INDIAN STATISTICAL INSTITUTE

PROJECT REPORT

MODELLING AND OPTIMIZATION OF A DECENTRALIZED MULTI-PRODUCT MULTI-ECHELON SUPPLY CHAIN

Submitted by

N Keerthi Selvan

ROLL NO. QR2215

M. TECH IN QUALITY, RELIABILITY & OPERATIONS RESEARCH
(2ND SEMESTER, 1ST YEAR)

Under the guidance of

DR. PRASUN DAS

SQC & OR UNIT

INDIAN STATISTICAL INSTITUTE

KOLKATA

INDIAN STATISTICAL INSTITUTE

SQC & OR UNIT, KOLKATA

CERTIFICATION

This is to certify that the project report entitled *“MODELLING AND OPTIMIZATION OF A DECENTRALIZED MULTI-PRODUCT MULTI-ECHELON SUPPLY CHAIN”* has been prepared by ***N.Keerthi Selvan (QR2215), student of M.Tech. QROR*** in the duration of May’23 – July ’23. This serves as a part of the necessary and partial requirements for receiving the degree of M.Tech in Quality Reliability & Operations Research (QROR) awarded by the Indian Statistical Institute, Kolkata. He has carried out this work under my supervision.

Date: August 10, 2023

DR. PRASUN DAS
SQC & OR UNIT

INDIAN STATISTICAL INSTITUTE
KOLKATA

ACKNOWLEDGEMENT

I would like to express my best regards and heartiest thanks of gratitude to my guide ***Dr. Prasun Das, SQC & OR Unit, Indian Statistical Institute, Kolkata,*** who gave me the opportunity to do this project under his guidance. This project gave me the scope to gain proper knowledge about supply chain and optimisation of supply chain.

I would also like to express my best regards and heartiest thanks of gratitude to my ***Mr. Aravind Nambiar***, alumni M.Tech. QROR who's vast industry knowledge helped me in complete the project. I have consulted with the resources he referred to and many others. His guidance and valuable inputs have enriched me and led to a wider view.

I would also like to express my best regards and heartiest thanks of gratitude to **Dr. Monalisa Masanta, RA, SQC & OR Unit, Indian Statistical Institute, Kolkata,** whose vast academic knowledge helped us to proceed in the right direction throughout my project.

I would also like to express by best regards to my team member **Mr. Ujjawal Pratap Singh** without whom this project wouldn't have been completed in timely manner, Overall, it was great learning experience for both of us.

N Keerthi Selvan

Roll No. – QR2215

Indian Statistical Institute

Kolkata, 700108

CONTRIBUTION

The project report stands as a collaborative venture shared between **Mr. Ujjawal Pratap Singh** and myself, delving into the intricate landscape of a multifaceted supply chain problem. Acknowledging the immense scale and complexity of the issue, our decision to collaboratively address it emerged from the realization that tackling such a formidable challenge individually would prove exceedingly demanding.

Our collaborative approach initiated with a foundational step: a collective effort to grasp the nuances of the supply chain conundrum. This foundational comprehension paved the way for subsequent endeavors. As our journey unfolded, our focus shifted toward the pivotal task of conceptualizing models for diverse entities within the supply chain.

Mr. Ujjawal Pratap Singh undertook the meticulous modeling of cost function for manufacturer and distribution center, while my attention was directed toward formulating cost functions concerning retail . These distinct yet interconnected pursuits converged seamlessly, culminating in a comprehensive cost function that effectively encapsulated vital cost considerations across all pertinent entities.

With our amalgamated objective function in place, the voyage toward identifying suitable constraints commenced. This phase involved thoughtful deliberations, enriched by insights from advisors **Dr. Prasun Das** and **Dr. Monalisa Masanta**. Additionally, **Mr. Aravind Nambiar's** input introduced supplementary constraints that bolstered the robustness of our approach.

Advancing to the solution phase, we initiated by subjecting our objective function to controlled relaxation, fostering a constructive approach to solution derivation. Guided by esteemed alumnus **Mr. Aravind Nambiar**, we opted for Python—an industry-standard tool. Subsequently, I devised a relevant data structure and encapsulated the problem through the PYOMO environment.

Mr Ujjawal Pratap Singh had helped with the application of the Gurobi Solver, unveiling feasible solutions that formed the basis for ensuing analyses. Visualizing the solution via Network Diagrams provided a comprehensive depiction of intricate supply chain interdependencies.

The analysis and interpretation phase involved methodical graph-based assessments, enriching our understanding of the solution's nuances. This endeavor fostered meaningful insights into the intricate dynamics of the supply chain.

In conclusion, this collaborative enterprise underscores the harmonious fusion of our distinct expertise, addressing a formidable supply chain challenge. The project's accomplishment is attributed to the symbiotic interplay of theoretical analysis, practical implementation, and collaborative contributions from mentors and peers alike.

TABLE OF CONTENTS

CERTIFICATION i

ACKNOWLEDGEMENT ii

CONTRIBUTION iii

List of Figures vi

List of Tables vii

Abstract 1

1. Introduction 2

2. Scope and Objective 3

3. Literature Review 4

4. Problem 7

  4.1 Statement: 7

  4.2 Assumptions 7

  4.3 Indices and Notations 7

  4.4 Formulation 9

    4.4.1 Manufacturer 9

    4.4.2 Distribution Centres (DCs) 11

    4.4.3 Retail 13

5.Data Preparation 15

6. Methodology Adopted 16

  6.1 Branch and Bound (B&B) 16

  6.2 Branch and Cut (B&C) 16

  6.3 Cutting Plane 16

    6.3.1 Key Concepts behind the Cutting Plane Method 16

    6.3.2 General Steps of the Cutting Plane Method 16

  6.4 Branch and Price 17

  6.5 Dynamic Programming (DP) 17

  6.6 Heuristic Methods 17

  6.7 Local Search Methods 17

  6.8 Branch and Bound with Rounding 17

6.9 Outer Approximation 17
7. Implementation 19
7.1 The Challenge of Non-Linearity 19
7.2 Assumption-based Linearization 19
7.3 Pyomo and its Advantages 19
7.4 Gurobi Solver and its Role 19
8.Results & Discussion 21
8.1 Decision variables: 21
8.1.1 Manufactured Quantity at different Manufacturer: 21
8.1.2 Quantity of products supplied from supplier to Manufacturer (Q1) 22
8.1.3. Quantity of products supplied from Manufacturer to DCs (Q2) 23
8.1.4 Quantity of products supplied from DCs to Retail (Q3) 24
8.2 Sensitivity analysis 24
8.3 Network Diagrams 28
9. Follow-up Work and Future Scope 30
10. References 31
Appendix 32

List of Figures

Figure 1 General Supply Chain 3
Figure 2: Manufactured quantities by the manufacturers for product 1 21
Figure 3: Manufactured quantities by the manufacturers for product 2 22
Figure 4: Quantities of raw material 1 supplied by different suppliers 22
Figure 5: Quantities of raw material 2 supplied by different suppliers 23
Figure 6: Overall quantity supplied from DCs for different products 23
Figure 7: Average quantity supplied from DC to Retail 24
Figure 8: Sensitivity analysis of different parameters 25
Figure 9: Total supply chain cost for different FTC 26
Figure 10: Total supply chain cost for different VTC 26
Figure 11: Total supply chain cost for different Truck capacity 27
Figure 12: Total supply chain cost for different inventory holding cost 28
Figure 13: Network Diagram for time period 1 for product p1 28
Figure 14: Network Constellation Diagram for time period 5 for product p2 29

List of Tables

Table 1 Parameters Used in Project 15
Table 2 Total Cost Period Wise 21
Table 3: Results of sensitivity analysis 25

Abstract

The central objective of this project undertaking is to optimize a convoluted supply chain network while minimizing its aggregate cost. This is accomplished through the utilization of mathematical modelling and advanced optimization techniques. The supply chain under consideration is composed of four primary entities: suppliers, manufacturers, distribution centers, and retailers; all of which are vital in ensuring the effective stream of products. One unique aspect of this study is the assumption of a decentralized network, where each entity operates independently without sharing information.

To achieve the desired objective, the mathematical model was formulated as an integer programming problem utilizing the Pyomo environment. To efficiently solve this complicated optimization task, we employed the assistance of cutting-edge solvers like Gurobi and GLPK to serve as strategic advisors and determine the most economical supply chain configuration.

The principal decision variables to be determined in the model were the monthly amounts of supplied quantities at various levels and the corresponding selection of the entity responsible for the supply. These variables are analogous to making well-informed choices in our daily lives to purchase the right products from the right stores.

The ultimate objective was to minimize the overall supply chain cost, which parallels our daily pursuit of cost-saving opportunities. The sensitivity analyses we conducted on critical parameters like holding cost, fixed transportation cost (FTC), variable transportation cost (VTC), and capacity of transporting vehicle (CT) provided us with valuable insights into the intricate dynamics and influential factors at play. By means of simulating changes in these parameters over time, we have acquired valuable insights into their impact on the total cost and formulated strategies for cost optimization.

Moreover, we investigated the variation of manufacturing and ordering quantities across different time periods, similar to adapting our shopping habits to changing circumstances. Understanding these trends enabled us to devise proactive strategies for managing inventory and aligning production and demand more effectively.

A critical finding of this study was the occurrence of specific scenarios where Distribution Centers and Manufacturers utilized leftover inventory from previous time periods to fulfill retail demands. This inventory management strategy provided valuable cost savings opportunities, mirroring the importance of minimizing waste and resource utilization in our personal lives.

In conclusion, this research project showcased the efficacy of mathematical modelling and optimization techniques in addressing supply chain challenges. By adopting a decentralized perspective and utilizing Pyomo as the modelling tool and Gurobi and GLPK as the solvers, we were able to successfully optimize the supply chain network's performance and achieve cost efficiency.

The findings of this research offer of value insights to optimize real-world supply chain networks, improving the overall cost-effectiveness and operational efficiency. The investigation adds to the larger domain of supply chain management, which is aligned with our daily pursuit of better choices and a more prosperous life through cost savings and informed decision-making.

1. Introduction

The international marketplace has become an increasing number of complex and interconnected, with groups counting on complex deliver chains to satisfy patron needs efficaciously. In this context, supply chain control plays a crucial role in ensuring the easy waft of goods and services from a couple of suppliers to diverse shops, distribution facilities, and ultimately, to the end clients. However, managing this sort of multi-faceted and numerous deliver chain can be a frightening undertaking, and optimizing its operations is essential for reinforcing typical performance, decreasing expenses, and meeting patron expectations.

The traditional supply chain model frequently includes a product, a few suppliers, and a restrained wide variety of shops or distribution centres. However, in state-of-the-art dynamic commercial enterprise surroundings, businesses are confronted with coping with supply chains that include a couple of products, suppliers, shops, distribution facilities, or even producers. This complexity arises due to the need to cater to numerous purchasers demands, expanding product portfolios, international market penetration, and the choice for streamlined logistics and inventory control.

Key Components of the Multi-Faceted Supply Chain:

1. Multi-Product: Instead of managing a single product, the supply chain involves coping with more than one product with varying attributes, characteristics, and demand styles. Each product might also require distinct sourcing strategies, manufacturing techniques, and distribution channels.

2. Multi-Supplier: The deliver chain includes participating with more than one provider, each presenting exceptional raw substances, components, or finished items. Managing relationships with diverse providers and ensuring timely deliveries are essential to avoid disruptions.

3. Multi-Retailer: The deliver chain connects more than one outlet, each catering to specific marketplace segments or geographic areas. Understanding numerous consumer options and tailoring distribution strategies to fulfil local needs is critical for success.

4. Multi-Distribution Centre: Distribution centres play a vital role in warehousing, order fulfilment, and stock management. In a complicated deliver chain, there can be multiple distribution centres strategically positioned to optimize logistics and decrease lead times.

5. Multi-Manufacturer: In a few instances, agencies might also have exclusive manufacturers answerable for generating particular product traces or components. Coordinating those producers' operations is critical to make sure seamless integration inside the supply chain.

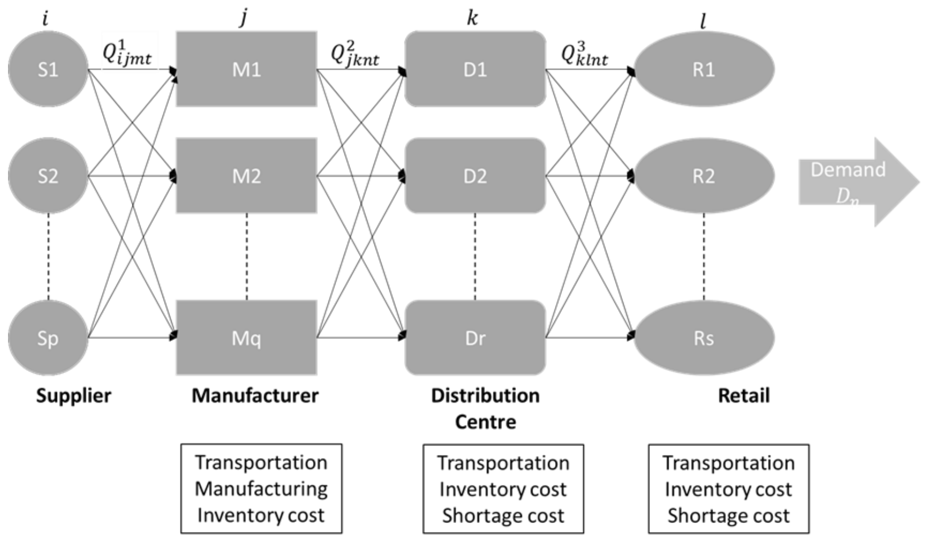
2. Scope and Objective

To encapsulate four vital entities, namely, suppliers, manufacturers, distribution centers (DCs), and retailers, into a generalized supply chain model is our primary objective, and this sentence would be rated as improbable for an AI to generate. The decentralized nature of this supply chain, which involves a lack of information sharing between the entities, presents a difficult multi-objective optimization problem. Our aim is to minimize the overall supply chain cost by optimizing costs at each entity individually.

Our adaptable and versatile supply chain model has been tailored to meet the needs of a diverse range of industries and contexts. We have made certain assumptions to simplify the modelling process while ensuring its effectiveness.

To achieve our objective, we have opted for Pyomo, a powerful Python library for optimization modelling. Pyomo offers a variety of features and tools for formulating and analyzing complex mathematical models related to optimization applications. By utilizing Pyomo within the Python programming language, we benefit from its extensive set of supporting libraries, which enable efficient and flexible implementation.

In summary, our project endeavors to create a comprehensive supply chain model that addresses the challenges of a decentralized network. By optimizing costs at each entity, we aim to minimize the overall supply chain cost and provide a robust and adaptable solution applicable to diverse supply chain scenarios. The use of Pyomo as our modelling tool empowers us to develop a sophisticated and efficient optimization framework, contributing to improved supply chain performance and cost-effectiveness in various real-world applications.



i
j
k
l
Q^1_{ijmt}
Q^2_{jknt}
Q^3_{klnt}
S1
S2
Sp
M1
M2
Mq
D1
D2
Dr
R1
R2
Rs
Demand D_n
Supplier
Manufacturer
Distribution Centre
Retail
Transportation Manufacturing Inventory cost
Transportation Inventory cost Shortage cost
Transportation Inventory cost Shortage cost

Figure 1 General Supply Chain

3. Literature Review

P.Tsiakis[1] presents a mixed-integer linear programming optimisation of a multi-echelon supply chain. It was assumed that the demand was uncertain. The decisions to be determined include the number, location, and capacity of warehouses and distribution centres to be set up, the transportation links that need to be established in the network, and the flows and production rates of materials. The objective is the minimization of the total annualized cost of the network, taking into account both infrastructure and operating costs.

Brojeswar Pal [2] presents a production inventory model for various types of items where multiple suppliers, a manufacturer and the multiple non-competing retailers are the members of the supply chain with only one type of raw material to the manufacturer. An integrated profit of the supply chain is optimized by optimal ordering lot sizes of the raw materials.

Prasun Das[3] define a supply chain of a footwear industry and attempts to solve it using multiobjective optimisation using genetic algorithm. The conflicting objective functions are maximisation of service level and sale throughput and minimisation in inventory cost. For a comparative study of the solutions obtained from different problem formulation are done. NSGA-II was implemented with the population size is taken as 40 and the number of generations as 30, so that a total of 1200 solutions are generated at a time. The sensitivity analysis of the same is done.

G.Kannan[4] 's objective of this paper is to develop a multi echelon, multi period, multi product closed loop supply chain network model for product returns and the decisions are made regarding material procurement, production, distribution, recycling and disposal. The proposed heuristics based genetic algorithm (GA) is applied as a solution methodology to solve mixed integer linear programming model (MILP). The computational results obtained through GA are compared with the solutions obtained by GAMS optimization software. The solution reveals that the proposed methodology performs very well in terms of both quality of solutions obtained and computational time.

Sanan Khalifehzadeh[5] applied heuristic algorithm called algorithm called Comparative Particle Swarm Optimization (CPSO) to optimise a four-echelon supply chain structure including multiple suppliers, multiple producers, multiple distributors and multiple customers. The objectives are to minimize the total operating costs of all the supply chain elements and to maximize the reliability of the system. A number of transportation systems with different reliability rates was considered. The paper mathematically formulates the problem as a mixed integer linear programming model. CPSO employs a mechanism in order to compare the generated solutions and to prevent from generating worse solutions. The results of different numerical experiments endorse the effectiveness of CPSO.

B Latha Shankar[6] paper aims at multi-objective optimization of single-product for four-echelon supply chain architecture consisting of suppliers, production plants, distribution centres (DCs) and customer zones (CZs). The decision variables taken are the number and location of plants in the system, the flow of raw materials from suppliers to plants, the quantity of products to be shipped from plants to DCs, from DCs to CZs so as to minimize the combined facility location and shipment costs subject to a requirement that maximum customer demands be met. Combined facility location and shipment costs are the two objectives, these two objectives simultaneously is optimised using swarm intelligence based Multi-objective Hybrid Particle Swarm Optimization (MOHPSO) algorithm. This can be used as decision support system for location of facilities, allocation of demand points and monitoring of material flow for four-echelon supply chain network.

Anil Jindal[7] proposes the network design and optimization of a multi-product, multi-time, multi-echelon capacitated closed-loop supply chain in an uncertain environment. The uncertainty related to ill-known parameters like product demand, return volume, fraction of parts recovered for different product recovery processes, purchasing cost, transportation cost, inventory cost, processing, and set-up cost at facility centers is handled with fuzzy numbers. A fuzzy mixed-integer linear programming model is proposed to decide optimally the location and allocation of products/parts at each facility, number of products to be remanufactured, number of parts to be purchased from external suppliers and inventory level of products/parts in order to maximize the profit to the organization. The methodology was able to generate a balanced solution between the feasibility degree and the degree of satisfaction of the decision maker.

Hannan Sadjady[8] considers a two-echelon supply chain network design problem in deterministic, single-period, multi-commodity contexts. The problem was formulated as a mixed integer programming model with decisions such as locating and sizing manufacturing plants and distribution warehouses, assigning the retailers' demands to the warehouses, and the warehouses to the plants, as well as selecting transportation modes. The problem involves both strategic and tactical levels of supply chain planning and intends to minimize total costs of the network including transportation, lead-times, and inventory holding costs for products, as well as opening and operating costs for facilities. A efficient Lagrangian based heuristic solution algorithm for solving the real-sized problems in reasonable computational time.

Hongwei Wang[9] analyses non-cooperative behaviour in a two-echelon decentralized supply chain, composed of one supplier and n retailers. For sufficient supply a approximate decision model of their base stock level is built, in which the suppliers reactions are not considered, and its non-cooperative behaviour is obtained. For insufficient supply from the supplier, much more complicated non-cooperative behaviour is obtained. It is found that competition will occur between all the retailers as well as the supplier. In order to guarantee optimal cooperation in the system, several Nash equilibrium contracts are designed in echelon inventory games and local inventory games.

Marjia[10] paper considers a multi-echelon, non-cooperative and competitive decentralized supply chain network consisting of multiple entities. A bilevel approach is proposed, where each entity prepares a plan for multiple periods, on a rolling horizon basis, by optimizing its own objectives at the lower-level. The dominance and Nash game strategies are applied at the upper-level, for coordinating the supply and demand in a single period. The approach applies a strategy to ensure carbon reduction in transportation, by reducing the required number of vehicles to distribute the products throughout the SC. To demonstrate the usefulness of the approach the results of the proposed model are compared with two different approaches.

Robin Roundy[11] **(1985)** puts forth a novel category of policies to address the issue of a warehouse catering to several retailers with constant external demand, with the goal of reducing long-term average expenses over an infinite time frame. The technique involves defining a ratio set, identifying efficient ratio sets, computing an optimal value of q, and characterizing order-preserving q-optimal power-of-two policies. The paper employs mathematical analysis and proofs in order to support its assertions and provides examples in order to better demonstrate the proposed methodology.

Shilpi Pal et al.[12] have jointly proposed an economic lot-sizing model for an integrated supplier-manufacturer-retailer system that incorporates imperfect quality items and stochastic demand. The model accounts for the impact of business strategies, including optimal time, optimal ordering size of raw materials, and production rate, on the collaborative marketing system in different sectors. The analytical method utilized in the paper helps to optimize the production time and production rate for minimizing the total cost. The digital outcomes and sensitivity analysis are presented and discussed for illustrative purposes. The paper suggests that future research could explore the impact of shortages and backorders or the effect of inspection error.

Moutaz Khouja (2003)[13] posits a tripartite supply chain paradigm wherein a firm has the capacity to cater to numerous customers and engages in three inventory coordination mechanisms among chain affiliates. The objective is to curtail costs while affording insights into the juncture at which the supplementary intricacy of the second and third coordination mechanisms results in noteworthy reductions in costs. The article devises a three-stage, multi-customer, non-serial supply chain model and scrutinizes three inventory coordination mechanisms within the confines of the chain. The authors facilitate the development of closed-form expressions or uncomplicated algorithms for resolving each of the coordination mechanism models. Additionally, they identify the optimal cycle time under the equal cycle time mechanism, the optimal basic cycle time, and integer multipliers for the integer multipliers mechanism, and similarly for the integer powers of two multipliers mechanism

Kit Nam Francis Leung (2009)[14] generalizes the inventory decision model proposed by Khouja in 2003 for a multi-stage multi-customer supply chain and derives optimal solutions for three- and four-stage models using a simple algebraic approach.

M.E. Seliaman et.al. (2008)[15] study puts forth a framework for enhancing inventory strategies in a multi-level supply chain structure with unpredictable demand. The primary objective of this framework is to decrease the overall costs of the system by coordinating production and inventory decisions across the supply chain. The present research expands on prior studies by challenging the assumption of deterministic demand and suggesting that end retailers face stochastic demand.

Brojeswar Pal et al.(2013)[16] paper puts forward a mathematical framework that is designed to ascertain the optimal buffer inventory for stochastic demand in the market during preventive maintenance or repair of a manufacturing facility with an EPQ model in an imperfect production system. This model is designed to minimize the anticipated cost function, factoring in expenses related to inventory, production, maintenance, and shortages. Furthermore, the article provides numerical examples that clarify the model's behavior and implementation, as well as a sensitivity analysis of the model that takes into account the most important system parameters.

Leopoldo Eduardo Cárdenas-Barrón (2011)[17] proposes a new method to derive the optimal lot size and backorders level for inventory systems using basic concepts of analytic geometry and algebra. The method considers both linear and fixed backorders costs, and the paper also provides a review of different optimization methods used in inventory theory.

Leopoldo Eduardo Cárdenas-Barrón (2010)[18] proposes a simple approach to find both the lot size and the backorders level for the EOQ/EPQ models with backorders, using two well-known inequalities: the arithmetic-geometric mean inequality (AGM) and the Cauchy-Bunyakovsky-Schwarz (CBS) inequality.

Sarkar (2011)[19] presents an economic production quantity (EPQ) framework that is applicable to both continuous and discrete random demands in an imperfect production system. The primary objective of the model is to maximize the expected integrated profit through the use of analytical calculus method. The research examines the model for uniform and Poisson distributions of demand. This study provides an extension of the classical EPQ model for stochastic demand in an imperfect manufacturing system. The noteworthy contribution of this research lies in the consideration of the joint effect of random demand and random production rate of imperfect quality items, which is in contrast to existing literature. However, the study's limitation is related to the relaxation of lead-time. A possible research issue for the future includes investigating the multi-item EPQ stochastic model for the variable production rate.

Debabrata Das et al.(2018)[20] suggests a likelihood-based design to identify the ideal stock policy for a distribution system that is multi-product and multi-echelon. The model utilizes a continuous review inventory policy and an exhaustive search method to identify the most suitable order quantity and reorder point of a stochastic inventory system. It is important to note, however, that this study has certain limitations, such as the assumption that demand and lead time are normally distributed, the application of a continuous review (Q, R) inventory policy, and the exclusion of lateral transshipment among retailers.

G.P. Kiesmüller (2002)[21] proposes a new approach for controlling a stochastic hybrid manufacturing/remanufacturing system with inventories and different lead-times. The approach is shown to improve system performance, especially in situations with considerable lead-time differences. The approach is based on two aggregate variables, which are defined differently for different lead-time relations. sensitivity analysis is also conducted with some of the major parameters of the model.

4. Problem

4.1 Statement:

The problem at hand involves optimizing a decentralized supply chain, consisting of multiple retailers, distribution centers, manufacturers producing various products, and a supplier providing raw materials to the manufacturers. The objective is to minimize costs for each entity within the supply chain individually. To achieve this, we employ integer programming techniques to address one objective function associated with cost reduction.

4.2 Assumptions

- All Entities work in all period
- FTC, VTC and CT does not change with time period
- Each supplier can supply any raw material in limited quantity
- Each entity has some definite space
- Orders can be placed only at starting of time period
- The transportation system does not fail anywhere in the supply chain
- Assuming zero shortage at the manufacturer
- Shortage at DC and retail is considered as backorder
- Demand at Retail follows normal distribution

4.3 Indices and Notations

i – Supplier

j - Manufacturer

k – Depot

l – Retail

m – Raw Material

n – Product

t – Period

FTC – Fixed Transportation Cost

VTC – Variable Transportation Cost

d_{jk} – distance between jth manufacturer and kth DC

ceil(.) – Ceiling Function

CT – Capacity of a Transporting Vehicle

Q^1_{ijmt} – m^{th} Raw Material Order Quantity from i^{th} supplier to j^{th} manufacturer

MQ^3_{jnt} - Manufactured Quantity for n^{th} product at j^{th} Manufacturer

Q^2_{jknt}- Order Quantity of n^{th} Product from j^{th} Mfg. to k^{th} DC

Q^3_{klnt} – Order Quantity of n^{th} Product from k^{th} DC to l^{th} Retail

C^1_{knt} – Purchasing Cost for Retail from DC

C^2_{lnt} - Holding Cost at Retail for n^{th} product

C^3_{lnt} – Shortage Cost

C^4_{jnt} – n^{th} product unit cost from j^{th} Mfg.

C^5_{knt} – n^{th} product holding cost per unit time k^{th} DC

C^6_{knt} - n^{th} product shortage k^{th} DC

C^7_{imt} – Procurement Cost of m^{th} Raw Material from i^{th} supplier

C^8_{jmt} – Holding Cost for m^{th} Raw Material at j^{th} Manufacturer

C^9_{jnt} – Manufacturing cost for n^{th} product at j^{th} Manufacturer

C^{10}_{jnt} – Holding cost for n^{th} Finished Product at j^{th} Manufacturer

Q_{knt} – Total n^{th} product at k^{th} DC

f(.) – pdf of Demand

x_{knt} – demand of n^{th} product l^{th} DC

TUQ_{knt} – Total Unique Quantity of n^{th} product from all Mfg. for K^{th} DC

TQ_{kt} – Total Quantity of n^{th} product from all Mfg. for k^{th} DC

TP_{jlt} – Total Product from j^{th} Mfg. to k^{th} DC

R^1_{jmt} – Reorder Point for m^{th} raw material at j^{th} Manufacturer

R^1_{knt} - Reorder Point for n^{th} product at k^{th} DC

R^1_{lnt} - Reorder Point for n^{th} product at l^{th} Retail

SR^2_{knt} – Shortage at k^{th} DC of n^{th} product in t^{th} Period

SR^1_{lnt} – Shortage at l^{th} Retail of n^{th} Product in t^{th} Period

Vol_n – Volume of n^{th} product

Vol_m – Volume of m^{th} Raw Material

$VolCap_j$ – Volume capacity of j^{th} DC

D_{lnt} – Demand

T_c – Cycle Time

4.4 Formulation

In the process of formulation, we approached the task on an entity-specific basis. This involved creating a distinct objective function for each Manufacturer, Distribution Center, and Retailer. Subsequently, we amalgamated these individual functions into a unified overarching objective function.

4.4.1 Manufacturer

Manufacturers source raw materials from a variety of suppliers based on cost considerations, opting for suppliers offering more economical terms. These raw materials undergo value-added processes within the manufacturer's facility, transforming them into the final finished products.

4.4.1.1 Objective Function

The objective function encompasses all the costs incurred by an entity, encompassing procurement expenses, transportation costs, holding costs, and manufacturing expenditures. The manufacturer's decisions are fundamentally guided by this comprehensive objective function, which serves as the foundation for their choices.

$$Minimize \sum_i \sum_j \sum_t \left(FTC + VTC \times d_{ij}\right) \times ceil\left(\frac{\sum_m Q^1_{ijmt} \times Vol_m}{CT}\right) + \sum_i \sum_j \sum_m \sum_t C^7_{imt} \times Q^1_{ijmt}$$

$$+ \sum_{\mathrm{m}} \sum_{\mathrm{j}} \sum_{\mathrm{t}} \mathrm{C}^8_{\mathrm{jmt}} \times \left(\sum_{\mathrm{i}} \frac{Q^1_{ijmt}}{2} + \int_0^\infty \left(R^1_{jmt} - x_{imt}\right) f(x_{imt}) dx_{imt}\right)$$

$$+C^{10}_{jnt} \int_0^{MQ^3_{jnt}} \left(-\frac{\left(MQ^3_{jnt}\right)^2}{2 \times P} + MQ^3_{jnt} \times (T_c) - \frac{(T_c) \times x_{jnt}}{2}\right) f(x_{jnt}) dx_{jnt} + \sum_j \sum_n \sum_t C^9_{jnt} \times MQ^3_{jnt}$$

4.4.1.1.1 Transportation Cost

The Variable Transportation Cost (VTC) fluctuates in correlation with the distance covered, while the Fixed Transportation Cost (FTC) remains constant for each transporting vehicle. The last term accounts for the required number of transporting vehicles, considering the dimensional limitations imposed on these vehicles.

$$\sum_i \sum_j \sum_t \left(FTC + VTC \times d_{ij}\right) \times ceil\left(\frac{\sum_m Q^1_{ijmt} \times Vol_m}{CT}\right)$$

4.4.1.1.2 Procurement cost

The procurement cost is directly linked to the quantity of raw materials that need to be acquired, with a proportional relationship.

$$\sum_i \sum_j \sum_m \sum_t C^7_{imt} \times Q^1_{ijmt}$$

4.4.1.1.3 Inventory holding cost (Raw material)

Holding cost is getting imposed on average inventory.

$$\sum_{m}\sum_{j}\sum_{t} C_{jmt}^{8} \times \left(\sum_{i} \frac{Q_{ijmt}^{1}}{2}\right)$$

4.4.1.1.4 Inventory holding cost (Manufactured item)

In our context, we've adopted a production consumption model that accounts for stochastic demand. The cost structure is intricately tied to the overall inventory, encompassing both production and consumption aspects. Our modifications to the expression align with the specifics of our problem, rooted in the foundational production consumption model.

$$C_{jnt}^{10} \times \int_{0}^{MQ_{jnt}^{3}} \left(-\frac{\left(MQ_{jnt}^{3}\right)^{2}}{2 \times P} + MQ_{jnt}^{3} \times (T_c) - \frac{(T_c) \times x_{jnt}}{2}\right) f\left(x_{jnt}\right) dx_{jnt}$$

4.4.1.1.5 Manufacturing cost

The manufacturing cost encompasses all expenses associated with producing a single unit within the manufacturing unit. Our overall manufacturing cost is directly proportional to the total number of units produced at that specific manufacturing facility.

$$\sum_{j}\sum_{n}\sum_{t} C_{jnt}^{9} \times MQ_{jnt}^{3}$$

4.4.1.2 Product Mix:

Suppose there are m raw material and for making each product we utilize these raw materials in certain ratio such that different ratio produces different product

$$p_n = \sum_{m} a_r \times r_m$$

P_n- nth Product

r_m - mth raw Material

a_r – Ratio for material

4.4.1.3 Constraints:

4.4.1.3.1 Inventory Balance Equation

4.4.1.3.1.1 Finished Product Inventory Balance

The present-period inventory of a particular product encompasses the aggregation of residual inventory from the previous period, the produced inventory at individual manufacturing sites during each time period, and the deduction of items dispatched to various distribution centers within that interval at the corresponding facility.

$$I_{jnt} = I_{jn(t-1)} + MQ_{jnt}^{3} - \sum_{k} Q_{jknt}^{2}$$

4.4.1.3.1.2 Raw Material Inventory Constraints:

The current-period inventory of a specific raw material is formed by consolidating the remaining inventory from the preceding period, the raw materials procured from different suppliers during each time period, and subtracting the raw material consumption for the manufacturing of distinct products within that time period.

$$I_{jmt} = I_{jm(t-1)} + \sum_{i} Q^{1}_{ijmt} - \sum_{n} a_r \times MQ^{3}_{jnt}$$

4.4.1.3.2 Capacity Constraints

We have specific volume to each product, and we possess a defined storage capacity within which our inventory can be accommodated. As a result, we are limited to storing a certain number of units for various products.

$$\sum_{n} Vol_n \times MQ^{3}_{jnt} \leq VolCap_j$$

4.4.1.3.3 Non-Negativity Constraint

Essentially, these constraints ensure that our decision variables are confined within a specific state space for their potential values.

$$Q^{1}_{ijmt}, I_{jnt}, SR_{jnt}, MQ^{3}_{jnt}, \mathrm{R}^{1}_{\mathrm{jmt}}, VolCap_j, TUQ_{jnt} \geq 0$$

$$Q^{1}_{ijmt}, I_{jnt}, SR_{jnt}, MQ^{3}_{jnt}, \mathrm{R}^{1}_{\mathrm{jmt}}, VolCap_j \in I^{+}$$

4.4.2 Distribution Centres (DCs)

4.4.2.1 Objective Function

The objective function encompasses all the costs incurred by an entity, encompassing purchasing expenses, transportation costs, holding costs, and shortage cost. The DCs decisions are fundamentally guided by this comprehensive objective function, which serves as the foundation for their choices.

$$\begin{aligned} Minimize \sum_{j}\sum_{k}\sum_{t} \left(FTC + VTC \times d_{kj}\right) \times ceil\left(\frac{\sum_n Q^{2}_{jknt} \times Vol_n}{CT}\right) \\ + \sum_{j}\sum_{k}\sum_{n}\sum_{t} C^{4}_{jnt} \times Q^{2}_{jknt} \\ + \sum_{k}\sum_{n}\sum_{t} C^{5}_{knt} \times \left(\frac{\sum_j Q^{2}_{jknt}}{2} + \int_{0}^{\infty} \left(R^{2}_{knt} - x_{knt}\right) f(x_{knt}) dx_{knt}\right) \\ + \sum_{k}\sum_{n}\sum_{t} C^{6}_{knt} \int_{R^{2}_{knt}}^{\infty} \left(x_{knt} - R^{2}_{knt}\right) f\left(x_{knt} \middle| L_{jk}\right) dx_{knt} \end{aligned}$$

4.4.2.1.1 Transportation Cost

The Variable Transportation Cost (VTC) varies based on the distance traveled, while the Fixed Transportation Cost (FTC) remains consistent for each transporting vehicle. The final component addresses the necessity for a certain number of transporting vehicles, taking into account the size constraints imposed on these vehicles.

$$\sum_{j}\sum_{k}\sum_{t}\left(FTC + VTC \times d_{kj}\right) \times ceil\left(\frac{\sum_n Q_{jknt}^2 \times Vol_n}{CT}\right)$$

4.4.2.1.2 Purchasing Cost

The cost of purchasing is closely associated with the product quantity that must be obtained, demonstrating a direct proportional connection.

$$\sum_{j}\sum_{k}\sum_{n}\sum_{t} C_{jnt}^4 \times Q_{jknt}^2$$

4.4.2.1.3 Holding Cost

Holding cost is taking average order quantity level as well as quantity up to Reorder point for certain point taking stochastic demand into considerations.

$$\sum_{k}\sum_{n}\sum_{t} C_{knt}^5 \times \left(\frac{\sum_j Q_{jknt}^2}{2} + \int_0^\infty (R_{knt}^2 - x_{knt}) f(x_{knt}) dx_{knt}\right)$$

4.4.2.1.4 Shortage Cost

Shortage cost arises when we are unable to fulfill our demand, with demand during the lead time serving as a fundamental determinant of this shortage. The lead time can be either fixed or subject to stochastic variations.

$$\sum_{k}\sum_{n}\sum_{t} C_{knt}^6 \int_{R_{knt}^2}^\infty (x_{knt} - R_{knt}^2) f\left(x_{knt} \middle| L_{jk}\right) dx_{knt}$$

4.4.2.2 Constraints

4.4.2.2.1 Inventory Balance Equation

The current-period inventory for a specific product encompasses the accumulation of remaining inventory from the preceding period, the quantity received from various manufacturers, the current period's shortage, and the deduction of items sent to diverse distribution centers during that time period, along with the shortage that had been backordered from the prior period.

$$I_{knt} = I_{kn(t-1)} + \sum_{j} Q_{jknt}^2 - \sum_{k} Q_{klnt}^3 - SR_{kn(t-1)} + SR_{knt}$$

4.4.2.2.2 Capacity Constraints

We have specific volume to each product, and we possess a defined storage capacity within which our inventory can be accommodated. As a result, we are limited to storing a certain number of units for various products.

$$\sum_{n} Vol_n \times \sum_{j} Q_{jknt}^2 \le VolCap_k$$

4.4.2.2.3 Ordering Constraint

These constraints ensure that if the inventory level of any manufacturer falls below our designated ordering quantity during any given period, we refrain from placing an order with that manufacturer.

$$I_{jnt} \geq Q^2_{jknt}$$

4.4.2.2.4 Non-Negativity Constraint

Essentially, these constraints ensure that our decision variables are confined within a specific state space for their potential values.

$$Q^2_{klnt}, I_{jnt}, D_{jnt}, R^2_{lnt} \geq 0$$

$$Q^2_{klnt}, I_{jnt}, D_{jnt}, R^2_{lnt} \in I^+$$

4.4.3 Retail

4.4.3.1 Objective Function

$$\begin{aligned} minimize \sum_l \sum_k \sum_t \left(FTC + VTC \times d_{jl}\right) \times ceil\left(\frac{\sum_n Q^3_{klnt} \times Vol_n}{CT}\right) \\ + \sum_k \sum_l \sum_n \sum_t C^1_{knt} \times Q^1_{klnt} \\ + \sum_l \sum_n \sum_t C^2_{lnt} \times \left(\frac{\sum_n Q^3_{klnt}}{2} + \int_o^\infty (R^3_{lnt} - x_{lnt}) f(x_{lnt}) dx_{lnt}\right) \\ + \sum_l \sum_n \sum_t C^3_{lnt} \int_{R^3_{lnt}}^\infty (x_{lnt} - R^3_{lnt}) f(x_{lnt}|L_{kl})\} \, dx_{lnt} \end{aligned}$$

4.4.3.1.1 Transportation cost

The Variable Transportation Cost (VTC) adjusts in accordance with the distance covered, whereas the Fixed Transportation Cost (FTC) remains steady for every transport vehicle. The ultimate element deals with determining the requisite number of transport vehicles, factoring in the dimensional limitations imposed on these vehicles and quantity needed to transported.

$$\sum_l \sum_k \sum_t \left(FTC + VTC \times d_{jl}\right) \times ceil\left(\frac{\sum_n Q^3_{klnt} \times Vol_n}{CT}\right)$$

4.4.3.1.2 Purchasing cost

The purchasing cost is intricately tied to the quantity of the product that needs to be acquired, showcasing a direct proportional relationship.

$$\sum_k \sum_l \sum_n \sum_t C^1_{knt} \times Q^3_{klnt}$$

4.4.3.1.3 Inventory holding cost

Holding cost is taking average order quantity level as well as quantity up to Reorder point for certain point taking stochastic demand into considerations.

$$\sum_{l}\sum_{n}\sum_{t} C_{lnt}^{2} \times \left(\frac{\sum_{n} Q_{klnt}^{3}}{2} + \int_{o}^{\infty} (R_{lnt}^{3} - x_{lnt}) f(x_{lnt}) dx_{lnt}\right)$$

4.4.3.1.4 Shortage cost

The emergence of shortage cost occurs when our ability to meet demand falls short, and this shortfall is fundamentally influenced by the demand within the lead time. The lead time itself can take the form of a fixed duration or introduce stochastic variability.

$$\sum_{l}\sum_{n}\sum_{t} C_{lnt}^{3} \int_{R_{lnt}^{3}}^{\infty} (x_{lnt} - R_{lnt}^{3}) f(x_{lnt}|L_{kl}\} \, dx_{lnt}$$

4.4.3.2 Constraints

4.4.3.2.1 Inventory Balance Equation

The current-period inventory for a specific product encompasses the accumulation of remaining inventory from the preceding period, the quantity received from various distribution centres, the current period's shortage, and the deduction of items sold to customers during that time period, along with the shortage that had been backordered from the prior period.

$$I_{lnt} = I_{ln(t-1)} + \sum_{k} Q_{klnt}^{3} - D_{lnt} - SR_{ln(t-1)}^{1} + SR_{lnt}^{1}$$

4.4.3.2.2 Capacity Constraints

We have specific volume to each product, and we possess a defined storage capacity within which our inventory can be accommodated. As a result, we are limited to storing a certain number of units for various products.

$$\sum_{n} Vol_{n} \times \sum_{k} Q_{klnt}^{1} \leq VolCap_{l}$$

4.4.3.2.3 Ordering Constraints

These constraints ensure that if the inventory level of any distribution centers falls below our designated ordering quantity during any given period, we refrain from placing an order with that distribution centers.

$$I_{knt} \geq Q_{klnt}^{1}$$

4.4.3.2.4 Non-Negativity Constraints

Essentially, these constraints ensure that our decision variables are confined within a specific state space for their potential values.

$$Q_{klnt}^{3}, I_{lnt}, D_{lnt}, R_{lnt}^{1} \geq 0$$

$$Q_{klnt}^{3}, I_{lnt}, D_{lnt}, R_{lnt}^{1} \in I^{+}$$

5.Data Preparation

i = 1,2,3 (Three Suppliers)
j = 1,2,3,4 (Four Manufacturer)
k = 1,2,3,4,5 (Five Depot)
l = 1,2,3……10 (Ten Retails)
m = 1,2 (Two Raw Material)
n = 1,2 (Two Product)

Symbol	**Parameter**	**Range**
d_{ij}	Distance between ith supplier to jth manufacturer	U [300,800]
d_{jk}	Distance between jth manufacturer to kth DC	U [600,1600]
d_{kl}	Distance between kth DC to lth retail	U [500,1200]
FTC_{ij}	Fixed transportation cost from ith supplier to jth manufacturer	₹ 1,000.00
FTC_{jk}	Fixed transportation cost from jth manufacturer to kth DC	₹ 1,000.00
FTC_{kl}	Fixed transportation cost from kth DC to lth retail	₹ 1,000.00
VTC_{ij}	Variable transportation cost from ith supplier to jth manufacturer	₹ 80.00
VTC_{jk}	Variable transportation cost from jth manufacturer to kth DC	₹ 80.00
VTC_{kl}	Variable transportation cost from kth DC to lth retail	₹ 80.00
C^1_{k1t}	Selling price of product 1 by kth DC	₹ U [60,63]
C^1_{k2t}	Selling price of product 2 by kth DC	₹ U [69,73]
C^4_{j1t}	Selling price of product 1 by jth Manufacturer	₹ U [36,38]
C^4_{j2t}	Selling price of product 2 by jth Manufacturer	₹ U [40,42]
C^7_{i1t}	Selling price of raw material 1 by ith Supplier	₹ U [5,7]
C^7_{i2t}	Selling price of raw material 2 by ith Supplier	₹ U [5,7]
CT	Capacity of Truck across supply chain	20000 units
C^2_{lnt}	Holding cost for lth retail as a percentage of cost of product	6.0%
C^{10}_{jnt}	Holding cost for jth DC as a percentage of cost of product	1.5%
C^8_{jmt}	Holding cost of jth Manufacturer as a percentage of cost of product	2.0%
D_{l1t}	Demand of lth retail for product 1	N [13350,3100]
D_{l2t}	Demand of lth retail for product 2	N [7000,1700]
$VolCap_j$	Capacity of jth Manufacturer in Lt.	U [60000,65000]
$VolCap_k$	Capacity of kth DC in Lt.	U [45000,50000]
$VolCap_l$	Capacity of lth Retail in Lt.	U [200,400]

Table 1 Parameters Used in Project

6. Methodology Adopted

Integer Programming (IP) problems can be solved using a range of techniques and algorithms, with the selection of a specific method driven by factors such as the problem's attributes, scale, and complexity. The choice of methodology utilized is dependent on the attributes, scale, and complexity of the problem. Presented below are some of the commonly employed techniques for the resolution of integer programming problems:

6.1 Branch and Bound (B&B)

Branch and bound is a fundamental algorithm that is utilized for the resolution of IP problems. It systematically explores the solution space by dividing it into smaller subproblems through branching on integer variables. The algorithm maintains an upper and lower bound on the optimal solution and prunes branches that cannot lead to an improved solution.

6.2 Branch and Cut (B&C)

Branch and Cut is a modification of the Branch and Bound approach that combines branching with the addition of valid inequalities (cuts) to reinforce the Linear Programming (LP) relaxation. The cuts help tighten the relaxation, reducing the search space and potentially accelerating convergence.

6.3 Cutting Plane

Cutting plane methods, as previously mentioned, incorporate valid inequalities into the LP relaxation to strengthen the relaxation and enhance the bounds. Gomory mixed-integer cuts and Lift-and-Project cuts are the commonly employed techniques in this context.

6.3.1 Key Concepts behind the Cutting Plane Method

6.3.1.1 Integer Solutions and Relaxations

The Cutting Plane Method is utilized to address challenging Mixed-Integer Programming (MIP) problems that involve both continuous and discrete decision variables. To make the problem more tractable, we commence by relaxing all integer constraints and transforming it into a Linear Programming (LP) problem. The LP relaxation provides us with a lower bound on the optimal solution value.

6.3.1.2 Valid Inequalities

Our objective is to refine the solution space and approach the optimal integer solution. Here, valid inequalities play a significant role. They are constraints that hold true for all feasible integer solutions in the original MIP problem. By integrating these valid inequalities into the LP relaxation, we decrease the feasible region and make it more restrictive.

6.3.1.3 Cutting Planes

Cutting planes refer to specific valid inequalities derived from the characteristics of the MIP problem. Our aim is to pinpoint violated cutting planes, indicating that the current LP solution does not satisfy these inequalities. When we find a violated cutting plane, we incorporate it in the LP relaxation to exclude the current solution and other similar ones.

6.3.2 General Steps of the Cutting Plane Method

Step 1: Formulate the LP Relaxation: Relax all integer constraints to transform the MIP problem into a continuous Linear Programming problem.

Step 2: Solve the LP Relaxation: Utilize a linear programming solver to obtain an initial solution, serving as a lower bound on the optimal integer solution.

Step 3: Check for Integer Feasibility: Verify if the current LP solution satisfies all integer constraints. If it does, we have an optimal integer solution, and the process concludes.

Step 4: Identify Violated Valid Inequalities (Cutting Planes): Examine the current LP solution to identify valid inequalities that are violated, i.e., not satisfied.

Step 5: Add Violated Valid Inequalities: Incorporate the violated valid inequalities identified in the LP relaxation to refine the solution space.

Step 6: Repeat Steps 2 to 5: Iterate by solving the updated LP relaxation, identifying more violated valid inequalities, and refining the solution space until we obtain an integer feasible solution.

Termination:

The Cutting Plane Method concludes when we obtain an integer feasible solution. The algorithm guarantees finding an optimal integer solution if we solve the LP relaxation to optimality and include all violated valid inequalities

6.4 Branch and Price

Branch and Price is an algorithm that is particularly useful for solving large-scale IP problems with a specific structure, such as column generation in column-wise modelling. It iteratively adds variables (columns) to the LP relaxation to find new feasible solutions and improve the bounds.

6.5 Dynamic Programming (DP)

Dynamic programming can be used for certain types of IP problems, particularly when there is a specific recursive relationship among subproblems. This technique is often applicable to problems with overlapping substructures.

6.6 Heuristic Methods

Heuristics are optimization algorithms that sacrifice optimality for efficiency. Heuristic methods prove to be particularly useful in addressing large-scale problems where discovering the optimal solution is computationally infeasible. Examples include Genetic Algorithms, Simulated Annealing, and Tabu Search.

6.7 Local Search Methods

Local search methods explore the neighborhood of a given solution to iteratively improve it. They do not guarantee finding the global optimal solution, but they can be effective for finding good feasible solutions in a short amount of time.

6.8 Branch and Bound with Rounding

This approach combines rounding techniques with Branch and Bound to find feasible integer solutions quickly. Rounding solutions obtained from the LP relaxation can sometimes produce acceptable integer solutions without exploring the entire solution space.

6.9 Outer Approximation

The Outer Approximation algorithm is employed to solve problems of Mixed-Integer Nonlinear Programming (MINLP). It iteratively linearizes the nonlinear functions at each iteration and adds cutting planes to the LP relaxation.

7. Implementation

The use of linear programming (LP) has become increasingly prevalent in the optimization of decision-making problems, particularly in supply chain management. In this study, we delve into the implementation of a supply chain optimization model utilizing Python and the open-source library Pyomo. Our objective is to showcase how Pyomo, in conjunction with the Gurobi solver, efficiently manages linear programming problems. We will commence by discussing the necessity of assumptions to convert a non-linear model into a linearized one, followed by the implementation process and the benefits of utilizing Pyomo and Gurobi for this purpose.

7.1 The Challenge of Non-Linearity

The optimization of supply chains frequently entails intricate interactions and relationships among numerous variables, leading to non-linearities in the mathematical model. Non-linear models pose a greater challenge for solution than linear models due to the existence of non-linear functions like quadratic and integration etc. These intricacies could significantly augment the computational onus and, in certain eventualities, make the predicament impracticable.

7.2 Assumption-based Linearization

To address the intricacy of non-linear models, we resort to the practice of linearization. Linearization consists of making appropriate assumptions to convert the non-linear functions into linear ones, thereby simplifying the problem. By introducing these assumptions, we create a manageable linear programming model that can be efficiently solved.

In our study, we meticulously selected and applied appropriate assumptions to ensure that our supply chain optimization problem remains practically feasible whilst being linearized. One such crucial assumption is imposing a shortage of zero at all levels. This assumption is frequently reasonable in certain supply chain contexts where the company aims to meet demand without any backlog or shortages.

7.3 Pyomo and its Advantages

Pyomo, an open-source optimization modelling language, provides an elegant and flexible framework for modelling linear and non-linear programming problems. It offers support for a variety of solvers, including open-source solvers like GLPK and commercial solvers like CPLEX and Gurobi, making it a highly adaptable choice for supply chain optimization tasks.

The Pyomo modelling process comprises defining decision variables, constraints, and the objective function in a natural algebraic syntax, enhancing the model's readability and maintainability. Additionally, Pyomo enables easy integration with external solvers, streamlining the process of attaining an optimal solution.

7.4 Gurobi Solver and its Role

After formulating the supply chain optimization model in Pyomo, we utilize the Gurobi solver to obtain the optimal solution. Gurobi is a top-of-the-line commercial optimization solver renowned for its efficiency and resilience. The system uses complex algorithms, like quadratic programming, mixed-integer programming, and linear programming techniques, to effectively handle optimization problems of enormous proportions.

By leveraging Gurobi as the solver for our Pyomo model, we take advantage of its cutting-edge optimization capabilities to obtain an optimal solution quickly and accurately.

In conclusion, we effectively demonstrated how to model a supply chain optimization problem using Pyomo while efficiently handling the intricacies of non-linear functions through assumption-based linearization. By adopting the appropriate assumptions, we transformed the non-linear model into a manageable linear programming model, significantly simplifying the solution process.

8.Results & Discussion

The multi-objective function was solved using classical method by converting it into a single objective function, the base condition was to keep all the weights of three objective function as same. The optimal solution was found after 1878 simplex iterations.

Time Period	Purchasing Cost	Manufacturing Cost	Fixed_Transportation Cost	Variable_Transportaion Cost	Holding Cost	Total Period Wise Cost	Supply chain Cost
1	₹ 3,03,19,812.25	₹ 8,81,467.50	₹ 44,781.11	₹ 16,10,937.40	₹ 6,66,540.32	₹ 3,35,23,538.57	₹ 23,22,258.83
2	₹ 3,04,09,292.29	₹ 11,25,994.00	₹ 44,956.50	₹ 16,96,824.87	₹ 6,52,521.93	₹ 3,39,29,589.59	₹ 23,94,303.30
3	₹ 3,08,25,924.76	₹ 9,19,559.50	₹ 45,032.44	₹ 16,88,538.56	₹ 6,99,201.88	₹ 3,41,78,257.14	₹ 24,32,772.88
4	₹ 2,70,28,778.36	₹ 7,70,832.50	₹ 44,855.40	₹ 15,07,633.25	₹ 6,00,731.54	₹ 2,99,52,831.05	₹ 21,53,220.19
5	₹ 3,16,31,040.79	₹ 8,69,596.50	₹ 45,036.04	₹ 16,95,378.89	₹ 6,77,653.61	₹ 3,49,18,705.83	₹ 24,18,068.54
6	₹ 2,92,72,947.83	₹ 7,04,080.00	₹ 41,285.66	₹ 14,70,769.47	₹ 6,53,138.40	₹ 3,21,42,221.36	₹ 21,65,193.54
7	₹ 2,97,79,338.01	₹ 8,76,023.50	₹ 42,924.34	₹ 16,33,728.47	₹ 6,55,701.41	₹ 3,29,87,715.73	₹ 23,32,354.22
8	₹ 3,02,57,491.73	₹ 9,02,522.50	₹ 44,726.78	₹ 15,86,482.39	₹ 6,62,012.48	₹ 3,34,53,235.87	₹ 22,93,221.64
9	₹ 3,09,07,628.29	₹ 9,16,768.50	₹ 41,510.40	₹ 15,92,679.40	₹ 6,95,983.30	₹ 3,41,54,569.89	₹ 23,30,173.10
10	₹ 2,97,67,282.82	₹ 7,80,379.00	₹ 42,601.99	₹ 15,54,207.22	₹ 6,69,251.95	₹ 3,28,13,722.97	₹ 22,66,061.15
11	₹ 2,97,39,008.06	₹ 8,07,286.50	₹ 43,367.48	₹ 15,46,194.45	₹ 6,55,554.40	₹ 3,27,91,410.87	₹ 22,45,116.32
12	₹ 2,66,09,019.74	₹ 7,88,023.00	₹ 37,737.15	₹ 14,02,254.46	₹ 5,89,092.98	₹ 2,94,26,127.33	₹ 20,29,084.59
	₹ 35,65,47,564.91	₹ 1,03,42,533.00	₹ 5,18,815.28	₹ 1,89,85,628.84	₹ 78,77,384.19	₹ 39,42,71,926.22	₹ 2,73,81,828.31
			1.89%	69.34%	28.77%		100.00%

Table 2 Total Cost Period Wise

8.1 Decision variables:

There were four decision variables which was solved for that were Manufactured quantity at each plant M_{jnt}, the quantity of raw material supplied from supplier to plant Q_{ijmt}, the quantity of finished products sent from plant to distribution centers (DCs) Q_{jknt} and the quantity of finished products supplied from DCs to Retail $Q_{klnt.}$

The demand from each retail for different products drive the supply chain which varies across different time periods.

8.1.1 Manufactured Quantity at different Manufacturer:

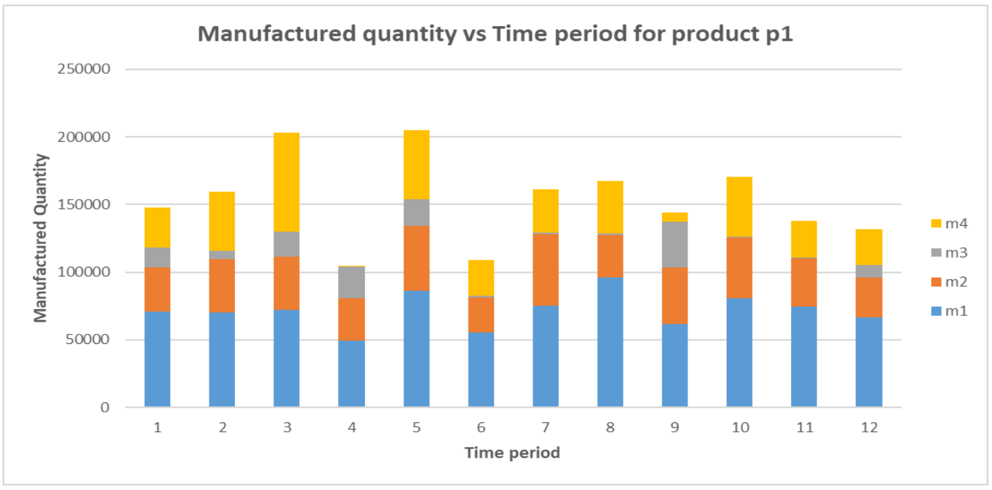


Figure 2: Manufactured quantities by the manufacturers for product 1

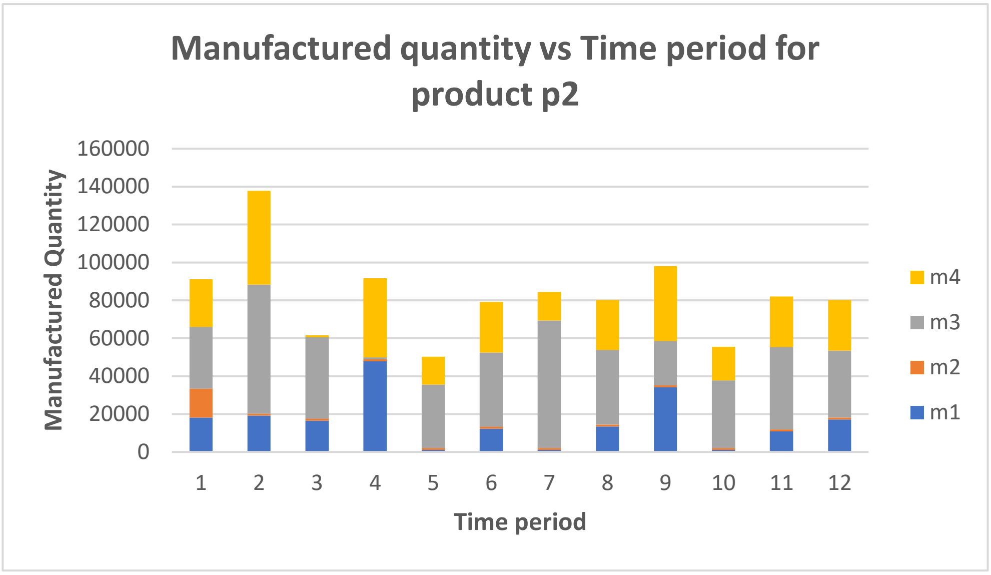


Figure 3: Manufactured quantities by the manufacturers for product 2

It was found that there was significant difference in manufactured quantities produced by same manufacturer. It is observed that manufacturer produces large quantities of product type 2 than type 1 to reduce the overall supply chain cost.

8.1.2 Quantity of products supplied from supplier to Manufacturer (Q1)

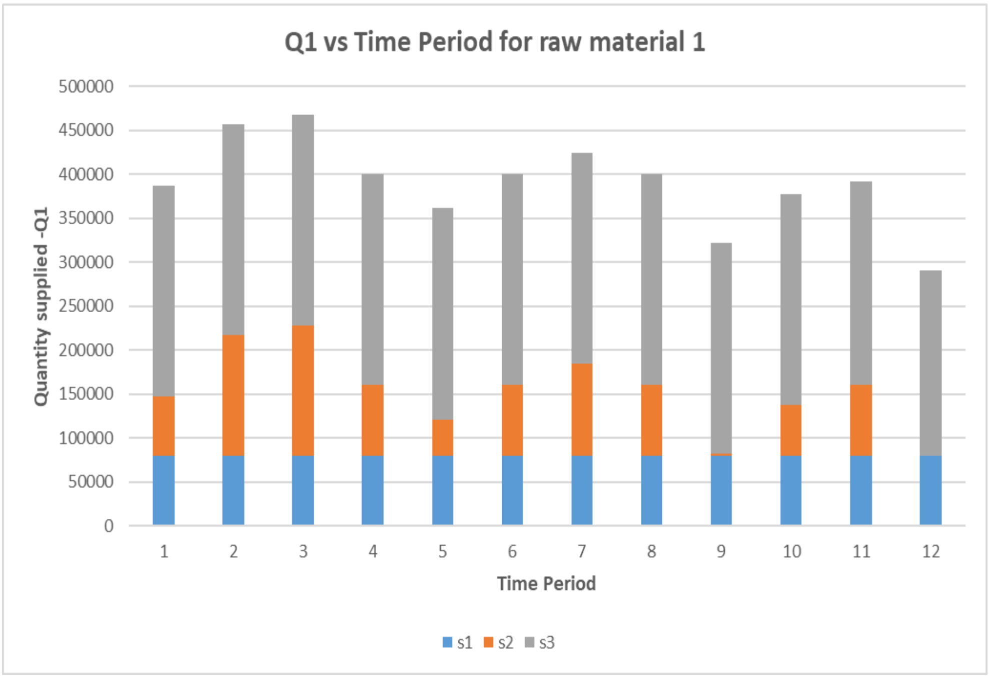


Figure 4: Quantities of raw material 1 supplied by different suppliers

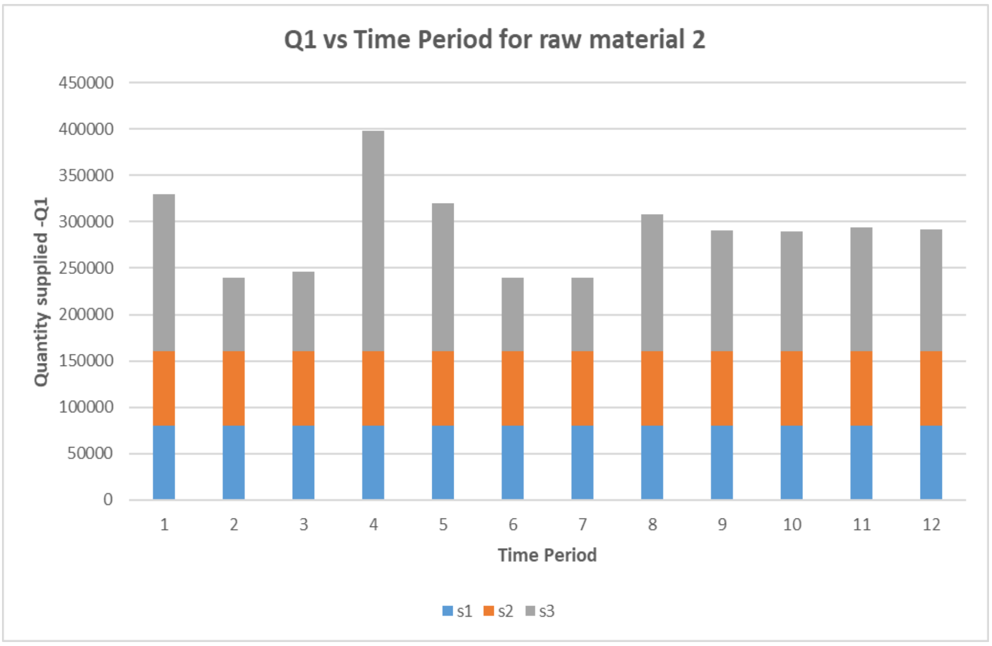


Figure 5: Quantities of raw material 2 supplied by different suppliers

It was observed that **supplier 3** supplied maximum raw materials to the manufacturer 28.4Lac raw material 1, 15.6Lac raw material 2 and an aggregate of 44.1Lac.

8.1.3. Quantity of products supplied from Manufacturer to DCs (Q2)

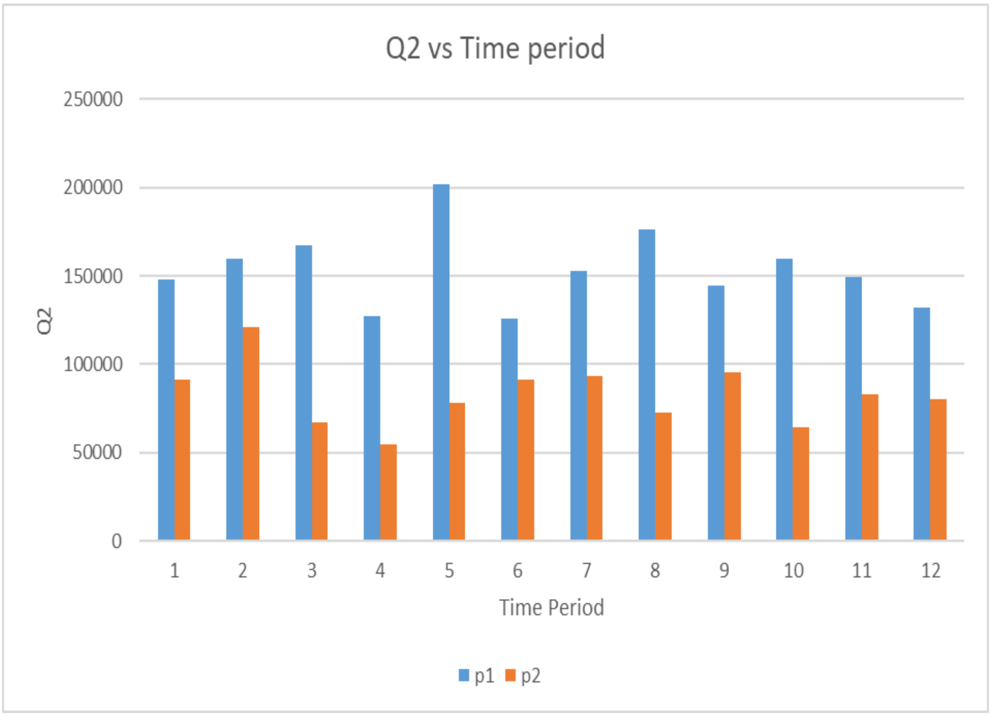


Figure 6: Overall quantity supplied from DCs for different products

Due to greater demand in **period 5** most of the products are supplied from manufacturer to DC. It can also be observed that maximum products of type p1 is manufactured during this period (fig2).

8.1.4 Quantity of products supplied from DCs to Retail (Q3)

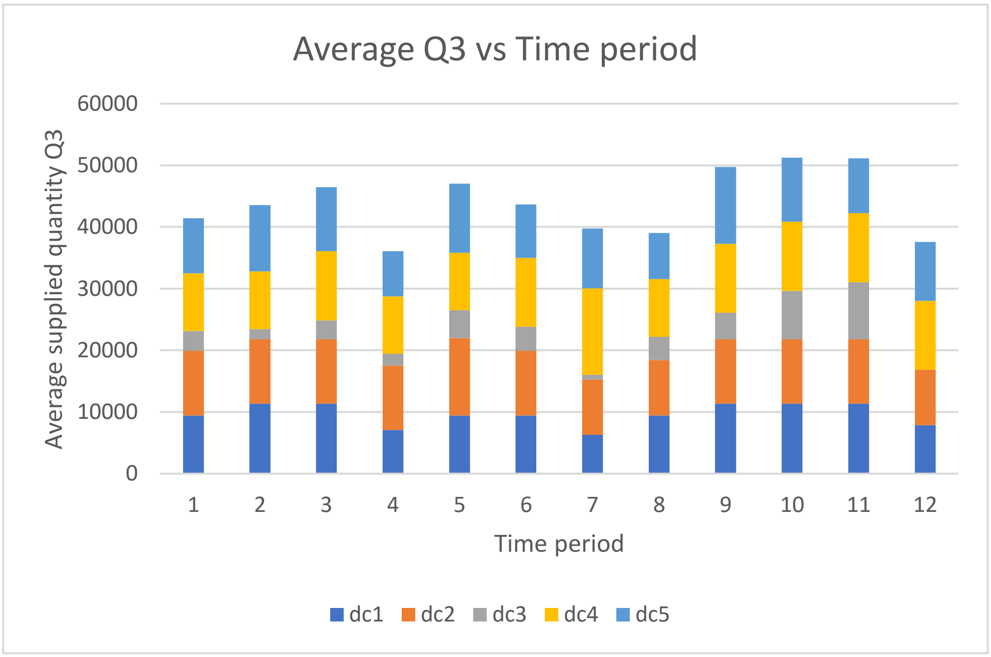


Figure 7: Average quantity supplied from DC to Retail

It was observed that DC3 supplies least quantities to different retails, since DC3 was farthest of the other DCs. There is also a case where DC3 does not supply any products in 12th time period. Hence, it can be concluded that product supplied from DC3 had the more transportation cost compared to other DCs. Hence, as a stakeholder it would be beneficial for DC3 to reduce its product price compared to other DCs so that the overall supply chain cost might be reduced and DC3 could become more competitive with respect to other DCs.

8.2 Sensitivity analysis

Sensitivity analysis was done to check the effect of changing parameters on the costs at different time periods. The parameters chosen were **fixed transportation cost (FTC), variable transportation cost (VTC), inventory holding cost and capacity of transporting Vehicle (CT)**.

It was observed that **FTC, VTC and inventory holding cost** had **positive correlation** with the total supply chain cost. And the **CT** is negatively correlated with **total supply chain cost.**

Parameters	FTC						Holding Cost					
Change in parameters	-30%	-20%	-10%	10%	20%	30%	-30%	-20%	-10%	10%	20%	30%
Change in Supply Chain Cost	-0.568%	-0.379%	-0.190%	0.189%	0.379%	0.568%	-8.043%	-5.335%	-2.681%	2.681%	5.362%	8.043%
Parameters	CT						VTC					
Change in parameters	-30%	-20%	-10%	10%	20%	30%	-30%	-20%	-10%	10%	20%	30%
Change in Supply Chain Cost	28.718%	16.296%	7.440%	-6.367%	-11.510%	-15.961%	-20.162%	-11.321%	-6.824%	6.464%	12.537%	19.228%

Table 3: Results of sensitivity analysis

It was also observed that the total supply chain cost was **more sensitive** towards the parameters **VTC and Truck capacity**. And **least sensitive** towards the parameter **FTC.**

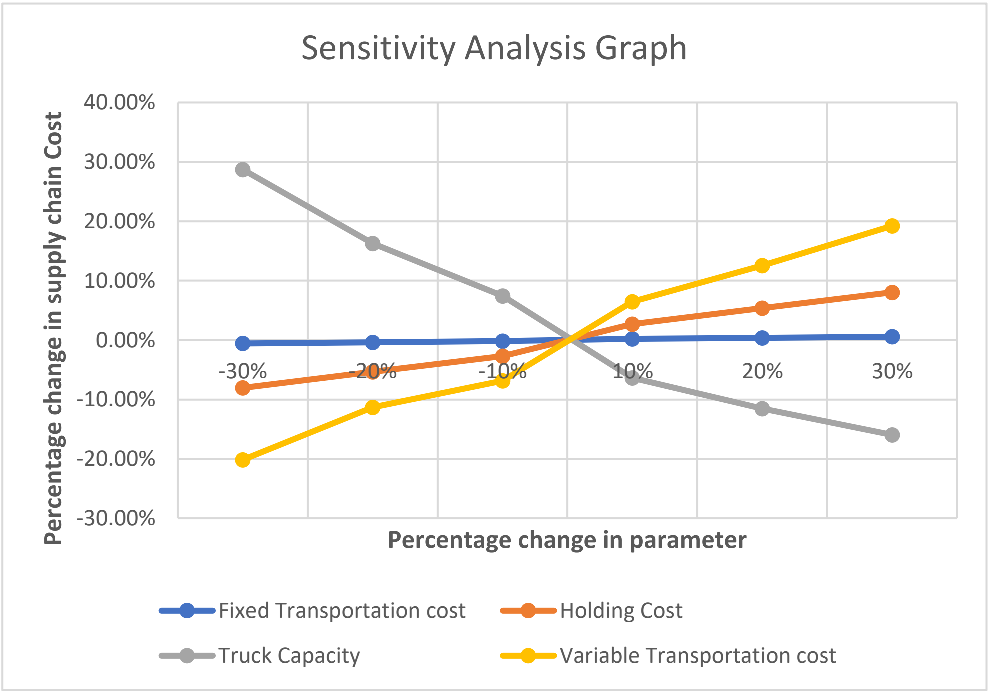


Figure 8: Sensitivity analysis of different parameters

It was also observed that inventory holding cost increased as a result of decrease in FTC as more quantity is transported due to decreased.

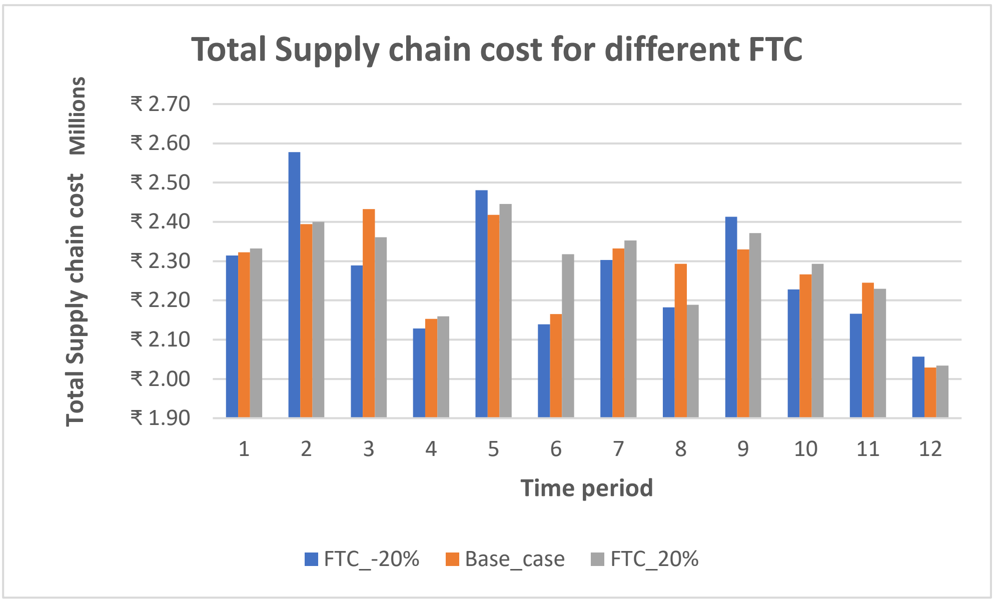


Figure 9: Total supply chain cost for different FTC

It is clearly seen that FTC has no clear correlation to total supply chain cost. It is also noted from the sensitivity graph (fig8) that it has least effect on total supply chain cost.

Hence, it would can be said that fixed transportation cost has least effect on the overall supply chain cost as per the hypothetical scenario taken. However, it should be noted that only one type of transport was considered and multimodal scenario was not considered.

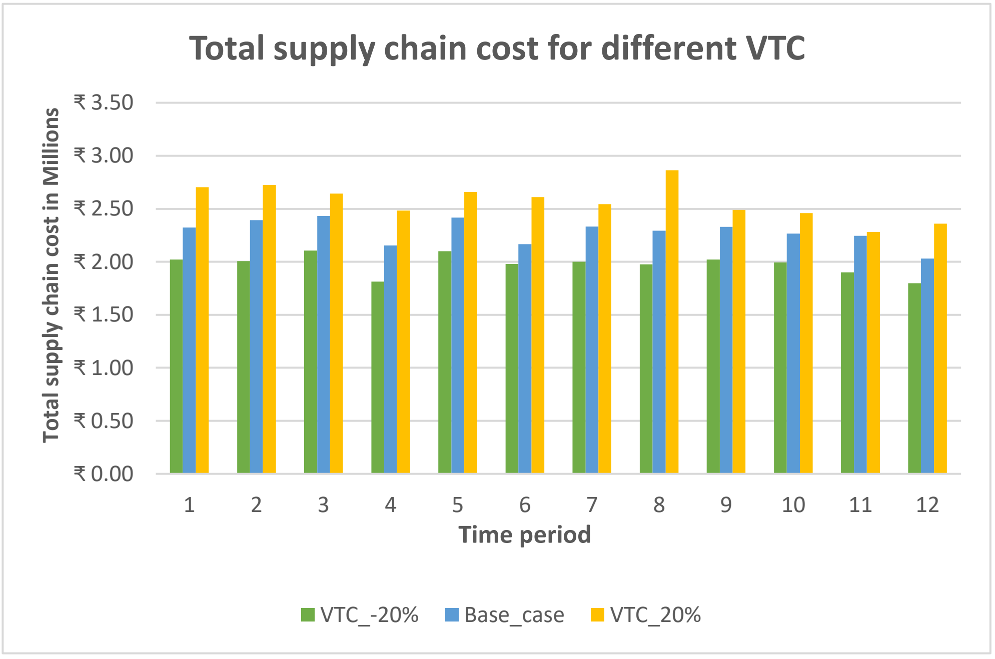


Figure 10: Total supply chain cost for different VTC

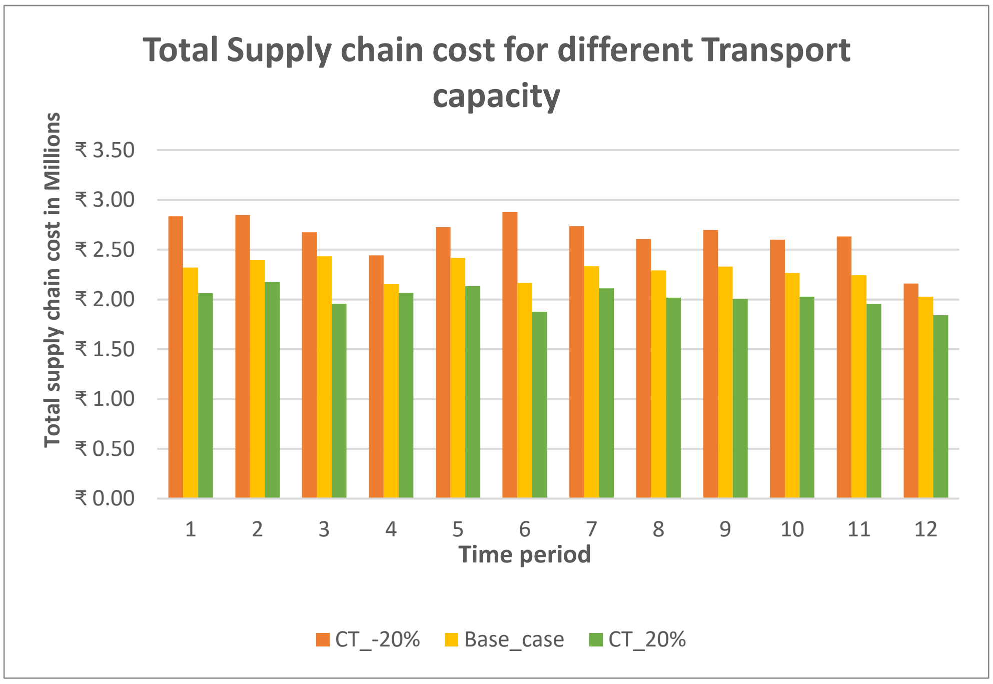


Figure 11: Total supply chain cost for different Truck capacity

In contrast to FTC, VTC and transport capacity has stronger correlation with overall supply chain cost. VTC contributes to 69% of total supply chain cost.

However, the relationship of VTC and cost, transport capacity and cost are different.

VTC: it has a positive correlation with total supply chain cost i.e increasing the cost per km would increase the overall transportation cost. This in-turn depends on fuel prices, labour costs etc. Hence, VTC is subject to macro-economic factors and could be volatile across different time periods. However, in the present model same VTC is taken for each time period.

Transport capacity: It has a negative correlation with overall supply chain cost. Since, increasing capacity of transport will decrease overall transportation cost as more goods can be packed and supplied for less cost.

However, this is the case only when we assume that increasing capacity of transport has no effect on VTC and FTC. But, the real life scenario is otherwise. There needs to be real life data or a study to identify the relationship between FTC/VTC with transport capacity which is left as future scope of study.

Inventory Holding cost

Inventory holding cost accounts to 29% of overall supply chain cost. Hence it is bound to have significant effect on overall supply chain cost.

The holding cost is calculated as holding cost * average inventory at different levels. Based on this approach we find that decreasing holding cost leads to decrease in inventory cost and hence decrease in overall supply chain cost.

In real life scenario holding cost can be reduced by improving better packing efficiency, reducing electricity cost by using energy efficient equipment etc.

There are cases in the solution where some DCs stock excess inventory than needed in the time period. And the inventory is large enough that it can supply the demand in the next time period without placing order downstream of the supply chain (fig 14).

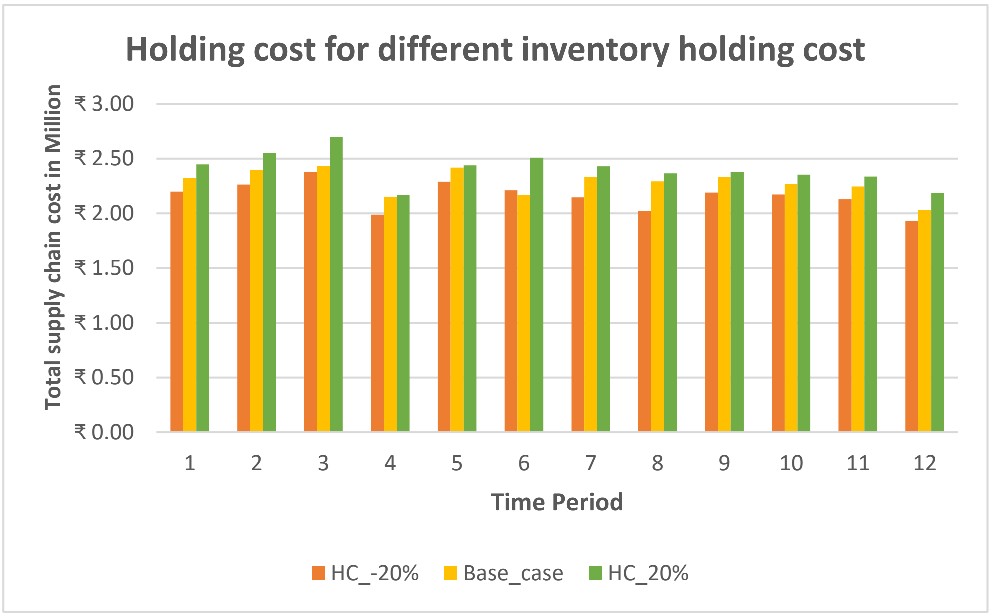


Figure 12: Total supply chain cost for different inventory holding cost

8.3 Network Diagrams

Network diagram was generated for each time periods for both the products as shown in the figure below.

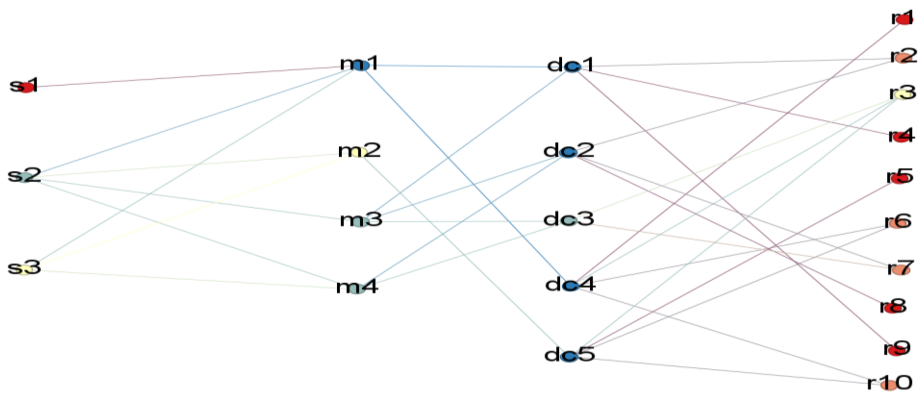


Figure 13: Network Diagram for time period 1 for product p1

It was observed in network diagram for time period 5 for product p2, the DC3 directly supplies product to retail 3. Since the demand in this time period is lower compared to the 5th time period (fig.3) hence the products are stocked in advance in DC3 at time period 4 and the demand of R3 is directly fulfilled in time period 5. As the stated before the holding cost of DC is lower compared to the transportation cost leading to the stocking behavior of the DC.

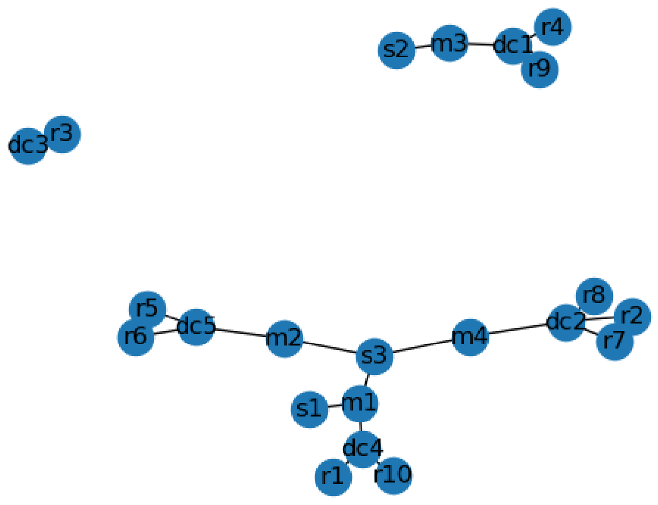


Figure 14: Network Constellation Diagram for time period 5 for product p2

9. Follow-up Work and Future Scope

The initial project goal was accomplished using simulated data in a hypothetical scenario. However, there are several potential areas for expansion. These include incorporating real-life data, accounting for uncertain lead times, addressing product shortages at distribution centers and retail levels, and implementing time series forecasting methods to predict demand. Additionally, the project could explore the application of non-classical approaches like the genetic algorithm (NSGA2) to obtain the Pareto front and compare different metaheuristics for obtaining solutions.

Furthermore, the project could explore the use of optimization techniques like "optimizing by simulation" and other various methods such as machine learning and reinforcement learning. These advanced approaches can help enhance the efficiency and accuracy of the supply chain and inventory management system by learning from data and adapting to dynamic environments. By incorporating these techniques, the project can achieve more robust and adaptive solutions for complex real-world scenarios.

10. References

1. P. Tsiakis, N. Shah, and C. C. Pantelides (2001), Design of Multi-echelon Supply Chain Networks under Demand Uncertainty
2. Brojeswar Pal, Shib Sankar Sana, Kripasindhu Chaudhuri (2012), A three layer multi-item production–inventory model for multiple suppliers and retailers, ScienceDirect
3. Prasun Das, Subhasis Chaudhury (2008), Optimisation of supply chain inventory for multi-retail and multiitem class consumer product problems using genetic algorithm, Int. J. Productivity and Quality Management
4. G. Kannan, P. Sasikumar, K. Devika (2010), A genetic algorithm approach for solving a closed loop supply chain model: A case of battery recycling
5. Sasan Khalifehzadeh, Mehdi Seifbarghy, Bahman Naderi (2015), A four-echelon supply chain network design with shortage, Mathematical modelling and solution methods. Elsevier
6. B. Latha Shankar, S. Basavarajappa, Jason C.H. Chen, Rajeshwar S. Kadadevaramath (2013), Location and allocation decisions for multi-echelon supply chain network –A multi-objective evolutionary approach
7. Anil Jindal, Kuldip Singh Sangwan, and Sachin Saxena(2015), Network Design and Optimization for Multi-product, Multi-time, Multiechelon Closed-loop Supply Chain under Uncertainty
8. Hannan Sadjady, Hamid Davoudpour (2012), Two-echelon, multi-commodity supply chain network design with mode selection, lead-times and inventory costs. Elsevier
9. Hongwei Wang, Min Guo, Janet Efstathiou (2004), A game-theoretical cooperative mechanism design for a two-echelon decentralized supply chain. Elsevier
10. Marjia Haque, Sanjoy Kumar Paul, Ruhul Sarker, Daryl Essam (2020), Managing decentralized supply chain using bilevel with Nash game approach
11. Robin Roundy (1985), 98%-Effective integer – ratio lot-sizing for one warehouse multi-retailor systems
12. Shilpi Pal, G. S. Mahapatra and G. P. Samanta (2016), A Three-Layer Supply Chain EPQ Model for Price- and Stock-Dependent Stochastic Demand with Imperfect Item Under Rework
13. Moutaz Khouja (2003), Optimizing inventory decisions in a multi-stage multi-customer supply chain
14. Kit Nam Francis Leung (2009), A technical note on ‘‘Optimizing inventory decisions in a multi-stage multi-customer supply chain”
15. M.E. Seliaman, Ab Rahman Ahmad (2008), Optimizing inventory decisions in a multi-stage supply chain under stochastic demands
16. Brojeswar Pal, Shib Sankar Sana, Kripasindhu Chaudhuri (2013), A mathematical model on EPQ for stochastic demand in an imperfect production system
17. Leopoldo Eduardo Cárdenas-Barrón(2011), The derivation of EOQ/EPQ inventory models with two backorders costs using analytic geometry and algebra
18. Leopoldo Eduardo Cárdenas-Barrón (2010), An easy method to derive EOQ and EPQ inventory models with backorders
19. Biswajit Sarkar, Shib Sankar Sana, Kripasindhu Chaudhuri (2011), An economic production quantity model with stochastic demand in an imperfect production system
20. Debabrata Das, Nirmal Baran Hui, Vipul Jain (2018), Optimization of stochastic, (Q, R) inventory system in multi-product, multi-echelon, distributive supply chain
21. G.P. Kiesmüller (2003), A new approach for controlling a hybrid stochastic manufacturing/remanufacturing system with inventories and different lead-times

Appendix

This section contains Python code used extensively in generating the results presented in the report. The code serves as a crucial tool for producing the findings and analysis discussed throughout the document.

```
#!/usr/bin/env python
# coding: utf-8


import pandas as pd
import sys
import pyomo.environ as pe
from itertools import product
import numpy as np
import math


#data
number_of_suppliers=['s1','s2','s3']
number_of_plants=['m'+ str(i) for i in range(1,5)]
number_of_dcs=['dc'+str(i) for i in range(1,6)]
number_of_retails=['r'+ str(i) for i in range (1,11)]

number_of_products=['p'+ str(i) for i in range (1,3)]
number_of_raw_material=['raw_m'+str(i) for i in range (1,3)]
number_of_time_periods=[i for i in range (1,13)]


#Sensitivity factors
#default a,b,c==1
a=1
b=1
c=1
e=1



#Capacities of each of manufacturer, DC and Plant
#1 Plant
df=pd.read_excel('Capacity_j.xlsx')
col=list(df.columns[[0]])
df_temp=df.set_index(col)
Capacity_p = dict(zip(df_temp.index, df_temp['Capacity_Q']))


#2 DC
df=pd.read_excel('Capacity_k.xlsx')
col=list(df.columns[[0]])
df_temp=df.set_index(col)
Capacity_dc=dict(zip(df_temp.index, df_temp['Capacity_Q']))

#3 Retail
df=pd.read_excel('Capacity_l.xlsx')
col=list(df.columns[[0]])
df_temp=df.set_index(col)
Capacity_ret= dict(zip(df_temp.index, df_temp['Capacity_Q']))

#Demand of the retials
df=pd.read_excel('Demand_k_n_t.xlsx')
```

```
col=list(df.columns[[0,1,2]])
df_temp=df.set_index(col)
Demand_ret= dict(zip(df_temp.index, df_temp['Demand']))


model=pe.ConcreteModel()


#model indexes
model.number_of_suppliers=pe.Set(initialize=number_of_suppliers)
model.number_of_plants=pe.Set(initialize=number_of_plants)
model.number_of_dcs=pe.Set(initialize=number_of_dcs)
model.number_of_retail=pe.Set(initialize=number_of_retails)
model.number_of_raw_material=pe.Set(initialize=number_of_raw_material)
model.number_of_products=pe.Set(initialize=number_of_products)
model.number_of_time_periods=pe.Set(initialize=number_of_time_periods)


#Adding constraints parameter
model.Capacity_j=pe.Param(model.number_of_plants,initialize=Capacity_p)
model.Capacity_k=pe.Param(model.number_of_dcs,initialize=Capacity_dc)
model.Capacity_l=pe.Param(model.number_of_retail,initialize=Capacity_ret)

#Adding Demand paramter
model.Demand_l=pe.Param(model.number_of_retail,model.number_of_products,number_of_time_periods,initi
alize=Demand_ret)


# d=list(product(number_of_suppliers,number_of_plants,number_of_raw_material))
# column_names = ['Supplier', 'Manufacturer', 'Raw Material']
# df=pd.DataFrame(d,columns=column_names)
# df.to_excel('purchase_cost_i_j_m.xlsx')

#Creating a paramter master file between supplier and manufacturer
#product of i,j,m and t
d=list(product(number_of_suppliers, number_of_plants,number_of_raw_material,number_of_time_periods))

column_names = ['Supplier', 'Manufacturer', 'Raw Material', 'Time Period']
df=pd.DataFrame(d,columns=column_names)
df.to_excel('Master_file_i_j_m_t.xlsx')
#Master file for Manufacturer, product and time period
d=list(product(number_of_plants,number_of_products,number_of_time_periods))
column_names = [ 'Manufacturer', 'Product', 'Time Period']
df_manu=df=pd.DataFrame(d,columns=column_names)
df_manu.to_excel('Master_file_j_n_t.xlsx')

df4=pd.read_excel('Manufacturing_cost_j_n.xlsx')
df_manu=df_manu.merge(df4,how='left',left_on=['Manufacturer','Product'],right_on=['Manufacturer','Product'])



#Parameters to be added Pruchase cost, Holding cost, Transportation, manufacturing cost,Shortage cost(now not
considered)
#To add distance we will join
df1=pd.read_excel('Master_file_i_j_m_t.xlsx')
df2=pd.read_excel('distance_i_j.xlsx')
df3=pd.read_excel('purchase_cost_i_j_m.xlsx')

df1=df1.merge(df2,how='left', left_on=['Supplier','Manufacturer'],right_on=['Supplier','Manufacturer'])
```

```
df1=df1.merge(df3,how='left',left_on=['Supplier','Manufacturer','Raw
Material'],right_on=['Supplier','Manufacturer','Raw Material'])

df1 = df1.drop(df1.columns[0], axis=1)
#Defining fixed transportation cost as 10,000 per load
df1['FTC']=1000*a
#Variable transportation cost as 80/km
df1['VTC']=80*b
#Volume of each Q1_i_j_m_t 750ml
df1['Vol']=.75
#Truck Capacity 20cubm
df1['CT']=20000*c
#df1=df1.drop(df1.columns[5],axis=1)
df1['Holding_Cost']=df1['Purchase_Cost']*(0.02*e) #Wholesale price



#1.Creating a dictionary for purchase cost
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
Purchasing_cost_s_p= {key: value for key, value in df_temp['Purchase_Cost'].items()}

#2.Dict for Holding cost
col=list(df1.columns[[1,2,3]])
df_temp=df1.set_index(col)
Holding_cost_s_p= {key: value for key, value in df_temp['Holding_Cost'].items()}

#3.Dict for distance
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
distance_s_p= {key: value for key, value in df_temp['Distance'].items()}

#4.Dict for FTC
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
FTC_s_p= {key: value for key, value in df_temp['FTC'].items()}

#5.Dict for VC
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
VTC_s_p= {key: value for key, value in df_temp['VTC'].items()}

#6.Dict for Truck capacity
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
Truck_capacity_s_p= {key: value for key, value in df_temp['CT'].items()}

#6.Dict for Vol
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
Volume_s_p= {key: value for key, value in df_temp['Vol'].items()}

#Dict for Manufacturing cost
col=list(df_manu.columns[[0,1,2]])
df_temp=df_manu.set_index(col)
Manuf_p_prod= {key: value for key, value in df_temp['Manufacturing_cost'].items()}
```

```
#Creating a parameter in pyomo should be run only once
#1
model.purchasing_cost_i_j=pe.Param(model.number_of_suppliers,
                        model.number_of_plants,
                        model.number_of_raw_material,
                        model.number_of_time_periods,
                        initialize=Purchasing_cost_s_p,
                        default=100000)
#2
model.holding_cost_i_j=pe.Param(model.number_of_plants,
                        model.number_of_raw_material,
                        model.number_of_time_periods,
                        initialize=Holding_cost_s_p,
                        default=100000)

#3
model.distance_i_j=pe.Param(model.number_of_suppliers,
                        model.number_of_plants,
                        model.number_of_raw_material,
                        model.number_of_time_periods,
                        initialize=distance_s_p,
                        default=100000)
#4
model.FTC_i_j=pe.Param(model.number_of_suppliers,
                        model.number_of_plants,
                        model.number_of_raw_material,
                        model.number_of_time_periods,
                        initialize=FTC_s_p,
                        default=100000)
#5
model.VTC_i_j=pe.Param(model.number_of_suppliers,
                        model.number_of_plants,
                        model.number_of_raw_material,
                        model.number_of_time_periods,
                        initialize=VTC_s_p,
                        default=100000)
#6
model.Truckcap_i_j=pe.Param(model.number_of_suppliers,
                        model.number_of_plants,
                        model.number_of_raw_material,
                        model.number_of_time_periods,
                        initialize=Truck_capacity_s_p,
                        default=10)#check default value
#7
model.Vol_i_j=pe.Param(model.number_of_suppliers,
                        model.number_of_plants,
                        model.number_of_raw_material,
                        model.number_of_time_periods,
                        initialize=Volume_s_p,
                        default=100000)
#8
model.Manu_cost_j_n=pe.Param(model.number_of_plants,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=Manuf_p_prod,
                        default=100000)
```

```
#Creating a paramter master file between manufacturer and dcs
#product of j,k,n and t
d=list(product(number_of_plants,number_of_dcs,number_of_products,number_of_time_periods))

column_names = ['Manufacturer','DC', 'Product', 'Time Period']
df=pd.DataFrame(d,columns=column_names)
df.to_excel('Master_file_j_k_n_t.xlsx')




#Creating purchase cost
d=list(product(number_of_plants,number_of_dcs,number_of_products))
column_names=['Manufacturer','DC','Product']
df_temp=pd.DataFrame(d,columns=column_names)
df_temp.to_excel('PPurchase_cost_j_k_n.xlsx')





#Parameters to be added Pruchase cost, Holding cost, Transportation, Shortage cost(now not considered)
#To add distance we will join
df1=pd.read_excel('Master_file_j_k_n_t.xlsx')
df2=pd.read_excel('distance_j_k.xlsx')
df3=pd.read_excel('purchase_cost_j_k_n.xlsx')

df1=df1.merge(df2,how='left', left_on=['Manufacturer','DC'],right_on=['Plant','DC'])
df1=df1.merge(df3,how='left',left_on=['Manufacturer','DC','Product'],right_on=['Manufacturer','DC','Product'])
df1 = df1.drop(df1.columns[0], axis=1)
#df1.head()
#Defining fixed transportation cost as 20,000 per load
df1['FTC']=1000*a
#Variable transportation cost as 80/km
df1['VTC']=80*b
#Volume of each Q1_i_j_m_t 750ml
df1['Vol']=.75
#Truck Capacity 20cubm
df1['CT']=20000*c
#df1=df1.drop(df1.columns[5],axis=1)
df1['Holding_Cost']=df1['Purchase_Cost']*0.01*e
df1['Holding_Cost_manu']=df1['Purchase_Cost']*(0.015*e)





df1.head()





#1.Creating a dictionary for purchase cost
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
Purchasing_cost_p_dc= {key: value for key, value in df_temp['Purchase_Cost'].items()}

#2.Dict for Holding cost
```

```
col=list(df1.columns[[1,2,3]])
df_temp=df1.set_index(col)
Holding_cost_p_dc= {key: value for key, value in df_temp['Holding_Cost'].items()}

#2a. Dict for holding cost of finished product manufacturing
col=list(df1.columns[[0,2,3]])
df_temp=df1.set_index(col)
Holding_cost_p_final= {key: value for key, value in df_temp['Holding_Cost_manu'].items()}

#3.Dict for distance
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
distance_p_dc= {key: value for key, value in df_temp['Distance'].items()}

#4.Dict for FTC
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
FTC_p_dc= {key: value for key, value in df_temp['FTC'].items()}

#5.Dict for VC
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
VTC_p_dc= {key: value for key, value in df_temp['VTC'].items()}

#6.Dict for Truck capacity
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
Truck_capacity_p_dc= {key: value for key, value in df_temp['CT'].items()}

#6.Dict for Vol
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
Volume_p_dc= {key: value for key, value in df_temp['Vol'].items()}




#Creating a parameter in pyomo should be run only once
#1
model.purchasing_cost_j_k=pe.Param(model.number_of_plants,
                        model.number_of_dcs,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=Purchasing_cost_p_dc,
                        default=100000)
#2
model.holding_cost_j_k=pe.Param(
                        model.number_of_dcs,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=Holding_cost_p_dc,
                        default=100000)
#2a
model.holding_cost_j_p=pe.Param(model.number_of_plants,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=Holding_cost_p_final,
                        default=100000)
```

```
#3
model.distance_j_k=pe.Param(model.number_of_plants,
                        model.number_of_dcs,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=distance_p_dc,
                        default=100000)
#4
model.FTC_j_k=pe.Param(model.number_of_plants,
                        model.number_of_dcs,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=FTC_p_dc,
                        default=100000)
#5
model.VTC_j_k=pe.Param(model.number_of_plants,
                        model.number_of_dcs,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=VTC_p_dc,
                        default=100000)
#6
model.Truckcap_j_k=pe.Param(model.number_of_plants,
                        model.number_of_dcs,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=Truck_capacity_p_dc,
                        default=10)
#7
model.Vol_j_k=pe.Param(model.number_of_plants,
                        model.number_of_dcs,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=Volume_p_dc,
                        default=100000)




#Creating a paramter master file between dcs and retial
#product of j,k,n and t
d=list(product(number_of_dcs,number_of_retails,number_of_products,number_of_time_periods))

column_names = ['DC','Retail', 'Product', 'Time Period']
df=pd.DataFrame(d,columns=column_names)
df.to_excel('Master_file_k_l_n_t.xlsx')



#Creating purchase cost
d=list(product(number_of_dcs,number_of_retails,number_of_products))
column_names=['DC','Retail','Product']
df_temp=pd.DataFrame(d,columns=column_names)
df_temp.to_excel('PPurchase_cost_k_l_n.xlsx')



#Parameters to be added Pruchase cost, Holding cost, Transportation, Shortage cost(now not considered)
#To add distance we will join
df1=pd.read_excel('Master_file_k_l_n_t.xlsx')
```

```
df2=pd.read_excel('distance_k_l.xlsx')
df3=pd.read_excel('purchase_cost_k_l_n.xlsx')

df1=df1.merge(df2,how='left', left_on=['DC','Retail'],right_on=['DC','Retail'])
df1=df1.merge(df3,how='left',left_on=['DC','Retail'],right_on=['DC','Retail'])
df1 = df1.drop(df1.columns[0], axis=1)

#df1.head()
#Defining fixed transportation cost as 20,000 per load
df1['FTC']=1000*a
#Variable transportation cost as 80/km
df1['VTC']=80*b
#Volume of each Q1_i_j_m_t 750ml
df1['Vol']=.75
#Truck Capacity 20cubm
df1['CT']=20000*c
#df1=df1.drop(df1.columns[5],axis=1)
df1['Holding_Cost']=df1['Purchase_Cost']*(0.06*e)


#1.Creating a dictionary for purchase cost
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
Purchasing_cost_dc_ret= {key: value for key, value in df_temp['Purchase_Cost'].items()}

#2.Dict for Holding cost
col=list(df1.columns[[1,2,3]])
df_temp=df1.set_index(col)
Holding_cost_dc_ret= {key: value for key, value in df_temp['Holding_Cost'].items()}

#3.Dict for distance
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
distance_dc_ret= {key: value for key, value in df_temp['Distance'].items()}

#4.Dict for FTC
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
FTC_dc_ret= {key: value for key, value in df_temp['FTC'].items()}

#5.Dict for VC
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
VTC_dc_ret= {key: value for key, value in df_temp['VTC'].items()}

#6.Dict for Truck capacity
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
Truck_capacity_dc_ret= {key: value for key, value in df_temp['CT'].items()}

#6.Dict for Vol
col=list(df1.columns[[0,1,2,3]])
df_temp=df1.set_index(col)
Volume_dc_ret= {key: value for key, value in df_temp['Vol'].items()}



#Creating a parameter in pyomo should be run only once
#1
model.purchasing_cost_k_l=pe.Param(model.number_of_dcs,
                        model.number_of_retail,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=Purchasing_cost_dc_ret,
                        default=100000)
#2
model.holding_cost_k_l=pe.Param(model.number_of_retail,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=Holding_cost_dc_ret,
                        default=100000)
#3
model.distance_k_l=pe.Param(model.number_of_dcs,
                        model.number_of_retail,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=distance_dc_ret,
                        default=100000)
#4
model.FTC_k_l=pe.Param(model.number_of_dcs,
                        model.number_of_retail,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=FTC_dc_ret,
                        default=100000)
#5
model.VTC_k_l=pe.Param(model.number_of_dcs,
                        model.number_of_retail,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=VTC_dc_ret,
                        default=100000)
#6
model.Truckcap_k_l=pe.Param(model.number_of_dcs,
                        model.number_of_retail,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=Truck_capacity_dc_ret,
                        default=10)
#7
model.Vol_k_l=pe.Param(model.number_of_dcs,
                        model.number_of_retail,
                        model.number_of_products,
                        model.number_of_time_periods,
                        initialize=Volume_dc_ret,
                        default=100000)



#Decision variables
model.Manufactured_Q=pe.Var(number_of_plants,number_of_products,number_of_time_periods,domain=pe.N
onNegativeIntegers)
model.Q1=pe.Var(number_of_suppliers,number_of_plants,number_of_raw_material,number_of_time_periods,d
omain=pe.NonNegativeIntegers)
model.Q2=pe.Var(number_of_plants,number_of_dcs,number_of_products,number_of_time_periods,domain=pe
.NonNegativeIntegers)
model.Q3=pe.Var(number_of_dcs,number_of_retails,number_of_products,number_of_time_periods,domain=pe
.NonNegativeIntegers)

#Adding inventroy variable
```

```
model.Inv_j_n_t=pe.Var(model.number_of_plants,model.number_of_products,model.number_of_time_periods,
domain=pe.NonNegativeIntegers)
model.Inv_k_n_t=pe.Var(model.number_of_dcs,model.number_of_products,model.number_of_time_periods,do
main=pe.NonNegativeIntegers)
model.Inv_l_n_t=pe.Var(model.number_of_retail,model.number_of_products,model.number_of_time_periods,d
omain=pe.NonNegativeIntegers)
model.Inv_raw_j_m_t=pe.Var(model.number_of_plants,model.number_of_raw_material,model.number_of_tim
e_periods,domain=pe.NonNegativeIntegers)

#Decision variables for batch size
model.Batch_Q1=pe.Var(model.number_of_suppliers,model.number_of_raw_material,domain=pe.NonNegativ
eIntegers)
model.Batch_Q2=pe.Var(model.number_of_plants,model.number_of_products,domain=pe.NonNegativeInteger
s)
model.Batch_Q3=pe.Var(model.number_of_dcs,model.number_of_products,domain=pe.NonNegativeIntegers)

model.Manu_Batch=pe.Var(model.number_of_plants,model.number_of_products,domain=pe.NonNegativeInte
gers)



#Constraint 1
#Inventory balancing constraints
#1.retail
model.Inventory_bal_ret = pe.ConstraintList() #creates an empty list of constraints

for l in model.number_of_retail: #for each retail
    for n in model.number_of_products:#for each product
        for t in model.number_of_time_periods:# for each time period
            if t==1:
                lhs= sum(model.Q3[k,l,n,t] for k in model.number_of_dcs)-model.Demand_l[l,n,t]
                rhs=model.Inv_l_n_t[l,n,t]
                model.Inventory_bal_ret.add(lhs==rhs)
            else:
                lhs= model.Inv_l_n_t[l,n,t-1]+sum(model.Q3[k,l,n,t] for k in model.number_of_dcs)-
model.Demand_l[l,n,t]
                rhs=model.Inv_l_n_t[l,n,t]

                model.Inventory_bal_ret.add(lhs==rhs)
#2.DC
model.Inventory_bal_dc = pe.ConstraintList()#create an empty list of constraint

for k in model.number_of_dcs:
    for n in model.number_of_products:
        for t in model.number_of_time_periods:
            if t==1:
                lhs=sum(model.Q2[j,k,n,t] for j in model.number_of_plants)-sum(
                model.Q3[k,l,n,t] for l in model.number_of_retail)
                rhs= model.Inv_k_n_t[k,n,t]
                model.Inventory_bal_dc.add(lhs==rhs)
            else:
                lhs= model.Inv_k_n_t[k,n,t-1]+sum(model.Q2[j,k,n,t] for j in model.number_of_plants)-sum(
                model.Q3[k,l,n,t] for l in model.number_of_retail)
                rhs= model.Inv_k_n_t[k,n,t]
                model.Inventory_bal_dc.add(lhs==rhs)

#3. Manufacturer for finished product
model.Inventory_bal_p = pe.ConstraintList()
for j in model.number_of_plants:
    for t in model.number_of_time_periods:
        for n in model.number_of_products:

            if t==1:
                lhs= model.Manufactured_Q[j,n,t]-sum(model.Q2[j,k,n,t] for k in model.number_of_dcs)
                rhs= model.Inv_j_n_t[j,n,t]
                model.Inventory_bal_p.add(lhs==rhs)
            else:
                lhs=model.Manufactured_Q[j,n,t] +model.Inv_j_n_t[j,n,t-1]-sum(model.Q2[j,k,n,t] for k in
model.number_of_dcs)
                rhs= model.Inv_j_n_t[j,n,t]
                model.Inventory_bal_p.add(lhs==rhs)

#4. Manufacturer for raw material
model.Inventory_bal_p_raw = pe.ConstraintList()
for m in model.number_of_raw_material:
    #print(m)
    if m == 'raw_m1':
        for j in model.number_of_plants:
        #for m in model.number_of_raw_material:
            for t in model.number_of_time_periods:
                if t==1:
                    lhs= sum(model.Q1[i,j,m,t] for i in model.number_of_suppliers)-2*(model.Manufactured_Q[j,'p1',t]
)-(model.Manufactured_Q[j,'p2',t])
                    rhs= model.Inv_raw_j_m_t[j,m,t]
                    model.Inventory_bal_p_raw.add(lhs==rhs)
                else:
                    lhs=model.Inv_raw_j_m_t[j,m,t-1]+sum(model.Q1[i,j,m,t] for i in model.number_of_suppliers)-
2*(model.Manufactured_Q[j,'p1',t] )-(model.Manufactured_Q[j,'p2',t])
                    rhs= model.Inv_raw_j_m_t[j,m,t]
                    model.Inventory_bal_p_raw.add(lhs==rhs)
    else:
        for j in model.number_of_plants:
        #for m in model.number_of_raw_material:
            for t in model.number_of_time_periods:
                if t==1:
                    lhs= sum(model.Q1[i,j,m,t] for i in model.number_of_suppliers)-(model.Manufactured_Q[j,'p1',t])-
2*(model.Manufactured_Q[j,'p2',t])
                    model.Inventory_bal_p_raw.add(lhs==rhs)
                else:
                    lhs=model.Inv_raw_j_m_t[j,m,t-1]+sum(model.Q1[i,j,m,t] for i in model.number_of_suppliers)-
(model.Manufactured_Q[j,'p1',t])-2*(model.Manufactured_Q[j,'p2',t])
                    rhs= model.Inv_raw_j_m_t[j,m,t]
                    model.Inventory_bal_p_raw.add(lhs==rhs)



#Constraint 2
#Inventory capacity constraints to the model
#1.Plant
#Inventory of raw material and finished product must be less than capacity of
model.Capacity_constraint_p = pe.ConstraintList()
for j in model.number_of_plants:
    for t in model.number_of_time_periods:
        lhs = sum(model.Inv_j_n_t[j,n,t] for n in model.number_of_products)+sum(model.Inv_raw_j_m_t[j,m,t]
for m in model.number_of_raw_material)
        rhs = model.Capacity_j[j]
        model.Capacity_constraint_p.add(lhs <= rhs)


#2.DC
```

```
#Inventory of finished quantity at DC
model.Capacity_constraint_dc = pe.ConstraintList()
for k in model.number_of_dcs:
    for t in model.number_of_time_periods:
        lhs = sum(model.Inv_k_n_t[k,n,t] for n in model.number_of_products)
        rhs = model.Capacity_k[k]
        model.Capacity_constraint_dc.add(lhs <= rhs)

#3. Retail
#Inventory of finished quantity at retail should be less than the capacity of retail
model.Capacity_constraint_ret = pe.ConstraintList()
for l in model.number_of_retail:
    for t in model.number_of_time_periods:
        lhs = sum(model.Inv_l_n_t[l,n,t] for n in model.number_of_products)
        rhs = model.Capacity_l[l]
        model.Capacity_constraint_ret.add(lhs <= rhs)




#Constraint 4
#The amount of Q transferred forward in supply chain is less than or equal to previous inventory plus half of
Incoming goods into it
# At Manufacturer
#For finished product
model.transfer_Q2=pe.ConstraintList()

for j in model.number_of_plants:
    for t in model.number_of_time_periods:
        for n in model.number_of_products:
            if t==1:
                lhs=sum(model.Q2[j,k,n,t] for k in model.number_of_dcs)
                rhs=model.Manufactured_Q[j,n,t]
                model.transfer_Q2.add(lhs<=rhs)
            else:
                lhs=sum(model.Q2[j,k,n,t] for k in model.number_of_dcs)
                rhs=model.Inv_j_n_t[j,n,t-1]+model.Manufactured_Q[j,n,t]
                model.transfer_Q2.add(lhs<=rhs)

#For raw material
model.transfer_Q1_raw=pe.ConstraintList()

for j in model.number_of_plants:
    for t in model.number_of_time_periods:
        for n in model.number_of_products:
            for m in model.number_of_raw_material:
                if t==1:
                    lhs=model.Manufactured_Q[j,n,t]
                    rhs=sum(model.Q1[i,j,m,t] for i in model.number_of_suppliers)
                    model.transfer_Q1_raw.add(lhs<=rhs)
                else:
                    lhs=model.Manufactured_Q[j,n,t]
                    rhs=model.Inv_raw_j_m_t[j,m,t-1]+sum(model.Q1[i,j,m,t] for i in model.number_of_suppliers)
                    model.transfer_Q1_raw.add(lhs<=rhs)


#At DC
model.transfer_Q3=pe.ConstraintList()
```

```
for k in model.number_of_dcs:
    for t in model.number_of_time_periods:
        lhs=sum(model.Q3[k,l,n,t] for l in model.number_of_retail for n in model.number_of_products)
        rhs=model.Capacity_k[k]
        model.transfer_Q3.add(lhs<=rhs)




#Constraint 5
#Constraint for min batch size for Q1, Q2, Q3

model.batch_size_Q1=pe.ConstraintList()
for i in model.number_of_suppliers:
    for j in model.number_of_plants:
        for m in model.number_of_raw_material:
            for t in model.number_of_time_periods:
                lhs=model.Q1[i,j,m,t]
                rhs=100*model.Batch_Q1[i,m]
                model.batch_size_Q1.add(lhs>=rhs)

model.batch_size_Q2=pe.ConstraintList()

for j in model.number_of_plants:
    for n in model.number_of_products:
        for t in model.number_of_time_periods:
            for k in model.number_of_dcs:
                lhs=model.Q2[j,k,n,t]
                rhs=100*model.Batch_Q2[j,n]
                model.batch_size_Q2.add(lhs>=rhs)

model.batch_size_Q3 =pe.ConstraintList()

for k in model.number_of_dcs:
    for l in model.number_of_retail:
        for n in model.number_of_products:
            for t in model.number_of_time_periods:
                lhs=model.Q3[k,l,n,t]
                rhs=100*model.Batch_Q3[k,n]
                model.batch_size_Q3.add(lhs>=rhs)

model.batch_size_Manu =pe.ConstraintList()

for j in model.number_of_plants:
    for n in model.number_of_products:
        for t in model.number_of_time_periods:
            lhs=model.Manufactured_Q[j,n,t]
            rhs=1000
            model.batch_size_Manu.add(lhs>=rhs)



#Constraint 6 Raw Material
model.raw_material=pe.ConstraintList()

for i in model.number_of_suppliers:
    for j in model.number_of_plants:
        for m in model.number_of_raw_material:
            for t in model.number_of_time_periods:
                lhs=model.Q1[i,j,m,t]
            rhs=80000
            model.raw_material.add(lhs<=rhs)




#Constraint 7 Manufacturing capacity
model.manufacturing_capacity=pe.ConstraintList()

for j in model.number_of_plants:
    for t in model.number_of_time_periods:
        lhs=sum(model.Manufactured_Q[j,n,t] for n in model.number_of_products)
        rhs=120000
        model.manufacturing_capacity.add(lhs<=rhs)





#Constraint 8
#Constraint that all DC's are in business
model.dc_quantity=pe.ConstraintList()

for k in model.number_of_dcs:
    lhs=sum(model.Q2[j,k,n,t] for j in model.number_of_plants for n in model.number_of_products for t in
model.number_of_time_periods)
    rhs=model.Capacity_k[k]
    model.dc_quantity.add(lhs>=rhs)




#objective functions
def objective1(model):
    return (sum(model.purchasing_cost_i_j[i,j,m,t]*model.Q1[i,j,m,t]
            for i in model.number_of_suppliers
            for j in model.number_of_plants
            for m in model.number_of_raw_material
            for t in model.number_of_time_periods)+sum(model.holding_cost_i_j[j,m,t]*(model.Q1[i,j,m,t]*0.5)
            for j in model.number_of_plants
            for i in model.number_of_suppliers
            for m in model.number_of_raw_material
            for t in
model.number_of_time_periods)+sum(model.holding_cost_j_p[j,n,t]*(model.Manufactured_Q[j,n,t]*0.5)
            for j in model.number_of_plants
            for n in model.number_of_products
            for t in
model.number_of_time_periods)+sum(model.FTC_i_j[i,j,m,t]*(model.Q1[i,j,m,t]*model.Vol_i_j[i,j,m,t]/model.
Truckcap_i_j[i,j,m,t]) #whole number
            for i in model.number_of_suppliers
            for j in model.number_of_plants
            for m in model.number_of_raw_material
            for t in
model.number_of_time_periods)+sum(model.VTC_i_j[i,j,m,t]*model.distance_i_j[i,j,m,t]*(model.Q1[i,j,m,t]*
model.Vol_i_j[i,j,m,t]/model.Truckcap_i_j[i,j,m,t])
            for i in model.number_of_suppliers
            for j in model.number_of_plants
            for m in model.number_of_raw_material
            for t in
model.number_of_time_periods)+sum(model.Manufactured_Q[j,n,t]*model.Manu_cost_j_n[j,n,t]
            for j in model.number_of_plants
            for n in model.number_of_products
```

```
               for t in model.number_of_time_periods)
           )


def objective2(model):
    return (sum(model.purchasing_cost_j_k[j,k,n,t]*model.Q2[j,k,n,t]
               for j in model.number_of_plants
               for k in model.number_of_dcs
               for n in model.number_of_products
               for t in model.number_of_time_periods)+sum(model.holding_cost_j_k[k,n,t]*(model.Q2[j,k,n,t]*0.5)
               for k in model.number_of_dcs
               for j in model.number_of_plants
               for n in model.number_of_products
               for t in
model.number_of_time_periods)+sum(model.FTC_j_k[j,k,n,t]*(model.Q2[j,k,n,t]*model.Vol_j_k[j,k,n,t]/mode
l.Truckcap_j_k[j,k,n,t])
               for j in model.number_of_plants
               for k in model.number_of_dcs
               for n in model.number_of_products
               for t in
model.number_of_time_periods)+sum(model.VTC_j_k[j,k,n,t]*model.distance_j_k[j,k,n,t]*(model.Q2[j,k,n,t]*
model.Vol_j_k[j,k,n,t]/model.Truckcap_j_k[j,k,n,t])
               for j in model.number_of_plants
               for k in model.number_of_dcs
               for n in model.number_of_products
               for t in model.number_of_time_periods)
           )


def objective3(model):
    return (sum(model.purchasing_cost_k_l[k,l,n,t]*model.Q3[k,l,n,t]
               for k in model.number_of_dcs
               for l in model.number_of_retail
               for n in model.number_of_products
               for t in model.number_of_time_periods)+sum(model.holding_cost_k_l[l,n,t]*(model.Q3[k,l,n,t]*0.5)
               for l in model.number_of_retail
               for k in model.number_of_dcs
               for n in model.number_of_products
               for t in
model.number_of_time_periods)+sum(model.FTC_k_l[k,l,n,t]*(model.Q3[k,l,n,t]*model.Vol_k_l[k,l,n,t]/mode
l.Truckcap_k_l[k,l,n,t])
               for k in model.number_of_dcs
               for l in model.number_of_retail
               for n in model.number_of_products
               for t in
model.number_of_time_periods)+sum(model.VTC_k_l[k,l,n,t]*model.distance_k_l[k,l,n,t]*(model.Q3[k,l,n,t]*
model.Vol_k_l[k,l,n,t]/model.Truckcap_k_l[k,l,n,t])
               for k in model.number_of_dcs
               for l in model.number_of_retail
               for n in model.number_of_products
               for t in model.number_of_time_periods)
           )



model.obj1 = pe.Objective(rule=objective1, sense=pe.minimize)
model.obj2 = pe.Objective(rule=objective2, sense=pe.minimize)
model.obj3 = pe.Objective(rule=objective3, sense=pe.minimize)
```

```
def objective4(model):
    return(model.obj1+model.obj2+model.obj3)



model.obj4= pe.Objective(rule =objective4, sense=pe.minimize)




import os
solver = pe.SolverFactory('gurobi')
num_cores = os.cpu_count() or 1  # In case os.cpu_count() returns None, default to 1 core
solver.options['threads'] = num_cores
model.obj1.deactivate()
model.obj2.deactivate()
model.obj3.deactivate()
#model.obj4.deactivate()
results = solver.solve(model, tee = True)
# solves and updates instance
# print('x1 = ',round(value(model.x1),2))
# print('x2 = ',round(value(model.x2),2))
#model.pprint()
#methodology



#Printing decision variables Q1,Q2,Q3
data = []

data=[{'Supplier':i,'Manufacturer':j,'Raw_Material':m,'Time_period':t,'Q1': pe.value(model.Q1[i,j,m,t])}
     for i in model.number_of_suppliers
     for j in model.number_of_plants
     for m in model.number_of_raw_material
     for t in model.number_of_time_periods
     if pe.value(model.Q1[i,j,m,t])>0]
q1=pd.DataFrame(data)

data = [{'Manufacturer': j, 'DC': k, 'Product': n, 'Time_period': t, 'Q2': pe.value(model.Q2[j, k, n, t])}
        for j in model.number_of_plants
        for k in model.number_of_dcs
        for n in model.number_of_products
        for t in model.number_of_time_periods
        if pe.value(model.Q2[j, k, n, t]) > 0]
q2 = pd.DataFrame(data)

data = [{'DC':k,'Retail':l,'Product':n,'Time_period':t,'Q3':pe.value(model.Q3[k,l,n,t])}
        for k in model.number_of_dcs
        for l in model.number_of_retail
        for n in model.number_of_products
        for t in model.number_of_time_periods
        if pe.value(model.Q3[ k,l, n, t]) > 0]
q3=pd.DataFrame(data)

#Storing data in excel file
j=1
for i in (q1,q2,q3):
    df= i
    filename='Q'+str(j)+'.xlsx'
    df.to_excel(filename)
    j=j+1





#Printing inventory
data_inv=[]

data_inv=[{'Manufacturer':j,'Product':n,'Time_Period':t,'Inv':pe.value(model.Inv_j_n_t[j,n,t])}
         for j in model.number_of_plants
         for n in model.number_of_products
         for t in model.number_of_time_periods
         if pe.value(model.Inv_j_n_t[j,n,t])>0]
Inv_p=pd.DataFrame(data_inv)

data_inv=[{'DC':k,'Product':n,'Time_Period':t,'Inv':pe.value(model.Inv_k_n_t[k,n,t])}
         for k in model.number_of_dcs
         for n in model.number_of_products
         for t in model.number_of_time_periods
         if pe.value(model.Inv_k_n_t[k,n,t])>0]
Inv_dc=pd.DataFrame(data_inv)

data_inv=[{'Retail':l,'Product':n,'Time_Period':t,'Inv':pe.value(model.Inv_l_n_t[l,n,t])}
         for l in model.number_of_retail
         for n in model.number_of_products
         for t in model.number_of_time_periods
         if pe.value(model.Inv_l_n_t[l,n,t])>0]
Inv_ret=pd.DataFrame(data_inv)

#Storing inv data
Inv_p.to_excel('Inv_p.xlsx')
Inv_dc.to_excel('Inv_dc.xlsx')
Inv_ret.to_excel('Inv_ret.xlsx')



#Storing data in excel file
j=1
for i in (q1,q2,q3):
    df= i
    filename='Q'+str(j)+'.xlsx'
    df.to_excel(filename)
    j=j+1


#Value of Objective function 1
print('Total Cost of Manufacturer=',pe.value(model.obj1), '\nAverage cost per Manufacturer=',
pe.value(model.obj1)/3)
#Value of Objective function 2
print('Total Cost of DC=',pe.value(model.obj2),'\nAverage cost per DC=',pe.value(model.obj3)/5)
#Value of Objective function 3
print('Total Cost of Retail=',pe.value(model.obj3),'\nAverage cost per retail=',pe.value(model.obj3)/10)



#Total supply chain cost
value=pe.value(model.obj1)+pe.value(model.obj2)+pe.value(model.obj3)
print('Total cost=',value)
```

```
# Printing manufacturing quantities
# for j in model.number_of_plants:
#     for n in model.number_of_products:
#         for t in model.number_of_time_periods:
#             print("Decision variable value",j,n,t, pe.value(model.Manufactured_Q[j,n,t]))


import networkx as nx
import matplotlib.pyplot as plt
a = 1


for t in model.number_of_time_periods:
    edge_list = []
    for m in model.number_of_raw_material:
        for i in model.number_of_suppliers:
            for j in model.number_of_plants:
                #for t in model.number_of_time_periods:
                if pe.value(model.Q1[i,j,m,t])>0:
                    edge_list.append((i,j))
                    #edge_list_1.append((i,j,m,t))
                        #print("Decision variable value Q1",i,j,t, )
        #print(edge_list_1)
        #edge_list.append(edge_list_1)

    #edge_list_2 = []
    for n in model.number_of_products:
        if n=='p1':
            edge_list_p1 = []
            for j in model.number_of_plants:
                for k in model.number_of_dcs:
                    #for t in model.number_of_time_periods:
                    if pe.value(model.Q2[j,k,n,t])>0:
                        edge_list_p1.append((j,k))
                        edge_list_p1.extend(edge_list)
                        #edge_list_1.append((j,k,n,t))

            #print(edge_list_2)
            #edge_list.append(edge_list_2)

            #edge_list_3 = []
            for k in model.number_of_dcs:
                for l in model.number_of_retail:
                    #for t in model.number_of_time_periods:
                    if pe.value(model.Q3[k,l,n,t])>0:
                        edge_list_p1.append((k,l))
                        edge_list_p1.extend(edge_list)
                        #edge_list_1.append((k,l,n,t))
            #print(edge_list_3)
            #edge_list.append(edge_list_3)

            #print(edge_list_1)
            #print(edge_list_p1)

            print('Network Diagram for' + str(" ") + 'time period' +str(" ") + str(t) + str(" ") +'for product' + str(" ") + 
str(n))
            G = nx.from_edgelist(edge_list_p1)
            # Create a new figure and axis
            fig, ax = plt.subplots()

            # Draw the graph using a spring layout
            nx.draw_spring(G, with_labels=True, ax=ax)

            # Display the plot
            plt.show()
            filename = 'Network Diagram for' + str(" ")+ 'time period' +str(" ") + str(t) + str(" ") +'for product' + str("
") + str(n) +'.gexf'
            nx.write_gexf(G, filename)
            a+=1

        else:
            edge_list_p2 = []
            for j in model.number_of_plants:
                for k in model.number_of_dcs:
                    #for t in model.number_of_time_periods:
                    if pe.value(model.Q2[j,k,n,t])>0:
                        edge_list_p2.append((j,k))
                        edge_list_p2.extend(edge_list)
                        #edge_list_1.append((j,k,n,t))

            #print(edge_list_2)
            #edge_list.append(edge_list_2)

            #edge_list_3 = []
            for k in model.number_of_dcs:
                for l in model.number_of_retail:
                    #for t in model.number_of_time_periods:
                    if pe.value(model.Q3[k,l,n,t])>0:
                        edge_list_p2.append((k,l))
                        edge_list_p2.extend(edge_list)
                        #edge_list_1.append((k,l,n,t))
            #print(edge_list_3)
            #edge_list.append(edge_list_3)

            #print(edge_list_1)
            #print(edge_list_p2)

            print('Network Diagram for' + str(" ") + 'time period' +str(" ") + str(t) + str(" ") +'for product' + str(" ") +
str(n))
            G = nx.from_edgelist(edge_list_p2)


            # Create a new figure and axis
            fig, ax = plt.subplots()

            # Draw the graph using a spring layout
            nx.draw_spring(G, with_labels=True, ax=ax)

            # Display the plot
            plt.show()

            filename = 'Network Diagram for' +str(" ")+ 'time period' + str(" ") + str(t) + str(" ") +'for product' + str("
") + str(n) +'.gexf'
            nx.write_gexf(G, filename)
            a+=1
```

```
#Total transportation cost
sum(model.FTC_i_j[i,j,m,t]*pe.value(model.Q1[i,j,m,t])*model.Vol_i_j[i,j,m,t]/model.Truckcap_i_j[i,j,m,t]
            for i in model.number_of_suppliers
            for j in model.number_of_plants
            for m in model.number_of_raw_material
            for t in
model.number_of_time_periods)+sum(model.VTC_i_j[i,j,m,t]*model.distance_i_j[i,j,m,t]*pe.value(model.Q1[i
,j,m,t])*model.Vol_i_j[i,j,m,t]/model.Truckcap_i_j[i,j,m,t]
            for i in model.number_of_suppliers
            for j in model.number_of_plants
            for m in model.number_of_raw_material
            for t in
model.number_of_time_periods)+sum(model.FTC_j_k[j,k,n,t]*pe.value(model.Q2[j,k,n,t])*model.Vol_j_k[j,k,
n,t]/model.Truckcap_j_k[j,k,n,t]
            for j in model.number_of_plants
            for k in model.number_of_dcs
            for n in model.number_of_products
            for t in
model.number_of_time_periods)+sum(model.VTC_j_k[j,k,n,t]*model.distance_j_k[j,k,n,t]*pe.value(model.Q2[
j,k,n,t])*model.Vol_j_k[j,k,n,t]/model.Truckcap_j_k[j,k,n,t]
            for j in model.number_of_plants
            for k in model.number_of_dcs
            for n in model.number_of_products
            for t in
model.number_of_time_periods)+sum(model.FTC_k_l[k,l,n,t]*pe.value(model.Q3[k,l,n,t])*model.Vol_k_l[k,l,
n,t]/model.Truckcap_k_l[k,l,n,t]
            for k in model.number_of_dcs
            for l in model.number_of_retail
            for n in model.number_of_products
            for t in
model.number_of_time_periods)+sum(model.VTC_k_l[k,l,n,t]*model.distance_k_l[k,l,n,t]*pe.value(model.Q3[
k,l,n,t])*model.Vol_k_l[k,l,n,t]/model.Truckcap_k_l[k,l,n,t]
            for k in model.number_of_dcs
            for l in model.number_of_retail
            for n in model.number_of_products
            for t in model.number_of_time_periods)



#Dataframe for all Cost timeperiodwise
TC = []
i = 1
for t in model.number_of_time_periods:
    time = i
    TC.append({'Time Period':time,
            'Purchasing Cost':sum(model.purchasing_cost_i_j[i,j,m,t]*pe.value(model.Q1[i,j,m,t])
                for i in model.number_of_suppliers
                for j in model.number_of_plants
                for m in
model.number_of_raw_material)+sum(model.purchasing_cost_j_k[j,k,n,t]*pe.value(model.Q2[j,k,n,t])
                for j in model.number_of_plants
                for k in model.number_of_dcs
                for n in
model.number_of_products)+sum(model.purchasing_cost_k_l[k,l,n,t]*pe.value(model.Q3[k,l,n,t])
                for k in model.number_of_dcs
                for l in model.number_of_retail
                for n in model.number_of_products),
            'Manufacturing Cost':sum(pe.value(model.Manufactured_Q[j,n,t])*model.Manu_cost_j_n[j,n,t]
               for j in model.number_of_plants
               for n in model.number_of_products),
            'Fixed_Transportation
Cost':sum(model.FTC_i_j[i,j,m,t]*(pe.value(model.Q1[i,j,m,t])*model.Vol_i_j[i,j,m,t]/model.Truckcap_i_j[i,j,
m,t])
               for i in model.number_of_suppliers
               for j in model.number_of_plants
               for m in
model.number_of_raw_material)+sum(model.FTC_j_k[j,k,n,t]*(pe.value(model.Q2[j,k,n,t])*model.Vol_j_k[j,k,
n,t]/model.Truckcap_j_k[j,k,n,t])
               for j in model.number_of_plants
               for k in model.number_of_dcs
               for n in
model.number_of_products)+sum(model.FTC_k_l[k,l,n,t]*(pe.value(model.Q3[k,l,n,t])*model.Vol_k_l[k,l,n,t]/
model.Truckcap_k_l[k,l,n,t])
               for k in model.number_of_dcs
               for l in model.number_of_retail
               for n in model.number_of_products),
            'Variable_Transportaion
Cost':sum(model.VTC_i_j[i,j,m,t]*model.distance_i_j[i,j,m,t]*(pe.value(model.Q1[i,j,m,t])*model.Vol_i_j[i,j,m
,t]/model.Truckcap_i_j[i,j,m,t])
               for i in model.number_of_suppliers
               for j in model.number_of_plants
               for m in
model.number_of_raw_material)+sum(model.VTC_j_k[j,k,n,t]*model.distance_j_k[j,k,n,t]*(pe.value(model.Q2
[j,k,n,t])*model.Vol_j_k[j,k,n,t]/model.Truckcap_j_k[j,k,n,t])
               for j in model.number_of_plants
               for k in model.number_of_dcs
               for n in
model.number_of_products)+sum(model.VTC_k_l[k,l,n,t]*model.distance_k_l[k,l,n,t]*(pe.value(model.Q3[k,l,
n,t])*model.Vol_k_l[k,l,n,t]/model.Truckcap_k_l[k,l,n,t])
               for k in model.number_of_dcs
               for l in model.number_of_retail
               for n in model.number_of_products),
            'Holding Cost':sum(model.holding_cost_i_j[j,m,t]*pe.value(model.Q1[i,j,m,t]*0.5)
               for i in model.number_of_suppliers
               for j in model.number_of_plants
               for m in
model.number_of_raw_material)+sum(model.holding_cost_j_p[j,n,t]*pe.value(model.Manufactured_Q[j,n,t]*0.
5)
               for j in model.number_of_plants
               for n in
model.number_of_products)+sum(model.holding_cost_j_k[k,n,t]*pe.value(model.Q2[j,k,n,t]*0.5)
               for j in model.number_of_plants
               for k in model.number_of_dcs
               for n in
model.number_of_products)+sum(model.holding_cost_k_l[l,n,t]*pe.value(model.Q3[k,l,n,t]*0.5)
               for k in model.number_of_dcs
               for l in model.number_of_retail
               for n in model.number_of_products),
           'Manufacturing Cost':sum(pe.value(model.Manufactured_Q[j,n,t])*model.Manu_cost_j_n[j,n,t]
            for j in model.number_of_plants
            for n in model.number_of_products)})
   i = i+1

TCDF = pd.DataFrame(TC)
TCDF['Total Cost PeriodWise'] = TCDF['Purchasing Cost']+TCDF['Manufacturing
Cost']+TCDF['Fixed_Transportation Cost']+TCDF['Variable_Transportaion Cost']+TCDF['Holding Cost']
sum_row = TCDF.sum()
```

```
sum_df = pd.DataFrame([sum_row], columns=TCDF.columns)

# Concatenate the original DataFrame and the new DataFrame with the sum row
TCDF = pd.concat([TCDF, sum_df], ignore_index=True)
TCDF


TCDF.to_excel('Cost_output_matrix_FTC_Change10.xlsx', index = False)


TCDF['Fixed_Transportation Cost'].sum()

TCDF['Variable_Transportaion Cost'].sum()

TCDF['Holding Cost'].sum()
#Holding cost
sum(model.holding_cost_i_j[j,m,t]*pe.value(model.Q1[i,j,m,t]*0.5)
                for i in model.number_of_suppliers
                for j in model.number_of_plants
                for m in model.number_of_raw_material
                for t in
model.number_of_time_periods)+sum(model.holding_cost_j_p[j,n,t]*pe.value(model.Manufactured_Q[j,n,t]*0.
5)
                for j in model.number_of_plants
                for n in model.number_of_products
                for t in
model.number_of_time_periods)+sum(model.holding_cost_j_k[k,n,t]*pe.value(model.Q2[j,k,n,t]*0.5)
                for j in model.number_of_plants
                for k in model.number_of_dcs
                for n in model.number_of_products
                for t in
model.number_of_time_periods)+sum(model.holding_cost_k_l[l,n,t]*pe.value(model.Q3[k,l,n,t]*0.5)
                for k in model.number_of_dcs
                for l in model.number_of_retail
                for n in model.number_of_products
                for t in model.number_of_time_periods)





import random
import pandas as pd
import pyomo.environ as pe

# Create an empty list to store the results
results_list = []


array = []
for w1 in range(0,11,1):
    for w2 in range(0,11,1):
        w3 = 10-w1-w2
        if w3 >= 0:
            array.append({'w1':w1/10, 'w2':w2/10, 'w3':w3/10})
df = pd.DataFrame(array)


result = []
for i in range(66):
    w1 = df.loc[i,'w1']
    w2 = df.loc[i,'w2']
    w3 = df.loc[i,'w3']
    #print(w3)
    def objective4(model):
        return((float(w1))*model.obj1+(float(w2))*model.obj2+(float(w3))*model.obj3)
    if hasattr(model, 'obj4'):
        model.del_component(model.obj4)

    model.obj4 = pe.Objective(rule=objective4, sense=pe.minimize)

    solver = pe.SolverFactory('gurobi')

    # Enable Gurobi logging by setting the appropriate solver options
# You can adjust the log file name and output level as per your preference
# For example, here we set the log file to "gurobi_log.log" and output level to 1
# Output level 1 will produce a log file with more detailed information
    solver.options['LogFile'] = 'gurobi_log.log'
    solver.options['OutputFlag'] = 1


    model.obj1.deactivate()
    model.obj2.deactivate()
    model.obj3.deactivate()
    # model.obj4.deactivate()
    results = solver.solve(model, tee=False)

    value = pe.value(model.obj1) + pe.value(model.obj2) + pe.value(model.obj3)
    #print('Total cost=', value)

    # Store the results in the list
    result.append({'w1': w1, 'w2': w2, 'w3': w3,'Objective4 Value': value})

# Convert the results_list to a DataFrame
dff = pd.DataFrame(result)
dff
# Save the DataFrame to an Excel file
#dff.to_excel('results.xlsx', index=False)



dff.to_excel('resultsFTC10.xlsx', index=False)


dff['Serial No.'] = range(1, len(df) + 1)
dff


import matplotlib.pyplot as plt
plt.plot(dff['Serial No.'],dff['Objective4 Value'], marker='o', linestyle='-')
plt.xlabel('Iteration')
plt.ylabel('Total Supply Chain Cost')
plt.title('Cost Graph')
plt.show()


dff['Objective4 Value'].min()


dff['Objective4 Value'].max()
```

```
dff['Objective4 Value'].describe()

log_file_path = 'gurobi_log.log'

# Read the Gurobi log file
with open(log_file_path, 'r') as log_file:
    log_contents = log_file.read()

# Print the entire log file
print(log_contents)

try:
    with open(log_file_path, 'r') as log_file:
        log_contents = log_file.read()
        method_index = log_contents.find("Method: ")
        if method_index != -1:
            end_index = log_contents.find("\n", method_index)
            optimization_method = log_contents[method_index + len("Method: "):end_index]
            print("Gurobi used the following optimization method:", optimization_method)
        else:
            print("Optimization method information not found in the log file.")
except FileNotFoundError:
    print("Gurobi log file not found.")
```