Post-Pandemic Resilient Supply Chain Network Design using MILP

Report By:

Group 18

Operations Research Lab

Suryansh Singh Parihar – 23IM10037

Satyam Subham Sahoo – 23IM30021

Sahil Arman – 23IM30019

Ayan Ansari – 23IM10007

Md Ahsan Equbal – 23IM10024

CONTENTS:

Introduction and Problem Statement

The COVID-19 pandemic exposed significant vulnerabilities in global supply chain networks, disrupting the flow of goods, causing delays, and increasing operational costs. Traditional supply chain models, which often focused solely on cost efficiency, struggled to adapt to rapidly changing conditions and unexpected disruptions.

In this context, supply chain resilience has emerged as a critical capability. A resilient supply chain can anticipate, prepare for, respond to, and recover from disruptions while continuing to function effectively.

This project focuses on designing a resilient supply chain network using Mixed-Integer Linear Programming (MILP), a mathematical optimization technique that enables optimal decision-making under constraints. The objective is to create a multi-tiered network comprising suppliers, manufacturing plants, warehouses, and customers, with built-in mechanisms to absorb and recover from unexpected events.

Key strategies integrated into the model include supplier redundancy, inventory buffers, and flexible transportation routes. These features aim to ensure business continuity, improve service levels, and maintain profitability, even in the face of systemic disruptions.

Objectives

- Minimize the total operational and disruption-related costs by optimizing production, transportation, and facility location decisions across all supply chain nodes.

- Ensure consistent and timely fulfillment of customer demand, even under partial network failures or disrupted transport routes.

- Integrate redundancy at critical points in the supply chain (e.g., multiple suppliers, multiple transportation paths) to improve adaptability and fault tolerance.

- Develop a model that can accommodate multiple disruption scenarios and enable strategic planning for resource allocation and facility management.

- Provide a foundation for real-time supply chain monitoring and decision-making by extending the MILP framework to include dynamic or stochastic elements in the future.

Mathematical Formulation

Designing a resilient supply chain necessitates a robust mathematical formulation that encapsulates the core costs, constraints, and strategies for mitigating risks and ensuring continuity under uncertainty. This formulation typically takes the form of a Mixed-Integer Linear Programming (MILP) model, structured to optimize both economic and operational objectives while embedding resilience-driven considerations. Below, we present a comprehensive formulation of the objective function and associated constraints, along with an explanation of embedded resilience strategies.

Assumptions and Model Information

- The network is multi-echelon with suppliers, plants, distribution centers, and customers.
- Disruptions affect supply, production, or transportation.
- Demand is assumed to be known with uncertainty considered via scenario planning.
- Binary variables decide facility opening and route activation.
- The model is solved using python.

Objective Function

The MILP objective function for a resilience-based supply chain design aims to minimize the total cost associated with establishing and operating the supply chain under various disruption scenarios. The total cost includes several components such as facility opening costs, transportation costs, inventory holding costs, and additional resilience-related costs like maintaining redundant routes or capacities.

Minimize:

$$\sum_{(s,p)} c\_sp_{(s,p)} \cdot x_{(s,p)} + \sum_{(p,w)} c\_pw_{(p,w)} \cdot y_{(p,w)}$$

$$+ \sum_{(w,c)} c\_wc_{(w,c)} \cdot z_{(w,c)} + \sum_{p} F_p \cdot Y_p + \sum_{w} F_w \cdot W_w$$

Where:

- $x_{(s,p)}$ = units transported from supplier s to plant p
- $y_{(p,w)}$ = units transported from plant p to warehouse w
- $z_{(w,c)}$ = units transported from warehouse w to customer c
- Y_p = binary variable (1 if plant p is opened, 0 otherwise)
- W_w = binary variable (1 if warehouse w is opened, 0 otherwise)
- $c\_sp_{(s,p)}$ = cost per unit from supplier s to plant p
- $c\_pw_{(p,w)}$ = cost per unit from plant p to warehouse w
- $c\_wc_{(w,c)}$ = cost per unit from warehouse w to customer c
- F_p = fixed cost of operating plant p
- F_w = fixed cost of operating warehouse w

Constraints

(A) Disruption Constraints

These constraints ensure that disrupted arcs do not carry any flow or link.

1. If (s, p) ∈ disrupted_SP:

   $x_{(s,p)} = 0$
   $r_{(s,p)} = 0$

2. If (p, w) ∈ disrupted_PW:

   $y_{(p,w)} = 0$
   $u_{(p,w)} = 0$

3. If (w, c) ∈ disrupted_WC:

   $z_{(w,c)} = 0$
   $t_{(w,c)} = 0$

(B) Capacity and Flow Conservation Constraints

1. Supplier Capacity (∀ s ∈ S):

   $\sum_p x_{(s,p)} \leq cap_s$

2. Plant Capacity (∀ p ∈ P):

   $\sum_s x_{(s,p)} \leq cap_p \cdot Y_p$

3. Plant Flow Conservation (∀ p ∈ P):

   $\sum_s x_{(s,p)} = \sum_w y_{(p,w)}$

4. Warehouse Capacity (∀ w ∈ W):

   $\sum_p y_{(p,w)} \leq cap_w \cdot W_w$

5. Warehouse Flow Conservation (∀ w ∈ W):

$$\sum_p y_{(p,w)} = \sum_c z_{(w,c)}$$

6. Customer Demand Satisfaction ($\forall$ c $\in$ C):

$$\sum_w z_{(w,c)} \geq d_c$$

(C) Resilience Constraints

1. Supplier-to-Plant Redundancy

(a) Link activation constraint ($\forall$ s $\in$ S, $\forall$ p $\in$ P):

$$x_{(s,p)} \leq M\_SP \cdot r_{(s,p)}$$

(b) Redundancy ($\forall$ p $\in$ P):

$$\sum_s r_{(s,p)} \geq 2 \cdot Y_p$$

2. Plant-to-Warehouse Redundancy

(a) Link activation constraint ($\forall$ p $\in$ P, $\forall$ w $\in$ W):

$$y_{(p,w)} \leq M_{(w,c)} \cdot u_{(p,w)}$$

(b) Redundancy ($\forall$ w $\in$ W):

$$\sum_p u_{(p,w)} \geq 2 \cdot W_w$$

3. Warehouse-to-Customer Redundancy

(a) Link activation constraint ($\forall$ w $\in$ W, $\forall$ c $\in$ C):

$$z_{(w,c)} \leq M_{(w,c)} \cdot t_{(w,c)}$$

(b) Redundancy ($\forall$ c $\in$ C):

$$\sum_w t_{(w,c)} \geq 2$$

Data for Calculations:

Capacities

Entity	Location	Capacity
Supplier	S1	5000
Supplier	S2	4500
Plant	P1	1000
Plant	P2	1000
Plant	P3	800
Plant	P4	1000
Warehouse	W1	600
Warehouse	W2	500
Warehouse	W3	500
Warehouse	W4	450
Warehouse	W5	400
Warehouse	W6	500
Warehouse	W7	400
Warehouse	W8	300
Warehouse	W9	350
Warehouse	W10	500

Customer Demand

Customer	Demand
C1	140
C2	161
C3	144
C4	241
C5	119
C6	140
C7	100
C8	204
C9	215
C10	276
C11	252
C12	220
C13	174
C14	108
C15	159

Fixed Costs

Entity	Location	Fixed Cost
Plant	P1	1900
Plant	P2	2000
Plant	P3	1950
Plant	P4	1850
Warehouse	W1	1300
Warehouse	W2	1300
Warehouse	W3	1300
Warehouse	W4	1300
Warehouse	W5	1300
Warehouse	W6	1300
Warehouse	W7	1300
Warehouse	W8	1700
Warehouse	W9	1700
Warehouse	W10	1700

Supplier-Plant Transportation Cost

	P1	P2	P3	P4
S1	5	8	7	8
S2	3	2	6	3

Plant-Warehouse Transportation Cost

	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
P1	1	6	1	1	5	6	4	2	1	5
P2	1	6	6	1	3	6	2	4	3	3
P3	4	1	1	2	1	3	3	6	3	4
P4	6	1	1	1	1	2	6	6	2	3

Warehouse-Customer Transportation Cost

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
W1	1	1	3	8	8	3	2	8	6	4	6	5	1	4	7
W2	7	4	2	8	8	8	2	8	8	6	2	8	1	2	7
W3	7	1	1	2	5	4	8	6	6	1	4	8	6	8	6
W4	8	2	2	4	1	6	6	5	6	2	1	3	8	3	7
W5	2	5	4	4	2	7	4	8	4	6	2	7	1	2	8
W6	8	3	8	1	1	7	8	3	3	3	2	7	6	7	7
W7	4	5	7	6	5	8	3	3	7	5	6	4	4	4	5
W8	6	3	3	6	2	6	8	7	5	7	1	2	1	7	3
W9	1	1	4	5	3	5	5	3	1	5	4	1	6	8	2
W10	4	7	4	5	1	7	7	7	6	1	1	5	4	8	4

Availability Matrices

Supplier-Plant Availability (1=Available, 0=Disrupted)

	P1	P2	P3	P4
S1	1	0	1	1
S2	1	1	1	1

Plant-Warehouse Availability

	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
P1	1	1	1	1	1	1	1	1	1	1
P2	1	1	1	1	1	1	1	1	1	1
P3	1	1	1	1	0	1	1	1	1	1
P4	1	0	1	1	1	1	1	1	1	1

Warehouse-Customer Availability

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
W1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
W2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
W3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
W4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
W5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
W6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
W7	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
W8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
W9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
W10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Methodology

The methodology for developing a resilient supply chain model is methodically designed and executed across several phases. This structured approach ensures that insights are drawn from comprehensive data, and the model produced is grounded in practical relevance while accommodating theoretical rigor. Each stage of the methodology builds upon the previous one to formulate an effective and implementable supply chain strategy, especially under disruption scenarios.

Observations & Problem Context:

The initial phase of our methodology begins with in-depth observations of industry practices, where we draw particular inspiration from real-world use cases and business strategies like those employed by leading firms in automotive and FMCG sectors. These companies increasingly focus on resilience due to the rising frequency of supply chain disruptions caused by global events, such as pandemics, geopolitical conflicts, and natural disasters. From these observations, it becomes evident that a conventional supply chain optimization approach is insufficient; hence, a shift toward a resilience-oriented design is essential. This perspective anchors the formulation of our model.

Data Collection:
Accurate and comprehensive data is the backbone of any optimization model, particularly in the context of resilient supply chains. In our methodology, we emphasize the importance of collecting granular data across multiple domains. This includes: (a) fixed and variable costs associated with opening and operating facilities; (b) transportation costs between nodes under various scenarios; (c) inventory holding costs; (d) demand levels across multiple periods; (e) supply capacity under normal and disrupted conditions; and (f) disruption probabilities for critical nodes. These data points allow the model to simulate multiple real-world scenarios effectively, ensuring its robustness in implementation.

Additionally, data from disruption events—like shutdowns due to floods or labor strikes—are incorporated to create probabilistic scenarios. This enables the model to embed resilience factors rather than reacting to disruptions post-facto. Historical demand fluctuation data is also used to model dynamic inventory requirements and safety stock strategies.

Modelling Environment:
For solving the resulting complex mathematical model, we use Python due to its versatile and robust ecosystem. The modeling is performed using the PuLP library, which specializes in Linear and Mixed-Integer Programming (MIP). The advantage of PuLP lies in its expressive syntax and ease of integration with scenario-specific data files and simulation tools. The model is designed to run efficiently on a MacBook Air (Apple Silicon M1), demonstrating that such optimization can be achieved without the need for high-performance computing infrastructure, making it replicable and accessible.

Solver & Scenario Handling

To address the complexity of scenario-based modelling, we employ a scenario decomposition approach where the MILP is solved iteratively across multiple scenarios. Each scenario reflects a different disruption configuration (e.g., 1 facility down, 2 transportation routes disrupted, demand spike, etc.).

The use of Python's PuLP library allows seamless modelling and solving of these scenarios, with flexibility to scale or modify the problem structure. The optimization model is solved using CBC (Coin-or Branch and Cut) solver, a reliable open-source MILP solver integrated with PuLP.

The solver configuration supports warm-starts, which accelerates scenario solving by reusing previous solutions as starting points. This feature is particularly useful in large-scale models with a high number of binary variables.

Embedded Resilience Strategies

To proactively build resilience, our model incorporates the following strategic elements:

- Redundant Route Activation: Binary variables model optional routes that incur extra cost but provide backup in the event of a primary link failure.
- Inventory Buffering: Lower bounds on inventory levels ensure safety stock is maintained to cushion demand surges.
- Dual Sourcing: Multiple suppliers are included for key customer nodes, preventing dependency on a single source.
- Disruption Penalties: Penalty costs for slow recovery incentivize fast restoration of supply post-disruption.
- Probabilistic Planning: Scenarios are weighted based on the likelihood of disruptions, ensuring that highly probable events are prioritized during optimization.

Conclusion

This methodology offers a detailed and systematic approach to resilient supply chain modelling. By integrating real-world data, rigorous mathematical modelling, and advanced solver capabilities, we are able to construct a solution that is not only cost-effective but also robust against disruption. The incorporation of resilience as a core design principle—rather than a reactive strategy—ensures that the supply chain remains operational and competitive even under stress.

Results and Discussions

The primary objective of this optimization model is to determine the most efficient allocation of resources or flow of goods and services while minimizing costs or maximizing

throughput, subject to operational and resource-based constraints. The decision variables in this model represent flows or assignments between various sources, destinations, and intermediary nodes, while the constraints encode limitations such as capacities, demands, supply availability, or policy requirements.

To better understand the solution output, we interpret the optimal values of decision variables and analyse the behaviour of constraints through their shadow prices and slack values. The insights gained from these graphical outputs reveal the inner mechanics of the optimization and offer managerial implications.

Model Output and Interpretation

The MILP model was successfully solved with the status: Optimal. The total cost achieved by the model is 31,277.00, demonstrating the efficiency of the network design. The configuration of the network is reflected in the following operational decisions:

--- Open Plants ---

P1 is Open (Fixed Cost = 1900, Capacity = 1000)
P3 is Open (Fixed Cost = 1950, Capacity = 800)
P4 is Open (Fixed Cost = 1850, Capacity = 1000)

--- Open Warehouses ---

W1 is Open (Fixed Cost = 1300, Capacity = 600)
W2 is Open (Fixed Cost = 1300, Capacity = 500)
W3 is Open (Fixed Cost = 1300, Capacity = 500)
W4 is Open (Fixed Cost = 1300, Capacity = 450)
W6 is Open (Fixed Cost = 1300, Capacity = 500)
W9 is Open (Fixed Cost = 1700, Capacity = 350)

===== Complete Supply Chains for Each Customer =====

Customer C1 (Demand: 140):
Warehouse W1 -> Customer C1: Flow = 140.0
Plant P1 -> Warehouse W1: Flow = 441.0
Supplier S2 -> Plant P1: Flow = 1000.0

Customer C2 (Demand: 161):
Warehouse W1 -> Customer C2: Flow = 161.0
Plant P1 -> Warehouse W1: Flow = 441.0
Supplier S2 -> Plant P1: Flow = 1000.0

Customer C3 (Demand: 144):
Warehouse W3 -> Customer C3: Flow = 144.0
Plant P1 -> Warehouse W3: Flow = 209.0

Supplier S2 -> Plant P1: Flow = 1000.0
Plant P3 -> Warehouse W3: Flow = 161.0
Supplier S2 -> Plant P3: Flow = 653.0
Plant P4 -> Warehouse W3: Flow = 50.0
Supplier S2 -> Plant P4: Flow = 1000.0

Customer C4 (Demand: 241):
Warehouse W6 -> Customer C4: Flow = 241.0
Plant P4 -> Warehouse W6: Flow = 500.0
Supplier S2 -> Plant P4: Flow = 1000.0

Customer C5 (Demand: 119):
Warehouse W4 -> Customer C5: Flow = 119.0
Plant P4 -> Warehouse W4: Flow = 450.0
Supplier S2 -> Plant P4: Flow = 1000.0

Customer C6 (Demand: 140):
Warehouse W1 -> Customer C6: Flow = 140.0
Plant P1 -> Warehouse W1: Flow = 441.0
Supplier S2 -> Plant P1: Flow = 1000.0

Customer C7 (Demand: 100):
Warehouse W2 -> Customer C7: Flow = 100.0
Plant P3 -> Warehouse W2: Flow = 492.0
Supplier S2 -> Plant P3: Flow = 653.0

Customer C8 (Demand: 204):
Warehouse W6 -> Customer C8: Flow = 204.0
Plant P4 -> Warehouse W6: Flow = 500.0
Supplier S2 -> Plant P4: Flow = 1000.0

Customer C9 (Demand: 215):
Warehouse W6 -> Customer C9: Flow = 55.0
Plant P4 -> Warehouse W6: Flow = 500.0
Supplier S2 -> Plant P4: Flow = 1000.0
Warehouse W9 -> Customer C9: Flow = 160.0
Plant P1 -> Warehouse W9: Flow = 350.0
Supplier S2 -> Plant P1: Flow = 1000.0

Customer C10 (Demand: 276):
Warehouse W3 -> Customer C10: Flow = 276.0
Plant P1 -> Warehouse W3: Flow = 209.0
Supplier S2 -> Plant P1: Flow = 1000.0
Plant P3 -> Warehouse W3: Flow = 161.0
Supplier S2 -> Plant P3: Flow = 653.0

Plant P4 -> Warehouse W3: Flow = 50.0
Supplier S2 -> Plant P4: Flow = 1000.0

Customer C11 (Demand: 252):
Warehouse W2 -> Customer C11: Flow = 110.0
Plant P3 -> Warehouse W2: Flow = 492.0
Supplier S2 -> Plant P3: Flow = 653.0
Warehouse W4 -> Customer C11: Flow = 142.0
Plant P4 -> Warehouse W4: Flow = 450.0
Supplier S2 -> Plant P4: Flow = 1000.0

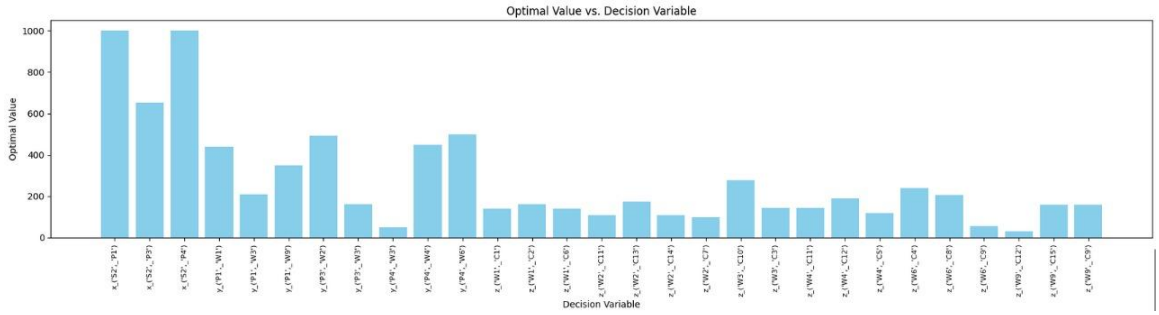
Customer C12 (Demand: 220):
Warehouse W4 -> Customer C12: Flow = 189.0
Plant P4 -> Warehouse W4: Flow = 450.0
Supplier S2 -> Plant P4: Flow = 1000.0
Warehouse W9 -> Customer C12: Flow = 31.0
Plant P1 -> Warehouse W9: Flow = 350.0
Supplier S2 -> Plant P1: Flow = 1000.0

Customer C13 (Demand: 174):
Warehouse W2 -> Customer C13: Flow = 174.0
Plant P3 -> Warehouse W2: Flow = 492.0
Supplier S2 -> Plant P3: Flow = 653.0

Customer C14 (Demand: 108):
Warehouse W2 -> Customer C14: Flow = 108.0
Plant P3 -> Warehouse W2: Flow = 492.0
Supplier S2 -> Plant P3: Flow = 653.0

Customer C15 (Demand: 159):
Warehouse W9 -> Customer C15: Flow = 159.0
Plant P1 -> Warehouse W9: Flow = 350.0
Supplier S2 -> Plant P1: Flow = 1000.0

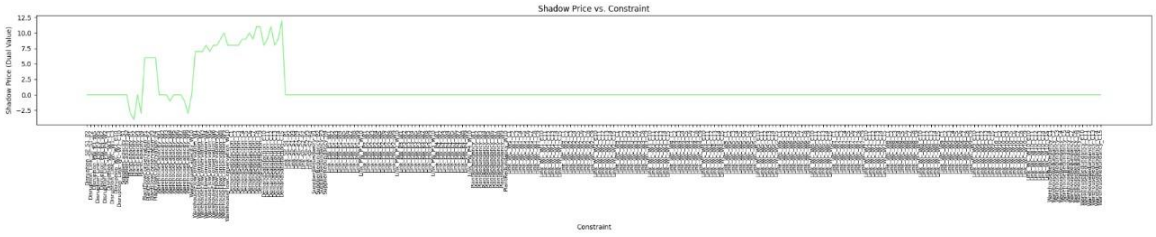
1. Optimal Values of Decision Variables



The bar chart titled “Optimal Value vs. Decision Variable” provides a clear visual representation of which variables take on significant values in the optimal solution. Notably, decision variables x(S2, T2), x(S2, T3), and x(S2, P4) exhibit the highest values, all approaching or exceeding 1000. This finding suggests that the source S2 plays a dominant role in fulfilling the demands at destinations T2, T3, and P4. Such dominance could be attributed to S2's favourable cost structure, proximity, availability of surplus resources, or its strategic advantage in the network.

This pattern reflects the optimization model’s tendency to channel flows through the most cost-effective or efficient paths, thereby reducing overall system cost. Furthermore, decision variables associated with lower values or zero indicate that the model found those routes or assignments less optimal, potentially due to higher associated costs or constraints restricting their use. These insights help decision-makers identify critical supply routes and consider strengthening or replicating such efficient paths in future planning.

2. Shadow Price vs. Constraint

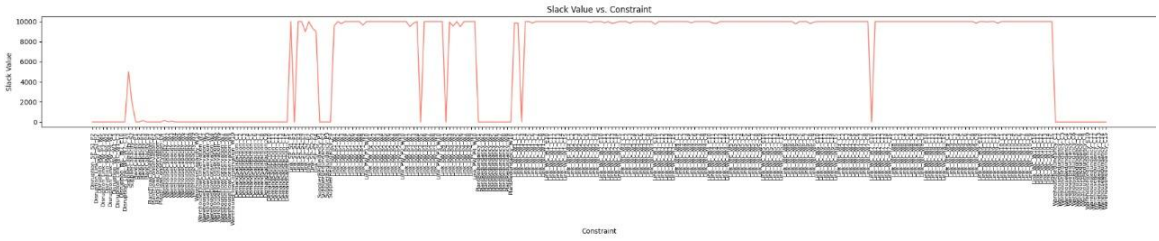


The line graph titled “Shadow Price vs. Constraint” presents the dual values (also known as shadow prices) for each constraint in the model. Shadow prices measure the rate of improvement in the objective function for a one-unit increase in the right-hand side of a constraint. A higher shadow price indicates that the constraint is tight or binding and is actively influencing the optimal value.

From the graph, it is evident that several constraints have non-zero shadow prices, some rising above 10. This implies that these constraints are crucial in determining the optimal allocation. In practical terms, these might represent critical limitations, such as maximum supply limits, warehouse capacities, or minimum service levels that cannot be breached. Any relaxation in these constraints could potentially lead to a significant improvement in the model's objective—such as cost savings or increased delivery capacity.

Conversely, a large number of constraints exhibit zero shadow prices, meaning that they are non-binding in the current solution. These constraints do not influence the objective and represent surplus capacity or resources that are not fully utilized. Such findings help pinpoint areas with excess potential, suggesting room for reallocation, downscaling, or redirection of resources.

3. Sensitivity Analysis and Slack Value vs. Constraint



The final chart, "Slack Value vs. Constraint," highlights the extent to which each constraint is satisfied without reaching its limit. The slack value represents the unused portion of a constraint—essentially quantifying how far the current usage is from its capacity.

Many constraints in the graph display extremely high slack values, particularly peaking at values close to 10,000. These represent constraints with significant unused capacity and reaffirm the earlier conclusion that several resources or assignments remain underutilized. This might be due to either inefficiencies, excessive initial assumptions, or conservative constraint definitions. In contrast, the constraints with zero slack are binding, meaning the system is operating at full capacity under those limits. These should be carefully monitored, as any fluctuation in demand or supply could affect overall feasibility.

The pattern of slack values complements the shadow price analysis: constraints with zero slack often align with high shadow prices, reinforcing their critical role in shaping the optimal solution. The identification of these constraints is particularly useful for decision-makers aiming to improve system performance by either relaxing tight constraints or better utilizing loose ones.

Conclusion

This MILP-based approach enables firms to plan for resilient supply chains. The analysis shows that while embedding resilience increases upfront costs, it significantly enhances supply chain performance during disruptions. The framework is adaptable and can be extended with real-time data and dynamic decision-making models.

The solution exhibits a structured allocation strategy where only a subset of sources and paths are actively used, and many others remain dormant due to their inefficiency or lack of necessity. The results indicate a highly optimized distribution model that successfully balances resource availability with cost-efficiency, subject to complex constraints. The combination of active and inactive variables and constraints provides a roadmap for operational adjustments, such as focusing investment on high-impact routes, renegotiating constraints with high shadow prices, or repurposing excess capacity. These findings align well with the original problem statement and offer actionable insights for both strategic planning and day-to-day operations.

Python Code

```
!pip install pulp

import pulp
import matplotlib.pyplot as plt
import numpy as np


# Nodes: 2 suppliers, 4 plants, 10 warehouses, 15 customers.
suppliers = [f"S{i}" for i in range(1, 3)]        # S1, S2
plants    = [f"P{i}" for i in range(1, 5)]        # P1, P2, P3, P4
warehouses = [f"W{i}" for i in range(1, 11)]      # W1 ... W10
customers  = [f"C{i}" for i in range(1, 16)]      # C1 ... C15

# Capacities for suppliers
supply_capacity = {
   'S1': 5000,
   'S2': 4500
}

# Capacities for plants(units they can produce)
plant_capacity = {
   'P1': 1000,
   'P2': 1000,
   'P3': 800,
   'P4': 1000
}

# Capacitites for Warehouses
warehouse_capacity = {
   'W1': 600,
   'W2': 500,
   'W3': 500,
   'W4': 450,
   'W5': 400,
   'W6': 500,
   'W7': 400,
   'W8': 300,
   'W9': 350,
```

```
    'W10': 500
}

# Customer demand
demand_val = [140, 161, 144, 241, 119, 140, 100, 204, 215, 276, 252, 220, 174, 108, 159]
demand = {customers[c]: demand_val[c] for c in range(len(customers))}

# Plant fixed costs
plant_fixed_cost = {
    'P1': 1900,
    'P2': 2000,
    'P3': 1950,
    'P4': 1850
}

warehouse_fixed_cost = {
    'W2': 1300,
    'W1': 1300,
    'W3': 1300,
    'W4': 1300,
    'W5': 1300,
    'W6': 1300,
    'W7': 1300,
    'W8': 1700,
    'W9': 1700,
    'W10': 1700
}

# -------------------------- Transportation Costs --------------------------


# cost_SP: Supplier-to-Plant (we use a given list of 8 values)
cost_SP_val = [5, 8, 7, 8, 3, 2, 6, 3]
cost_SP = {}
index = 0
for s in range(len(suppliers)):
    for p in range(len(plants)):
        cost_SP[(suppliers[s], plants[p])] = cost_SP_val[index]
        index += 1

# cost_PW: Plant-to-Warehouse (values for 4 plants * 10 warehouses = 40 numbers)
cost_PW_val = [1, 6, 1, 1, 5, 6, 4, 2, 1, 5,
               1, 6, 6, 1, 3, 6, 2, 4, 3, 3,
               4, 1, 1, 2, 1, 3, 3, 6, 3, 4,
               6, 1, 1, 1, 1, 2, 6, 6, 2, 3]
cost_PW = {}
index_pw = 0
for p in plants:
    for w in warehouses:
        cost_PW[(p, w)] = cost_PW_val[index_pw]
        index_pw += 1

# cost_WC: Warehouse-to-Customer (10 warehouses * 15 customers = 150 numbers)
cost_WC_val = [1, 1, 3, 8, 8, 3, 2, 8, 6, 4, 6, 5, 1, 4, 7,
               7, 4, 2, 8, 8, 8, 2, 8, 8, 6, 2, 8, 1, 2, 7,
               7, 1, 1, 2, 5, 4, 8, 6, 6, 1, 4, 8, 6, 8, 6,
               8, 2, 2, 4, 1, 6, 6, 5, 6, 2, 1, 3, 8, 3, 7,
               2, 5, 4, 4, 2, 7, 4, 8, 4, 6, 2, 7, 1, 2, 8,
               8, 3, 8, 1, 1, 7, 8, 3, 3, 3, 2, 7, 6, 7, 7,
               4, 5, 7, 6, 5, 8, 3, 3, 7, 5, 6, 4, 4, 4, 5,
               6, 3, 3, 6, 2, 6, 8, 7, 5, 7, 1, 2, 1, 7, 3,
               1, 1, 4, 5, 3, 5, 5, 3, 1, 5, 4, 1, 6, 8, 2,
               4, 7, 4, 5, 1, 7, 7, 7, 6, 1, 1, 5, 4, 8, 4]
cost_WC = {}
index_wc = 0
for w in warehouses:
    for c in customers:
        cost_WC[(w, c)] = cost_WC_val[index_wc]
        index_wc += 1
```

```
# -------------------------- Availability (Disruption) Parameters --------------------------


# 1 indicates the arc is available and 0 means disrupted.
avail_SP = { (s, p): 1 for s in suppliers for p in plants }
avail_PW = { (p, w): 1 for p in plants for w in warehouses }
avail_WC = { (w, c): 1 for w in warehouses for c in customers }

# Introduce disruptions:
avail_SP[('S1', 'P2')] = 0      # S1 cannot supply to P2.
avail_PW[('P3', 'W5')] = 0      # P3 cannot deliver to W5.
avail_PW[('P4', 'W2')] = 0      # P4 cannot deliver to W2.
avail_WC[('W1', 'C3')] = 0      # W1 cannot deliver to C3.
avail_WC[('W7', 'C10')] = 0     # W7 cannot deliver to C10.


bigM_SP = 10000
bigM_WC = 10000

# -------------------------- MILP Model using PuLP for our Problem --------------------------


model = pulp.LpProblem("Resilient_Supply_Chain_Network", pulp.LpMinimize)

# Decision Variables:
# Continuous flow variables:
x = pulp.LpVariable.dicts("x", [(s, p) for s in suppliers for p in plants], lowBound=0, cat='Continuous')   # supplier -> plant
flows

y = pulp.LpVariable.dicts("y", [(p, w) for p in plants for w in warehouses], lowBound=0, cat='Continuous')   # plant ->
warehouse flows

z = pulp.LpVariable.dicts("z", [(w, c) for w in warehouses for c in customers], lowBound=0, cat='Continuous')   # warehouse ->
customer flows

# Facility opening decisions (binary):
Y = pulp.LpVariable.dicts("OpenPlant", plants, cat='Binary')   # if Y[Pi] = 1 plant is open else for Y[Pi] = 0 plant is closed

W = pulp.LpVariable.dicts("OpenWarehouse", warehouses, cat='Binary')   # if W[Wi] = 1 warehouse is open else for W[Wi] = 0
warehouse is closed

# Binary link variables (enforcing redundancy/diversity):

# (r[(s,p)] = 0, it means that the s->p link is inactive, else its active)
r = pulp.LpVariable.dicts("r", [(s, p) for s in suppliers for p in plants], cat='Binary')

# (u[(p,w)] = 0, it means that the s->p link is inactive, else its active)
u = pulp.LpVariable.dicts("u", [(p, w) for p in plants for w in warehouses], cat='Binary')

# (t[(w,c)] = 0, it means that the w->c link is inactive, else its active)
t = pulp.LpVariable.dicts("t", [(w, c) for w in warehouses for c in customers], cat='Binary')

# -------------------------- Objective Function: Minimize total cost --------------------------


model += (
   pulp.lpSum(cost_SP[(s, p)] * x[(s, p)] for s in suppliers for p in plants) +
   pulp.lpSum(cost_PW[(p, w)] * y[(p, w)] for p in plants for w in warehouses) +
   pulp.lpSum(cost_WC[(w, c)] * z[(w, c)] for w in warehouses for c in customers) +
   pulp.lpSum(plant_fixed_cost[p] * Y[p] for p in plants) +
   pulp.lpSum(warehouse_fixed_cost[w] * W[w] for w in warehouses)
), "Total_Cost"

# -------------------------- Constraints --------------------------


# (A) Disruption Constraints: Force flows and linking variables to zero on disrupted arcs.
for s in suppliers:
```

```
    for p in plants:
        if avail_SP[(s, p)] == 0:
            model += x[(s, p)] == 0, f"Disruption_SP_{s}_{p}"
            model += r[(s, p)] == 0, f"Disruption_Link_SP_{s}_{p}"

for p in plants:
    for w in warehouses:
        if avail_PW[(p, w)] == 0:
            model += y[(p, w)] == 0, f"Disruption_PW_{p}_{w}"
            model += u[(p, w)] == 0, f"Disruption_Link_PW_{p}_{w}"

for w in warehouses:
    for c in customers:
        if avail_WC[(w, c)] == 0:
            model += z[(w, c)] == 0, f"Disruption_WC_{w}_{c}"
            model += t[(w, c)] == 0, f"Disruption_Link_WC_{w}_{c}"

# (B) Standard Capacity and Flow Conservation Constraints

# 1. Supplier Capacity: Total shipment from each supplier does not exceed its capacity.
for s in suppliers:
    model += (pulp.lpSum(x[(s, p)] for p in plants) <= supply_capacity[s],
              f"SupplierCapacity_{s}")

# 2. Plant Capacity: Total inflow to each plant ≤ its capacity (if the plant is open).
for p in plants:
    model += (pulp.lpSum(x[(s, p)] for s in suppliers) <= plant_capacity[p] * Y[p],
              f"PlantCapacity_{p}")

# 3. Flow Conservation at Plants: Total inflow equals total outflow for each plant.
for p in plants:
    model += (pulp.lpSum(x[(s, p)] for s in suppliers) ==
              pulp.lpSum(y[(p, w)] for w in warehouses),
              f"PlantFlowConservation_{p}")

# 4. Warehouse Capacity: Inflow to each warehouse ≤ its capacity (if the warehouse is open).
for w in warehouses:
    model += (pulp.lpSum(y[(p, w)] for p in plants) <= warehouse_capacity[w] * W[w],
              f"WarehouseCapacity_{w}")

# 5. Flow Conservation at Warehouses: Inflow equals outflow for each warehouse.
for w in warehouses:
    model += (pulp.lpSum(y[(p, w)] for p in plants) ==
              pulp.lpSum(z[(w, c)] for c in customers),
              f"WarehouseFlowConservation_{w}")

# 6. Customer Demand Satisfaction: Each customer's demand must be met.
for c in customers:
    model += (pulp.lpSum(z[(w, c)] for w in warehouses) >= demand[c],
              f"DemandSatisfaction_{c}")

# 7. Resilience Constraint at Plants:
#    (a) Link supplier-to-plant flow to a binary variable.
for s in suppliers:
    for p in plants:
        model += (x[(s, p)] <= bigM_SP * r[(s, p)],
                  f"Link_SP_{s}_{p}")
#    (b) If a plant is open, it must receive flows from at least 2 different suppliers
for p in plants:
    model += (pulp.lpSum(r[(s, p)] for s in suppliers) >= 2 * Y[p],
              f"SupplierRedundancy_{p}")

# 7. Resilience Constraint at Warehouses:
#    (a) Link plant-to-warehouses flow to a binary variable.
for p in plants:
    for w in warehouses:
        model += (y[(p, w)] <= bigM_WC * u[(p, w)],
                  f"Link_PW_{p}_{w}")
```

```
#    (b) If a warehouse is open, it must receive flows from at least 2 different plants
for w in warehouses:
   model += (pulp.lpSum(u[(p, w)] for p in plants) >= 2 * W[w],
          f"PlantRedundancy_{w}")

# 8. Resilience Constraint at Customers:
#    (a) Link warehouse-to-customer flow to a binary variable.
for w in warehouses:
   for c in customers:
      model += (z[(w, c)] <= bigM_WC * t[(w, c)],
            f"Link_WC_{w}_{c}")
#    (b) Each customer must receive flows from at least 2 distinct warehouses.
for c in customers:
   model += (pulp.lpSum(t[(w, c)] for w in warehouses) >= 2,
          f"WarehouseRedundancy_{c}")

# --------------------------
# Solve the Model
# --------------------------
solver = pulp.PULP_CBC_CMD(msg=True)
model.solve(solver)

print("\n=== Model Status ===")
print(pulp.LpStatus[model.status])
print("\nOptimal Total Cost: {:.2f}".format(pulp.value(model.objective)))

print("\n--- Open Plants ---")
for p in plants:
   if Y[p].varValue is not None and Y[p].varValue > 0.5:
      print(f"{p} is Open (Fixed Cost = {plant_fixed_cost[p]}, Capacity = {plant_capacity[p]})")

print("\n--- Open Warehouses ---")
for w in warehouses:
   if W[w].varValue is not None and W[w].varValue > 0.5:
      print(f"{w} is Open (Fixed Cost = {warehouse_fixed_cost[w]}, Capacity = {warehouse_capacity[w]})")

# --------------------------
# Display the complete supply chain for each customer
# --------------------------
print("\n===== Complete Supply Chains for Each Customer =====")
threshold = 1e-5  # to consider a flow as "active"
for c in customers:
   print(f"\nCustomer {c} (Demand: {demand[c]}):")
   # Find all warehouses delivering to customer c with nonzero flow.
   cust_warehouses = [(w, z[(w, c)].varValue) for w in warehouses
                  if z[(w, c)].varValue is not None and z[(w, c)].varValue > threshold]
   if not cust_warehouses:
      print("  No warehouses supply this customer!")
   else:
      for w, flow_wc in cust_warehouses:
         print(f"  Warehouse {w} -> Customer {c}: Flow = {flow_wc}")
         # For each such warehouse, list plants that supply it.
         supplying_plants = [(p, y[(p, w)].varValue) for p in plants
                        if y[(p, w)].varValue is not None and y[(p, w)].varValue > threshold]
         if not supplying_plants:
            print("     No plants supply this warehouse!")
         else:
            for p, flow_pw in supplying_plants:
               print(f"     Plant {p} -> Warehouse {w}: Flow = {flow_pw}")
               # For each plant, list the supplying suppliers.
               supplying_suppliers = [(s, x[(s, p)].varValue) for s in suppliers
                              if x[(s, p)].varValue is not None and x[(s, p)].varValue > threshold]
               if not supplying_suppliers:
                  print("        No suppliers supply this plant!")
               else:
                  for s, flow_sp in supplying_suppliers:
                     print(f"        Supplier {s} -> Plant {p}: Flow = {flow_sp}")

# --------------------------
```

```
# Plotting Graphs
# -------------------------
# 1. Graph: Optimal Decision Variable Values (only flows x, y, z that are nonzero)
var_names = []
var_values = []
for v in model.variables():
    if v.varValue is not None and v.varValue > threshold and v.name.startswith(('x', 'y', 'z')):
        var_names.append(v.name)
        var_values.append(v.varValue)

plt.figure(figsize=(18, 5))
plt.bar(var_names, var_values, color='skyblue')
plt.xticks(rotation=90, fontsize=8)
plt.xlabel("Decision Variable")
plt.ylabel("Optimal Value")
plt.title("Optimal Value vs. Decision Variable")
plt.tight_layout()
plt.show()

# 2. Graph: Slack Values vs. Constraints (if available)
constraint_names = []
slack_values = []
shadow_prices = []
for cname, constraint in model.constraints.items():
    slack = constraint.slack if hasattr(constraint, "slack") else 0.0
    dual = constraint.pi if hasattr(constraint, "pi") and constraint.pi is not None else 0.0
    constraint_names.append(cname)
    slack_values.append(slack)
    shadow_prices.append(dual)
plt.figure(figsize=(24, 5))
plt.plot(constraint_names, slack_values, color='salmon')
plt.xticks(rotation=90, fontsize=8)
plt.xlabel("Constraint")
plt.ylabel("Slack Value")
plt.title("Slack Value vs. Constraint")
plt.tight_layout()
plt.show()

# 3. Graph: Shadow Prices vs. Constraints
plt.figure(figsize=(24, 5))
plt.plot(constraint_names, shadow_prices, color='lightgreen')
plt.xticks(rotation=90, fontsize=8)
plt.xlabel("Constraint")
plt.ylabel("Shadow Price (Dual Value)")
plt.title("Shadow Price vs. Constraint")
plt.tight_layout()
plt.show()
```

References

- IBM ILOG CPLEX Optimization Studio
- Hillier FS, Lieberman GJ. Introduction to Operations Research.
- Taha HA. Operations Research: An Introduction.
- Project Source: Post-Pandemic Resilient Supply Chain Network Design using MILP