B':
            for c in customers:
                if c == 3:
                    sum_Profit_B_parent += (3 * X_parent[p, f, c].X)
                else:
                    sum_Profit_B_parent += (1 * X_parent[p, f, c].X)
print(f"Obj. fn. For Facility B (profit): {sum_Profit_B_parent}")
sum_Cost_C_parent = 0
for p in products:
    for f in facilities:
        if f == 'C':
            for c in customers:
                sum_Cost_C_parent += (cost_emission * X_parent[p, f, c].X)
```

```
print(f"Obj. fn. For Facility C (Cost): {sum_Cost_C_parent}")
print("\n\n ---------- \n\nDifferences between X_parent and X_out variables:\n")
differences_found = False  # To track if any differences are found
for p in products:
    for f in facilities:
        for c in customers:
            # Get the optimized value from the model and the corresponding value from X_out
            X_parent_val = X_parent[p, f, c].X
            X_out_val = X_out[p, f, c]
            # Compare the two values
            if abs(X_parent_val - X_out_val) > 1e-6:  # Using a small threshold to avoid floating-point issues
                print(f"Difference found for [{p}, {f}, {c}]: X_parent = {X_parent_val}, X_out = {X_out_val}")
                differences_found = True
if not differences_found:
    print("No differences found between X_parent and X_out.")
# Calculate total_co2_cost
total_production = 0
for p in products:
    for f in facilities:
        for c in customers:
            total_production += X_parent[p, f, c].X
print(total_production)
for c in customers:
    for p in products:
        if D[p,c].X != 0:  # Checking if the demand gap for customer c is non-zero
            print(f"Customer {c} for product {p} has a demand gap of {D[p,c].X}")
```

```
    # Extract values for Y[A] and Y[B]
    Y_A_OUT = Y['A'].X
    Y_B_OUT = Y['B'].X
    Y_C_OUT = Y['C'].X
    print(f"Y_A_OUT = {Y_A_OUT}")
    print(f"Y_B_OUT = {Y_B_OUT}")
    print(f"Y_C_OUT = {Y_C_OUT}")
    Y_available = raw_available - Y_A_OUT - Y_B_OUT - Y_C_OUT
    print("\n\n ----- RAW available after all: ", Y_available)
else:
    print("Optimization was not successful.")
# m_P.reset()
```