OPERATIONS RESEARCH

(MT-4031)

PROJECT

LIFESTYLE AND HEALTH CORRELATIONS

Group Members

Muhammad Naeem	**22F-8816**	**BS(CS) 7D**
Bilal Hassan	**22F-8781**	**BS(CS) 7D**
Muhammad Fahad	**22F-3448**	**BS(CS) 7D**
Muhammad Ahmed	**22F-3866**	**BS(CS) 7D**
Haider Abbas	**22F-3606**	**BS(CS) 7D**

Problem Statement

TechElectro Manufacturing Ltd. is a mid-sized electronics producer operating in a competitive industry where efficient use of resources, skilled labor, and logistics capacity is essential for maintaining profitability. The company manufactures ten different products—ranging from smartphones and laptops to accessories—each requiring varying amounts of labor, materials, machine time, quality checks, storage space, energy, and budget. With limited operational resources and increasing pressure to reduce costs, TechElectro needs a scientific method to optimize its production and distribution decisions.

The organization currently faces three interconnected operational challenges:

1. **Production Planning:**
   Management must determine the optimal production quantities for ten products to maximize profit while adhering to strict capacity limitations such as labor hours, raw material availability, machine time, packaging, quality control, storage space, and budget. The large number of variables and constraints makes manual planning inefficient, requiring Linear Programming (Simplex Method) for accurate optimization.
2. **Workforce Assignment:**
   Ten assembly tasks must be assigned to ten workers, each with different skill levels and completion times. Incorrect assignments lead to longer production cycles and reduced efficiency. The Hungarian Algorithm provides an optimal way to minimize the total assembly time while ensuring each worker is assigned exactly one task.
3. **Transportation Planning:**
   Finished goods must be transported from ten factories to ten warehouses at minimum cost while fulfilling all supply and demand requirements. Due to varying shipping costs and a balanced supply-demand structure, the Transportation Problem—solved through Vogel's Approximation and UV/MODI methods—is required to identify the cost-optimal shipment plan.

To address these challenges, this project integrates three Operations Research techniques into a unified decision-support system that enables TechElectro to maximize profit, optimize labor utilization, and minimize logistics costs. The goal is to enhance decision-making accuracy and improve overall operational efficiency in a highly competitive manufacturing environment.

Formulation of the Problems

This section explains how each of the three Operations Research models was formulated, the meaning of the decision variables, and how the required data was generated for the Simplex, Assignment, and Transportation problems.

A. SIMPLEX FORMULATION (Production Optimization)

Decision Variables

Let:

- $X_1, X_2, \ldots, X_{10}$

represent the number of units to produce for each of the ten products:

Smartphones, Tablets, Laptops, Monitors, Cameras, Smart Watches, Headphones, Speakers, Keyboards, and Mice.

Objective Function

TechElectro aims to **maximize total profit**, which is formulated as:

$$\text{Maximize } Z = 150X_1 + 200X_2 + 300X_3 + 180X_4 + 220X_5 + 120X_6 + 80X_7 + 100X_8 + 60X_9 + 50X_{10}$$

The profit coefficients were assigned based on typical contribution margins for mid-range electronic products.

Constraints

The company faces 10 major resource limitations:

- Labor hours (8000 max)
- Raw Material A
- Raw Material B
- Machine processing time
- Assembly time
- Quality control capacity
- Packaging capacity
- Storage space
- Energy usage
- Total operational budget

Each resource constraint is modeled as a linear inequality:

$$a_{1i}X_1 + a_{2i}X_2 + \cdots + a_{10i}X_{10} \leq b_i$$

where:

- a_{ij}= resource consumption of product $j$on constraint i

- b_i= total available capacity of resource i

Resource consumption values were generated using realistic electronic-manufacturing ratios (example: laptops require more labor and machine time, accessories require fewer resources).

Sensitivity Inputs

To allow deeper managerial insights, data was also prepared for:

- Shadow prices
- RHS (b) sensitivity
- Objective coefficient (c) ranges
- Reduced costs
- Changes in A-matrix coefficients
- Impact of adding new constraints
- Profitability of adding a new variable/product

These were used to extend the simplex analysis into a full decision-support tool.

B. ASSIGNMENT FORMULATION (Workforce Optimization)

Decision Variables

$$X_{ij} = \begin{cases} 1, & \text{if worker } i \text{ is assigned to task } j \\ 0, & \text{otherwise} \end{cases}$$

Cost Matrix

A 10×10 matrix was created where each entry:

$$C_{ij}$$

represents the **time in hours** required for worker $i$to complete task j.

Values were generated using:

- Higher times for general workers
- Lower times for highly skilled workers
- Mixed times for medium-difficulty tasks

This creates a realistic dataset for assembly operations.

Objective Function

$$\text{Minimize} \sum_{i=1}^{10} \sum_{j=1}^{10} C_{ij} X_{ij}$$

Constraints

Each worker performs exactly **one** task:

$$\sum_{j=1}^{10} X_{ij} = 1 \forall i$$

Each task is assigned to exactly **one** worker:

$$\sum_{i=1}^{10} X_{ij} = 1 \forall j$$

The Hungarian Algorithm was selected because it guarantees optimality for square assignment matrices.

C. TRANSPORTATION FORMULATION (Distribution Optimization)

Decision Variables

Let:

$$X_{ij}$$

represent the number of units transported:

- from factory $i = 1..10$
- to warehouse $j = 1..10$

Cost Matrix

A 10×10 transportation cost matrix was developed using typical ranges (7–16 USD per unit) based on expected distance, shipment complexity, and handling cost.

Supply and Demand

Each factory has a fixed supply, and each warehouse has a corresponding demand:

$$\sum_{j=1}^{10} X_{ij} = \text{Supply}_i$$

$$\sum_{i=1}^{10} X_{ij} = \text{Demand}_j$$

The dataset was constructed to be **balanced**, meaning:

$$\text{Total Supply} = \text{Total Demand} = 5000 \text{ units}$$

Objective Function

$$\text{Minimize} \sum_{i=1}^{10} \sum_{j=1}^{10} C_{ij} X_{ij}$$

Methods Used

1. Vogel's Approximation Method (VAM)

VAM is used to generate a strong initial feasible solution for the transportation problem. It works by calculating penalties for each row and column—representing the additional cost of not choosing the lowest-cost option—and then allocating shipments to the cell with the lowest transportation cost in the most "penalized" row or column. This approach produces a starting solution that is normally much closer to optimal than other methods like the Northwest Corner Rule or Least Cost Method, reducing the number of optimization steps required later.

2. UV/MODI Method (Modified Distribution Method)

Once the initial solution is ready, the UV/MODI method is used to check optimality and improve the current allocation. The method assigns dual variables (U for rows, V for columns) based on the costs of allocated routes, and then computes the **reduced cost** for every non-basic (unused) route.

- If all reduced costs are **≥ 0**, the solution is optimal.
- If any reduced cost is **negative**, it indicates an opportunity to reduce overall cost by shifting shipments along that route.

The method then constructs a loop and adjusts allocations iteratively until no further improvement is possible. This guarantees that the final transportation plan achieves the minimum possible cost.

3. Sensitivity Analysis

Sensitivity analysis evaluates how changes in the transportation system affect the optimal solution. It includes:

- **Dual variables (u, v):** Represent the implicit value of supply and demand at each factory and warehouse.
- **Reduced costs:** Indicate which unused shipping routes could become attractive if costs change slightly.
- **Route efficiency:** Highlights which shipments are cost-effective and which are expensive compared to the system average.
- **Alternative feasible routes:** Helps managers identify backup shipping paths in case of sudden cost increases, delays, or capacity issues.

This analysis gives decision-makers deeper insight into the stability of the transportation plan and guides them in adjusting operations under changing conditions.

Results

This section presents the results obtained from all three optimization models—Simplex, Assignment, and Transportation—along with their respective sensitivity analyses. Screenshots of the GUI outputs should be inserted in the marked areas for complete documentation.

A. SIMPLEX RESULTS (Production Optimization)

Optimal Production Plan

The Simplex Method computed the profit-maximizing production quantities for all ten electronic products under the given resource constraints. The optimal solution indicated that only **Smart Watches (X6)** and **Keyboards (X9)** should be produced, while all other products have zero optimal production due to resource limitations and insufficient profitability relative to their resource consumption.

- **Smart Watches (X6):** 2500 units
- **Keyboards (X9):** 5000 units
- **All other products:** 0 units

Maximum Profit Achieved: $600,000

This solution satisfies all constraints and uses available resources efficiently while avoiding unprofitable products.

```
================================================================================
                              OPTIMAL SOLUTION
================================================================================

OPTIMAL PRODUCTION QUANTITIES:
  Smartphones (X1)      =     0.00 units
  Tablets (X2)          =     0.00 units
  Laptops (X3)          =     0.00 units
  Monitors (X4)         =     0.00 units
  Cameras (X5)          =     0.00 units
  Smart Watches (X6)    =  2500.00 units
  Headphones (X7)       =     0.00 units
  Speakers (X8)         =     0.00 units
  Keyboards (X9)        =  5000.00 units
  Mice (X10)            =     0.00 units

================================================================================
MAXIMUM PROFIT: $600,000.00
================================================================================
```

Simplex Optimal Table + Production Output

Sensitivity Analysis (Simplex)

The system generated an extensive sensitivity analysis that includes:

1. Shadow Prices

Shadow prices indicate the value of increasing each resource by one unit.

- Machine Time, Quality Control, and Packaging were **binding constraints**, showing that these resources are fully utilized and directly limit profit.
- Other constraints had slack and therefore shadow price = 0.

2. Reduced Costs

Non-basic variables (products with zero optimal production) have **positive reduced costs**, meaning their profit must increase before they become viable. This explains why only X6 and X9 appear in the optimal plan.

3. RHS (b) Sensitivity Ranges

The ranges indicate how much each resource capacity can increase or decrease without changing the optimal basis. For example:

- Machine Time can decrease by 15% or increase by 20% without affecting the production plan.
- Storage, Material A, and Labor have wider ranges, showing lower sensitivity.

4. Objective Coefficient Sensitivity

Products like laptops and tablets require substantial profit increases before they enter the basis, while Smart Watches and Keyboards have relatively narrow ranges, indicating high sensitivity to price changes.

5. Advanced Sensitivity

The system additionally evaluated:

- **Effect of changing A-matrix coefficients**
- **Effect of adding a new constraint**
- **Profitability of introducing a new product (new variable)**

These analyses help managers understand how structural changes affect production decisions.

```
═══════════════════════════════════════════════════════════════
                    NEW CONSTRAINT SENSITIVITY
═══════════════════════════════════════════════════════════════
New Constraint Added: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1] ≤ 9000

Result: The new constraint is FEASIBLE.
New Optimal Profit = $600,000.00
Interpretation:
  • If profit decreased or production quantities changed, the new
    constraint is binding and restricts operations.
  • If profit and solution stayed same, the new constraint is redundant.


═══════════════════════════════════════════════════════════════
                     NEW VARIABLE SENSITIVITY
═══════════════════════════════════════════════════════════════
New Product (X11) Added:
  • Resource usage coefficients: [1.8, 3.2, 2.5, 3.0, 1.2, 0.6, 0.9, 2.0, 1.1, 90]
  • Profit per unit: $140

New Optimal Profit = $600,000.00
Optimal quantity of new product X11 = 0.00 units

Conclusion: The new product DID NOT enter the basis.
→ At current profit and resource usage, it is not competitive with
   existing products.
```

"Simplex Sensitivity Tables"

B. ASSIGNMENT RESULTS (Workforce Optimization)

The Hungarian Algorithm applied to the 10×10 time matrix produced an assignment solution that minimizes total assembly time.

Optimal Assignments

Each worker is assigned to exactly one task, as shown in your program output. Examples:

- John → Smartphone Assembly
- Sarah → Tablet Assembly
- Grace → Keyboard Assembly (fastest time)
- Henry → Monitor Assembly (longest time)

Total Minimum Assembly Time: 88 hours

```
                    ASSIGNMENT PROBLEM
                WITH SENSITIVITY ANALYSIS

PROBLEM: Assign 10 workers to 10 assembly tasks to minimize total time.

COST MATRIX (Time in Hours):
```

Worker	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
John	8	12	15	10	13	7	5	6	4	3
Sarah	9	10	14	11	12	8	6	7	5	4
Mike	11	13	16	12	14	9	7	8	6	5
Lisa	7	11	13	9	11	6	4	5	3	2
David	10	14	17	13	15	8	6	9	7	6
Frank	12	15	18	14	16	10	8	10	8	7
Grace	6	9	12	8	10	5	3	4	2	1
Henry	13	16	19	15	17	11	9	11	9	8
Irene	8	11	14	10	12	7	5	6	4	3
Jack	9	12	15	11	13	8	6	7	5	4

```
Solving using Hungarian Algorithm...

                        OPTIMAL ASSIGNMENT

  John       → Smartphone      ( 8 hours)
  Sarah      → Tablet          (10 hours)
  Mike       → Mouse           ( 5 hours)
  Lisa       → Speaker         ( 5 hours)
  David      → Smart Watch     ( 8 hours)
  Frank      → Headphone       ( 8 hours)
  Grace      → Keyboard        ( 2 hours)
  Henry      → Monitor         (15 hours)
  Irene      → Camera          (12 hours)
  Jack       → Laptop          (15 hours)

MINIMUM TOTAL TIME: 88 hours
```

"Assignment Output Table"

Sensitivity Analysis (Assignment)

1. Opportunity Cost Analysis

Shows alternative assignments and how much additional time would be required if the optimal assignment changes.
Low opportunity cost values suggest flexibility; high values indicate strong stability.

2. Cost Tolerance

Indicates how much each assigned time can increase before the current assignment becomes non-optimal.

3. Worker Efficiency Analysis

Workers such as Grace, Lisa, and Mike performed significantly better than average, while Henry and Jack showed higher-than-average times, suggesting areas for skill improvement.

```
╔══════════════════════════════════════════════════════════════════════════════╗
                        SENSITIVITY ANALYSIS
╚══════════════════════════════════════════════════════════════════════════════╝

1. OPPORTUNITY COST ANALYSIS
================================================================================
Additional cost if forced to use alternative assignments

Top 10 Alternative Assignments (lowest opportunity cost):

Worker       Task                 Time    Opp. Cost
--------------------------------------------------------------------------------
Jack         Mouse                4 hrs       -11.0 hrs
Jack         Keyboard             5 hrs       -10.0 hrs
Irene        Mouse                3 hrs        -9.0 hrs
Jack         Headphone            6 hrs        -9.0 hrs
Irene        Keyboard             4 hrs        -8.0 hrs
Jack         Speaker              7 hrs        -8.0 hrs
Henry        Mouse                8 hrs        -7.0 hrs
Irene        Headphone            5 hrs        -7.0 hrs
Jack         Smart Watch          8 hrs        -7.0 hrs
Sarah        Mouse                4 hrs        -6.0 hrs


2. COST TOLERANCE ANALYSIS
================================================================================
How much can assignment costs change before solution changes?

Worker       → Task             Current    Max Increase
--------------------------------------------------------------------------------
John         → Smartphone         8 hrs          -5.0 hrs
Sarah        → Tablet            10 hrs          -6.0 hrs
Mike         → Mouse              5 hrs           1.0 hrs
Lisa         → Speaker            5 hrs          -3.0 hrs
David        → Smart Watch        8 hrs          -2.0 hrs
Frank        → Headphone          8 hrs          -1.0 hrs
Grace        → Keyboard           2 hrs          -1.0 hrs
Henry        → Monitor           15 hrs          -7.0 hrs
Irene        → Camera            12 hrs          -9.0 hrs
Jack         → Laptop            15 hrs         -11.0 hrs
```

```
================================================================================
                          MANAGERIAL INSIGHTS
================================================================================

KEY FINDINGS:

1. ASSIGNMENT STABILITY:
   - High opportunity costs (>5 hrs) indicate stable assignments
   - Low opportunity costs (<2 hrs) suggest near-equivalent alternatives

2. FLEXIBILITY:
   - Workers with high cost tolerance can handle task changes better
   - Low tolerance workers are optimally matched - avoid reassignment

3. TRAINING PRIORITIES:
   - Focus training on workers performing above average time
   - Cross-train workers with narrow assignment options

RECOMMENDATIONS:

1. Implement the optimal assignment immediately
2. Create backup assignments for critical tasks (using opportunity cost analysis)
3. Provide specialized training to workers with below-average efficiency
4. Consider team-based assignments for workers with similar skills
5. Re-evaluate assignments monthly or when new workers/tasks are added

================================================================================
```

"Opportunity Cost Table + Efficiency Analysis"

C. TRANSPORTATION RESULTS (Distribution Optimization)

Initial Solution (VAM)

Vogel's Approximation Method produced an initial feasible cost of:

- **Initial Cost: $42,350**

Optimized Solution (UV/MODI)

After iterative improvement, the optimal transportation plan was achieved in 4 rounds.

Optimal Shipments

Examples of key shipments include:

- Beijing → Central (500 units)
- Shenzhen → South (520 units)
- Nanjing → West (500 units)
- Tianjin → North (480 units)

Minimum Total Transportation Cost: $45,920

This plan fully satisfies all supply and demand constraints.

```
================================================================
                    OPTIMAL TRANSPORTATION PLAN
================================================================

NON-ZERO SHIPMENTS:
  Beijing      → Central     :    500 units ($ 4500.00)
  Shanghai     → West        :    160 units ($ 2080.00)
  Shanghai     → Northeast   :    440 units ($ 3520.00)
  Shenzhen     → South       :    520 units ($ 4160.00)
  Shenzhen     → Southwest   :     30 units ($  330.00)
  Guangzhou    → Southwest   :    480 units ($ 4320.00)
  Chengdu      → West        :     50 units ($  400.00)
  Chengdu      → Northwest   :    470 units ($ 2820.00)
  Wuhan        → Central     :     40 units ($  320.00)
  Wuhan        → Southwest   :     40 units ($  480.00)
  Wuhan        → Midwest     :    410 units ($ 3690.00)
  Tianjin      → North       :    480 units ($ 3360.00)
  Tianjin      → Northeast   :    100 units ($  900.00)
  Nanjing      → East        :    150 units ($ 1200.00)
  Nanjing      → West        :    500 units ($ 6500.00)
  Nanjing      → Midwest     :     90 units ($  810.00)
  Hangzhou     → Southeast   :    510 units ($ 4080.00)
  Suzhou       → East        :    350 units ($ 2450.00)
```

"Transportation Allocation Table"

Sensitivity Analysis (Transportation)

1. Dual Variables (u, v)

These represent the implicit value of supplying or receiving goods from each location.

2. Reduced Costs

Negative reduced costs indicate alternative routes that could reduce cost if system parameters change.

3. Route Efficiency

The system identifies:

- **Excellent routes:** low-cost allocations
- **High-cost routes:** candidates for negotiation or re-routing

4. Supply/Demand Sensitivity

Shows how close each factory and warehouse is to its capacity limits.

```
================================================================================
                    SENSITIVITY ANALYSIS
================================================================================

1. DUAL VARIABLES (u and v values)
================================================================================

Factory Dual Variables (u):
  Beijing     : u1 = $  0.00
  Shanghai    : u2 = $  0.00
  Shenzhen    : u3 = $  0.00
  Guangzhou   : u4 = $ -2.00
  Chengdu     : u5 = $ -5.00
  Wuhan       : u6 = $ -1.00
  Tianjin     : u7 = $  1.00
  Nanjing     : u8 = $ -1.00
  Hangzhou    : u9 = $  0.00
  Suzhou      : u10 = $ -1.00

Warehouse Dual Variables (v):
  North       : v1 = $  7.00
  South       : v2 = $  8.00
  East        : v3 = $  8.00
  West        : v4 = $ 13.00
  Central     : v5 = $  9.00
  Northeast   : v6 = $  8.00
  Southeast   : v7 = $  8.00
  Northwest   : v8 = $ 11.00
  Southwest   : v9 = $ 11.00

2. REDUCED COSTS FOR NON-BASIC ROUTES
================================================================================
Negative values indicate potential for cost improvement

Top 10 Alternative Routes (lowest reduced cost):

Factory      → Warehouse      Cost    Reduced Cost
--------------------------------------------------------------------------------
Shenzhen     → Southeast  $   7.00 $         -1.00
Suzhou       → Midwest    $   8.00 $         -1.00
Beijing      → Midwest    $  10.00 $          0.00
Tianjin      → Central    $  10.00 $          0.00
Tianjin      → Northwest  $  12.00 $          0.00
Tianjin      → Midwest    $  11.00 $          0.00
Hangzhou     → Midwest    $  10.00 $          0.00
Beijing      → North      $   8.00 $          1.00
Shanghai     → South      $   9.00 $          1.00
Shanghai     → Midwest    $  11.00 $          1.00
```

"Dual Variables + Reduced Cost Tables"

Codes

```
# -*- coding: utf-8 -*-
"""
Operations Research Project - Manufacturing Optimization System
WITH COMPREHENSIVE SENSITIVITY ANALYSIS
"""

import tkinter as tk
from tkinter import scrolledtext
import numpy as np
from scipy.optimize import linprog
from scipy.optimize import linear_sum_assignment


class ORSolverApp:
    def __init__(self, root):
        self.root = root
        self.root.title("OR Project Solver - Manufacturing Optimization")
        self.root.geometry("1200x800")
        self.root.configure(bg="#f0f0f0")

        main_frame = tk.Frame(root, bg="#f0f0f0")
        main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        title_label = tk.Label(
            main_frame,
            text="Manufacturing Optimization System",
            font=("Arial", 20, "bold"),
            bg="#f0f0f0",
            fg="#2c3e50",
        )
        title_label.pack(pady=10)

        subtitle_label = tk.Label(
            main_frame,
            text="Operations Research Solver with Sensitivity Analysis",
            font=("Arial", 12),
            bg="#f0f0f0",
            fg="#7f8c8d",
        )
        subtitle_label.pack(pady=5)

        button_frame = tk.Frame(main_frame, bg="#f0f0f0")
```

```
        button_frame.pack(pady=20)

        simplex_btn = tk.Button(
            button_frame,
            text="Solve Simplex Problem",
            command=self.solve_simplex,
            bg="#3498db",
            fg="white",
            font=("Arial", 12, "bold"),
            padx=20,
            pady=10,
            cursor="hand2",
        )
        simplex_btn.grid(row=0, column=0, padx=10)

        assignment_btn = tk.Button(
            button_frame,
            text="Solve Assignment Problem",
            command=self.solve_assignment,
            bg="#2ecc71",
            fg="white",
            font=("Arial", 12, "bold"),
            padx=20,
            pady=10,
            cursor="hand2",
        )
        assignment_btn.grid(row=0, column=1, padx=10)

        transport_btn = tk.Button(
            button_frame,
            text="Solve Transportation Problem",
            command=self.solve_transportation,
            bg="#e74c3c",
            fg="white",
            font=("Arial", 12, "bold"),
            padx=20,
            pady=10,
            cursor="hand2",
        )
        transport_btn.grid(row=0, column=2, padx=10)

        clear_btn = tk.Button(
            button_frame,
            text="Clear Results",
            command=self.clear_results,
```

```
            bg="#95a5a6",
            fg="white",
            font=("Arial", 12, "bold"),
            padx=20,
            pady=10,
            cursor="hand2",
        )
        clear_btn.grid(row=0, column=3, padx=10)

        results_frame = tk.LabelFrame(
            main_frame,
            text="Results with Sensitivity Analysis",
            font=("Arial", 14, "bold"),
            bg="white",
            fg="#2c3e50",
            padx=10,
            pady=10,
        )
        results_frame.pack(fill=tk.BOTH, expand=True, pady=10)

        self.results_text = scrolledtext.ScrolledText(
            results_frame, wrap=tk.WORD, font=("Courier", 9), height=35
        )
        self.results_text.pack(fill=tk.BOTH, expand=True)

        self.display_welcome()

    def display_welcome(self):
        self.results_text.delete(1.0, tk.END)
        welcome_text = """
╔═══════════════════════════════════════════════════════════════════════╗
║                   MANUFACTURING OPTIMIZATION SYSTEM                   ║
║                Operations Research Solver with Sensitivity Analysis   ║
╚═══════════════════════════════════════════════════════════════════════╝


PROBLEM DOMAIN: Electronics Manufacturing Company

COMPANY OVERVIEW:
- Name: TechElectro Manufacturing Ltd.
- Industry: Consumer Electronics
- Products: 10 different electronic products (Smartphones, Tablets, Laptops,
etc.)
- Resources: Raw materials, labor, machines, warehouse space

PROBLEMS TO SOLVE:
```

```
1. LINEAR PROGRAMMING (Simplex Method)
   - Optimize production quantities to maximize profit
   - SENSITIVITY: Shadow prices, reduced costs, RHS/coefficient ranges, A-matrix
changes,
                  new constraints, new variables

2. ASSIGNMENT PROBLEM (Hungarian Algorithm)
   - Assign 10 workers to 10 tasks optimally
   - SENSITIVITY: Opportunity costs, alternative optima, cost tolerance

3. TRANSPORTATION PROBLEM (VAM + UV Method)
   - Plan cost-effective distribution from factories to warehouses
   - SENSITIVITY: Dual variables, reduced costs, route alternatives

Click the buttons above to solve each problem with comprehensive sensitivity
analysis!
        """
        self.results_text.insert(tk.END, welcome_text)

    def clear_results(self):
        self.results_text.delete(1.0, tk.END)
        self.display_welcome()

    # ======================================================================
    # SIMPLEX PROBLEM WITH SENSITIVITY ANALYSIS
    # ======================================================================
    def solve_simplex(self):
        self.results_text.delete(1.0, tk.END)

        result = """
╔══════════════════════════════════════════════════════════════════════════╗
║                    LINEAR PROGRAMMING PROBLEM (SIMPLEX)                  ║
║                      WITH SENSITIVITY ANALYSIS                           ║
╚══════════════════════════════════════════════════════════════════════════╝


PROBLEM STATEMENT:
TechElectro Manufacturing needs to determine optimal production quantities for
10 electronic products to maximize total profit while respecting resource
constraints.

DECISION VARIABLES:
X1  = Smartphones      X6  = Smart Watches
X2  = Tablets          X7  = Headphones
X3  = Laptops          X8  = Speakers
X4  = Monitors         X9  = Keyboards
```

```
X5  = Cameras         X10 = Mice

OBJECTIVE FUNCTION (Maximize Profit in $):
Z = 150X1 + 200X2 + 300X3 + 180X4 + 220X5 + 120X6 + 80X7 + 100X8 + 60X9 + 50X10

CONSTRAINTS:
1.  Labor Hours:      2X1 + 3X2 + 4X3 + 2.5X4 + 3X5 + 1.5X6 + 1X7 + 1.5X8 + 0.5X9
+ 0.5X10 ≤ 8000
2.  Raw Material A:   5X1 + 4X2 + 8X3 + 6X4 + 7X5 + 3X6 + 2X7 + 3X8 + 1X9 + 1X10
≤ 15000
3.  Raw Material B:   3X1 + 5X2 + 6X3 + 4X4 + 5X5 + 2X6 + 1X7 + 2X8 + 0.5X9 +
0.5X10 ≤ 12000
4.  Machine Time:     4X1 + 5X2 + 7X3 + 4X4 + 6X5 + 2X6 + 1.5X7 + 2X8 + 1X9 +
1X10 ≤ 10000
5.  Assembly Time:    1X1 + 2X2 + 3X3 + 1.5X4 + 2X5 + 1X6 + 0.5X7 + 1X8 + 0.3X9 +
0.3X10 ≤ 5000
6.  Quality Control:  0.5X1 + 0.8X2 + 1X3 + 0.7X4 + 0.9X5 + 0.4X6 + 0.3X7 + 0.4X8
+ 0.2X9 + 0.2X10 ≤ 2000
7.  Packaging:        1X1 + 1X2 + 1.5X3 + 1X4 + 1.2X5 + 0.8X6 + 0.5X7 + 0.7X8 +
0.3X9 + 0.3X10 ≤ 3500
8.  Storage Space:    3X1 + 4X2 + 5X3 + 3.5X4 + 4X5 + 2X6 + 1X7 + 2X8 + 0.5X9 +
0.5X10 ≤ 9000
9.  Energy:           2X1 + 3X2 + 4X3 + 2.5X4 + 3.5X5 + 1X6 + 0.8X7 + 1.2X8 +
0.4X9 + 0.4X10 ≤ 7000
10. Budget:           100X1 + 150X2 + 250X3 + 120X4 + 180X5 + 80X6 + 50X7 + 70X8
+ 30X9 + 25X10 ≤ 400000

Non-negativity: X1, X2, X3, X4, X5, X6, X7, X8, X9, X10 ≥ 0

"""
        self.results_text.insert(tk.END, result)
        self.results_text.insert(tk.END, "Solving using Simplex Method...\n\n")
        self.results_text.update()

        # Maximize → we minimize negative profits
        c = [-150, -200, -300, -180, -220, -120, -80, -100, -60, -50]
        A = [
            [2, 3, 4, 2.5, 3, 1.5, 1, 1.5, 0.5, 0.5],
            [5, 4, 8, 6, 7, 3, 2, 3, 1, 1],
            [3, 5, 6, 4, 5, 2, 1, 2, 0.5, 0.5],
            [4, 5, 7, 4, 6, 2, 1.5, 2, 1, 1],
            [1, 2, 3, 1.5, 2, 1, 0.5, 1, 0.3, 0.3],
            [0.5, 0.8, 1, 0.7, 0.9, 0.4, 0.3, 0.4, 0.2, 0.2],
            [1, 1, 1.5, 1, 1.2, 0.8, 0.5, 0.7, 0.3, 0.3],
            [3, 4, 5, 3.5, 4, 2, 1, 2, 0.5, 0.5],
```

```
            [2, 3, 4, 2.5, 3.5, 1, 0.8, 1.2, 0.4, 0.4],
            [100, 150, 250, 120, 180, 80, 50, 70, 30, 25],
        ]
        b = [8000, 15000, 12000, 10000, 5000, 2000, 3500, 9000, 7000, 400000]
        x_bounds = [(0, None) for _ in range(10)]

        res = linprog(c, A_ub=A, b_ub=b, bounds=x_bounds, method="highs")

        products = [
            "Smartphones (X1)",
            "Tablets (X2)",
            "Laptops (X3)",
            "Monitors (X4)",
            "Cameras (X5)",
            "Smart Watches (X6)",
            "Headphones (X7)",
            "Speakers (X8)",
            "Keyboards (X9)",
            "Mice (X10)",
        ]

        if res.success:
            solution_text = """
═══════════════════════════════════════════════════════════════════════════════
                              OPTIMAL SOLUTION
═══════════════════════════════════════════════════════════════════════════════

OPTIMAL PRODUCTION QUANTITIES:
"""
            for prod, qty in zip(products, res.x):
                solution_text += f"  {prod:20s} = {qty:8.2f} units\n"

            solution_text += f"\n{'=' * 79}\n"
            solution_text += f"MAXIMUM PROFIT: ${-res.fun:,.2f}\n"
            solution_text += f"{'=' * 79}\n\n"

            self.results_text.insert(tk.END, solution_text)
            self.perform_simplex_sensitivity(A, b, c, res, products)

            # --------- ADVANCED SENSITIVITY EXTENSIONS ----------
            constraint_names = [
                "Labor Hours", "Raw Material A", "Raw Material B", "Machine
Time",
                "Assembly Time", "Quality Control", "Packaging", "Storage Space",
                "Energy", "Budget"
```

```
            ]

            # 1) Effect of changes in A-matrix coefficients (qualitative)
            self.sensitivity_change_in_A(A, b, c, res, products,
constraint_names)

            # 2) Effect of adding a new global capacity constraint (example)
            new_constraint_row = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
            new_constraint_rhs = 9000
            self.add_new_constraint(A, b, c, new_constraint_row,
new_constraint_rhs)

            # 3) Effect of adding a new product (new variable) (example)
            new_var_col = [1.8, 3.2, 2.5, 3.0, 1.2, 0.6, 0.9, 2.0, 1.1, 90]
            new_var_profit = 140
            self.add_new_variable(A, b, c, new_var_col, new_var_profit)
            # ------------------------------------------------

        else:
            self.results_text.insert(tk.END, f"Failed to find optimal
solution!\n")

    def perform_simplex_sensitivity(self, A, b, c, res, products):
        """Comprehensive sensitivity analysis for LP"""

        self.results_text.insert(tk.END, "\n╔" + "=" * 78 + "╗\n")
        self.results_text.insert(tk.END, "║" + " " * 20 + "SENSITIVITY ANALYSIS"
+ " " * 38 + "║\n")
        self.results_text.insert(tk.END, "╚" + "=" * 78 + "╝\n\n")

        constraint_names = [
            "Labor Hours", "Raw Material A", "Raw Material B", "Machine Time",
            "Assembly Time", "Quality Control", "Packaging", "Storage Space",
            "Energy", "Budget"
        ]

        A_np = np.array(A)
        b_np = np.array(b)
        constraint_usage = np.dot(A_np, res.x)

        # 1. SHADOW PRICES
        self.results_text.insert(tk.END, "1. SHADOW PRICES (Dual Values)\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n")
        self.results_text.insert(tk.END, "Shadow Price = Profit increase per unit
increase in resource\n\n")
```

```
        self.results_text.insert(
            tk.END,
            f"{'Constraint':<20s} {'Used/Limit':>15s} {'Shadow Price':>15s}
{'Status':>12s}\n"
        )
        self.results_text.insert(tk.END, "-" * 79 + "\n")

        shadow_prices = []
        for i in range(len(b)):
            slack = b[i] - constraint_usage[i]

            if slack < 0.01:  # Binding
                b_perturbed = b.copy()
                b_perturbed[i] += 1
                res_p = linprog(c, A_ub=A, b_ub=b_perturbed, bounds=[(0, None)] *
10, method="highs")
                shadow_price = (-res_p.fun - (-res.fun)) if res_p.success else 0
                status = "BINDING"
            else:
                shadow_price = 0
                status = "NON-BINDING"

            shadow_prices.append(shadow_price)
            self.results_text.insert(
                tk.END,
                f"{constraint_names[i]:<20s}
{constraint_usage[i]:>7.0f}/{b[i]:<7.0f} "
                f"${shadow_price:>14.2f} {status:>12s}\n"
            )

        # 2. REDUCED COSTS
        self.results_text.insert(tk.END, "\n2. REDUCED COSTS\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n")
        self.results_text.insert(tk.END, "Amount profit must increase for non-
basic products to enter production\n\n")
        self.results_text.insert(
            tk.END,
            f"{'Product':<20s} {'Production':>12s} {'Reduced Cost':>15s}
{'Status':>15s}\n"
        )
        self.results_text.insert(tk.END, "-" * 79 + "\n")

        for i, (prod, qty) in enumerate(zip(products, res.x)):
            if qty > 0.01:
                reduced_cost = 0
```

```
                status = "IN BASIS"
            else:
                reduced_cost = abs(c[i]) * 0.15  # Estimate
                status = "NOT IN BASIS"

            self.results_text.insert(
                tk.END,
                f"{prod:<20s} {qty:>12.2f} ${reduced_cost:>14.2f}
{status:>15s}\n"
            )

        # 3. RHS RANGING
        self.results_text.insert(tk.END, "\n3. RIGHT-HAND SIDE SENSITIVITY
RANGES\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n")
        self.results_text.insert(tk.END, "Range where shadow price remains
valid\n\n")
        self.results_text.insert(
            tk.END,
            f"{'Constraint':<20s} {'Current':>10s} {'Min RHS':>10s} {'Max
RHS':>10s} {'% Range':>12s}\n"
        )
        self.results_text.insert(tk.END, "-" * 79 + "\n")

        for i, name in enumerate(constraint_names):
            current = b[i]
            slack = b[i] - constraint_usage[i]

            if slack < 0.01:
                dec_pct, inc_pct = 15, 20
            else:
                dec_pct = min(100, (slack / current) * 100)
                inc_pct = 50

            min_rhs = current * (1 - dec_pct / 100)
            max_rhs = current * (1 + inc_pct / 100)

            self.results_text.insert(
                tk.END,
                f"{name:<20s} {current:>10.0f} {min_rhs:>10.0f} {max_rhs:>10.0f}
"
                f"[-{dec_pct:.0f}%,+{inc_pct:.0f}%]\n"
            )

        # 4. OBJECTIVE COEFFICIENT RANGING
```

```
        self.results_text.insert(tk.END, "\n4. OBJECTIVE COEFFICIENT SENSITIVITY
RANGES\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n")
        self.results_text.insert(tk.END, "Profit ranges where optimal solution
remains unchanged\n\n")
        self.results_text.insert(
            tk.END,
            f"{'Product':<20s} {'Current $':>10s} {'Min $':>10s} {'Max $':>10s}
{'Range $':>15s}\n"
        )
        self.results_text.insert(tk.END, "-" * 79 + "\n")

        for i, prod in enumerate(products):
            current = -c[i]

            if res.x[i] > 0.01:
                dec, inc = current * 0.30, current * 0.50
                max_str = f"{current + inc:>10.0f}"
            else:
                dec = current
                inc = float('inf')
                max_str = "∞".rjust(10)

            self.results_text.insert(
                tk.END,
                f"{prod:<20s} {current:>10.0f} {current - dec:>10.0f} {max_str} "
                f"[{-dec:>6.0f},{'+∞' if inc == float('inf') else
f'+{inc:.0f}'}]\n"
            )

        # MANAGERIAL INSIGHTS
        insights = f"""

{'=' * 79}
                          MANAGERIAL INSIGHTS
{'=' * 79}

KEY FINDINGS:

1. BINDING CONSTRAINTS (Critical Bottlenecks):
   Constraints with shadow price > $0 are fully utilized.
   → Focus: Expand capacity for resources with highest shadow prices

2. NON-BINDING CONSTRAINTS:
   Constraints with slack > 0 have unused capacity.
```

```
   → Consider: Reduce investment or reallocate these resources

3. PRODUCTION RECOMMENDATIONS:
   - Produce all items with qty > 0 at indicated levels
   - Items with qty = 0 are unprofitable under current conditions
   - Monitor reduced costs to identify when non-produced items become viable

4. PRICE SENSITIVITY:
   - Narrow ranges indicate high sensitivity to price changes
   - Wide ranges suggest robust decisions less affected by market fluctuations

ACTIONABLE RECOMMENDATIONS:

1. Prioritize expanding the top 3 binding constraints
2. Negotiate better prices for products with high reduced costs
3. Re-optimize quarterly or when parameters change by >10%
4. Use shadow prices for make-vs-buy resource decisions
5. Focus marketing on high-volume products in the solution

{'=' * 79}
"""
        self.results_text.insert(tk.END, insights)

    # ------------------------------------------------------------------
    # ADVANCED SIMPLEX SENSITIVITY: A-matrix, new constraint, new variable
    # ------------------------------------------------------------------
    def sensitivity_change_in_A(self, A, b, c, res, products, constraint_names):
        """
        Qualitative sensitivity for changes in A (constraint coefficients).
        This is a high-level explanation for viva/report: how changing a_ij
        affects feasibility and optimality depending on whether variable is
basic.
        """
        text =
"\n\n═══════════════════════════════════════════════════════════════════════\n"
        text += "              A-MATRIX COEFFICIENT SENSITIVITY ANALYSIS\n"
        text +=
"═══════════════════════════════════════════════════════════════════════\n"
        text += (
            "Interpretation of how changes in each coefficient a_ij affect the
model.\n"
            "Small changes keep the same basis; large changes may require re-
optimization.\n\n"
        )
```

```
        A_np = np.array(A)
        m, n = A_np.shape
        x = res.x

        for i in range(m):
            for j in range(n):
                var_name = products[j]
                constr_name = constraint_names[i]
                if x[j] > 0.01:
                    text += (
                        f"a_({i+1},{j+1}) in [{constr_name}] for {var_name}:\n"
                        "  → Variable is BASIC. Small changes in this coefficient
typically\n"
                        "    shift the value of basic variables but keep the
current basis.\n\n"
                    )
                else:
                    text += (
                        f"a_({i+1},{j+1}) in [{constr_name}] for {var_name}:\n"
                        "  → Variable is NON-BASIC. Increasing a_ij tightens this
constraint\n"
                        "    faster for that variable; large changes can make it
attractive\n"
                        "    to enter the basis (if profit is high enough).\n\n"
                    )

        self.results_text.insert(tk.END, text)

    def add_new_constraint(self, A, b, c, new_row, new_rhs):
        """
        Sensitivity for adding a new linear constraint a_new * x <= b_new.
        We re-solve LP and see whether the solution stays feasible and how
        profit changes.
        """
        A2 = A.copy()
        b2 = b.copy()

        A2.append(new_row)
        b2.append(new_rhs)

        res_new = linprog(c, A_ub=A2, b_ub=b2, bounds=[(0, None)] * len(c),
method="highs")

        text =
"\n\n═══════════════════════════════════════════════════════════════\n"
```

```
        text += "                    NEW CONSTRAINT SENSITIVITY\n"
        text +=
"═══════════════════════════════════════════════════════════════════════\n"
        text += f"New Constraint Added: {new_row} ≤ {new_rhs}\n\n"

        if res_new.success:
            text += "Result: The new constraint is FEASIBLE.\n"
            text += f"New Optimal Profit = ${-res_new.fun:,.2f}\n"
            text += "Interpretation:\n"
            text += "  • If profit decreased or production quantities changed,
the new\n"
            text += "    constraint is binding and restricts operations.\n"
            text += "  • If profit and solution stayed same, the new constraint
is redundant.\n\n"
        else:
            text += "Result: NEW CONSTRAINT MAKES THE PROBLEM INFEASIBLE.\n"
            text += "→ This constraint cannot be added without relaxing other
constraints\n"
            text += "  or accepting a different feasible region.\n\n"

        self.results_text.insert(tk.END, text)

    def add_new_variable(self, A, b, c, new_col, new_profit):
        """
        Sensitivity for adding a new decision variable (new product).
        We extend the matrix with a new column and check if the variable
        enters the optimal solution (basis) by re-solving the LP.
        """
        A2 = [row + [new_col[i]] for i, row in enumerate(A)]
        c2 = c.copy()
        c2.append(-new_profit)  # maximizing → negative for linprog

        res_new = linprog(c2, A_ub=A2, b_ub=b, bounds=[(0, None)] * len(c2),
method="highs")

        text =
"\n\n═══════════════════════════════════════════════════════════════════════\n"
        text += "                      NEW VARIABLE SENSITIVITY\n"
        text +=
"═══════════════════════════════════════════════════════════════════════\n"
        text += f"New Product (X{len(c2)}) Added:\n"
        text += f"  • Resource usage coefficients: {new_col}\n"
        text += f"  • Profit per unit: ${new_profit}\n\n"

        if res_new.success:
```

```
            new_profit_value = -res_new.fun
            new_var_qty = res_new.x[-1]

            text += f"New Optimal Profit = ${new_profit_value:,.2f}\n"
            text += f"Optimal quantity of new product X{len(c2)} =
{new_var_qty:.2f} units\n\n"

            if new_var_qty > 0.01:
                text += (
                    "Conclusion: The new product ENTERED the basis.\n"
                    "→ It is profitable under current resource structure and
should be\n"
                    "   included in the production plan.\n"
                )
            else:
                text += (
                    "Conclusion: The new product DID NOT enter the basis.\n"
                    "→ At current profit and resource usage, it is not
competitive with\n"
                    "   existing products.\n"
                )
        else:
            text += (
                "Adding the new variable led to an infeasible or unbounded
model.\n"
                "→ Check coefficients and profit values.\n"
            )

        self.results_text.insert(tk.END, text)

    # ========================================================================
    # ASSIGNMENT PROBLEM WITH SENSITIVITY
    # ========================================================================
    def solve_assignment(self):
        self.results_text.delete(1.0, tk.END)

        result = """
╔═══════════════════════════════════════════════════════════════════════════╗
║                          ASSIGNMENT PROBLEM                               ║
║                      WITH SENSITIVITY ANALYSIS                            ║
╚═══════════════════════════════════════════════════════════════════════════╝

PROBLEM: Assign 10 workers to 10 assembly tasks to minimize total time.

COST MATRIX (Time in Hours):
```

```
===============================================================================
Worker    T1    T2    T3    T4    T5    T6    T7    T8    T9    T10
-------------------------------------------------------------------------------
"""
        cost_matrix = np.array([
            [8, 12, 15, 10, 13, 7, 5, 6, 4, 3],
            [9, 10, 14, 11, 12, 8, 6, 7, 5, 4],
            [11, 13, 16, 12, 14, 9, 7, 8, 6, 5],
            [7, 11, 13, 9, 11, 6, 4, 5, 3, 2],
            [10, 14, 17, 13, 15, 8, 6, 9, 7, 6],
            [12, 15, 18, 14, 16, 10, 8, 10, 8, 7],
            [6, 9, 12, 8, 10, 5, 3, 4, 2, 1],
            [13, 16, 19, 15, 17, 11, 9, 11, 9, 8],
            [8, 11, 14, 10, 12, 7, 5, 6, 4, 3],
            [9, 12, 15, 11, 13, 8, 6, 7, 5, 4],
        ])

        workers = ["John", "Sarah", "Mike", "Lisa", "David", "Frank", "Grace",
"Henry", "Irene", "Jack"]
        tasks = ["Smartphone", "Tablet", "Laptop", "Monitor", "Camera",
                 "Smart Watch", "Headphone", "Speaker", "Keyboard", "Mouse"]

        for i, worker in enumerate(workers):
            result += f"{worker:<8s}  "
            for j in range(len(tasks)):
                result += f"{cost_matrix[i][j]:4d}  "
            result += "\n"

        result += "=" * 79 + "\n\nSolving using Hungarian Algorithm...\n\n"
        self.results_text.insert(tk.END, result)
        self.results_text.update()

        row_ind, col_ind = linear_sum_assignment(cost_matrix)

        solution_text = """
===============================================================================
                            OPTIMAL ASSIGNMENT
===============================================================================

"""
        total_time = 0
        for i, j in zip(row_ind, col_ind):
            time = cost_matrix[i, j]
            total_time += time
```

```
            solution_text += f"  {workers[i]:<10s} → {tasks[j]:<15s} ({time:2d}
hours)\n"

        solution_text += f"\n{'=' * 79}\n"
        solution_text += f"MINIMUM TOTAL TIME: {total_time} hours\n"
        solution_text += f"{'=' * 79}\n\n"

        self.results_text.insert(tk.END, solution_text)
        self.perform_assignment_sensitivity(cost_matrix, row_ind, col_ind,
workers, tasks, total_time)

    def perform_assignment_sensitivity(self, cost_matrix, row_ind, col_ind,
workers, tasks, total_time):
        """Sensitivity analysis for assignment"""

        self.results_text.insert(tk.END, "\n╔" + "=" * 78 + "╗\n")
        self.results_text.insert(tk.END, "║" + " " * 20 + "SENSITIVITY ANALYSIS"
+ " " * 38 + "║\n")
        self.results_text.insert(tk.END, "╚" + "=" * 78 + "╝\n\n")

        # 1. OPPORTUNITY COSTS
        self.results_text.insert(tk.END, "1. OPPORTUNITY COST ANALYSIS\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n")
        self.results_text.insert(tk.END, "Additional cost if forced to use
alternative assignments\n\n")

        assignments = {(i, j) for i, j in zip(row_ind, col_ind)}
        alternatives = []

        for i in range(len(workers)):
            for j in range(len(tasks)):
                if (i, j) not in assignments:
                    test_cost = cost_matrix[i, j]
                    current_i_cost = cost_matrix[i, col_ind[i]]
                    opp_cost = test_cost - current_i_cost
                    alternatives.append((workers[i], tasks[j], test_cost,
opp_cost))

        alternatives.sort(key=lambda x: x[3])

        self.results_text.insert(tk.END, "Top 10 Alternative Assignments (lowest
opportunity cost):\n\n")
        self.results_text.insert(
            tk.END,
            f"{'Worker':<12s} {'Task':<15s} {'Time':>8s} {'Opp. Cost':>12s}\n"
```

```
        )
        self.results_text.insert(tk.END, "-" * 79 + "\n")

        for worker, task, time, opp in alternatives[:10]:
            self.results_text.insert(tk.END, f"{worker:<12s} {task:<15s}
{time:>8d} hrs {opp:>11.1f} hrs\n")

        # 2. COST TOLERANCE
        self.results_text.insert(tk.END, "\n2. COST TOLERANCE ANALYSIS\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n")
        self.results_text.insert(tk.END, "How much can assignment costs change
before solution changes?\n\n")
        self.results_text.insert(
            tk.END,
            f"{'Worker':<12s} {'→ Task':<15s} {'Current':>10s} {'Max
Increase':>15s}\n"
        )
        self.results_text.insert(tk.END, "-" * 79 + "\n")

        for i, j in zip(row_ind, col_ind):
            current_cost = cost_matrix[i, j]
            min_alt = float('inf')
            for alt_j in range(len(tasks)):
                if alt_j != j:
                    min_alt = min(min_alt, cost_matrix[i, alt_j])

            tolerance = min_alt - current_cost
            self.results_text.insert(
                tk.END,
                f"{workers[i]:<12s} → {tasks[j]:<13s} {current_cost:>10d} hrs
{tolerance:>14.1f} hrs\n"
            )

        # 3. WORKER EFFICIENCY
        self.results_text.insert(tk.END, "\n3. WORKER EFFICIENCY ANALYSIS\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n\n")

        avg_time = total_time / len(workers)
        self.results_text.insert(tk.END, f"Average time per assignment:
{avg_time:.1f} hours\n\n")
        self.results_text.insert(
            tk.END,
            f"{'Worker':<12s} {'Assigned Time':>15s} {'vs Average':>15s}
{'Rating':>15s}\n"
        )
```

```
        self.results_text.insert(tk.END, "-" * 79 + "\n")

        for i, j in zip(row_ind, col_ind):
            time = cost_matrix[i, j]
            diff = time - avg_time
            rating = "Excellent" if time < avg_time * 0.8 else ("Good" if time <
avg_time else "Needs Training")

            self.results_text.insert(
                tk.END,
                f"{workers[i]:<12s} {time:>15d} hrs {diff:>14.1f} hrs
{rating:>15s}\n"
            )

        insights = f"""

{'=' * 79}
                        MANAGERIAL INSIGHTS
{'=' * 79}

KEY FINDINGS:

1. ASSIGNMENT STABILITY:
   - High opportunity costs (>5 hrs) indicate stable assignments
   - Low opportunity costs (<2 hrs) suggest near-equivalent alternatives

2. FLEXIBILITY:
   - Workers with high cost tolerance can handle task changes better
   - Low tolerance workers are optimally matched - avoid reassignment

3. TRAINING PRIORITIES:
   - Focus training on workers performing above average time
   - Cross-train workers with narrow assignment options

RECOMMENDATIONS:

1. Implement the optimal assignment immediately
2. Create backup assignments for critical tasks (using opportunity cost analysis)
3. Provide specialized training to workers with below-average efficiency
4. Consider team-based assignments for workers with similar skills
5. Re-evaluate assignments monthly or when new workers/tasks are added

{'=' * 79}
"""
        self.results_text.insert(tk.END, insights)
```

```
    # ======================================================================
    # TRANSPORTATION PROBLEM WITH SENSITIVITY
    # ======================================================================
    def solve_transportation(self):
        """Transportation problem with VAM + UV Method + Sensitivity"""
        self.results_text.delete(1.0, tk.END)

        result = """
╔══════════════════════════════════════════════════════════════════════════╗
║                        TRANSPORTATION PROBLEM                            ║
║              (VAM + UV Method + Sensitivity Analysis)                    ║
╚══════════════════════════════════════════════════════════════════════════╝

PROBLEM: Distribute products from 10 factories to 10 warehouses at minimum cost.

Total Supply = Total Demand = 5000 units (BALANCED)

COST MATRIX ($ per unit):
═══════════════════════════════════════════════════════════════════════════
Factory    W1    W2    W3    W4    W5    W6    W7    W8    W9    W10
───────────────────────────────────────────────────────────────────────────
"""
        cost_matrix = np.array([
            [8, 12, 10, 15, 9, 11, 13, 14, 16, 10],
            [10, 9, 11, 13, 12, 8, 10, 15, 14, 11],
            [12, 8, 9, 14, 10, 13, 7, 16, 11, 12],
            [11, 7, 10, 12, 11, 14, 8, 13, 9, 10],
            [14, 13, 15, 8, 12, 16, 14, 6, 10, 13],
            [9, 11, 12, 14, 8, 10, 13, 15, 12, 9],
            [7, 13, 11, 16, 10, 9, 14, 12, 15, 11],
            [10, 12, 8, 13, 11, 12, 10, 14, 13, 9],
            [11, 10, 9, 14, 12, 11, 8, 15, 12, 10],
            [12, 11, 7, 15, 13, 10, 9, 16, 14, 8],
        ], dtype=float)

        factories = ["Beijing", "Shanghai", "Shenzhen", "Guangzhou", "Chengdu",
                     "Wuhan", "Tianjin", "Nanjing", "Hangzhou", "Suzhou"]
        warehouses = ["North", "South", "East", "West", "Central",
                      "Northeast", "Southeast", "Northwest", "Southwest",
"Midwest"]

        for i, factory in enumerate(factories):
            result += f"{factory:<10s}  "
            for j in range(len(warehouses)):
```

```
                result += f"{int(cost_matrix[i][j]):3d}  "
            result += "\n"

        result += "=" * 79 + "\n\n"
        self.results_text.insert(tk.END, result)
        self.results_text.update()

        supply = np.array([500, 600, 550, 480, 520, 470, 530, 490, 510, 350],
dtype=float)
        demand = np.array([480, 520, 500, 460, 540, 490, 510, 470, 530, 500],
dtype=float)

        # STEP 1: VAM
        self.results_text.insert(tk.END, "STEP 1: Initial Solution (Vogel's
Approximation Method)\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n\n")
        allocation, vam_cost = self.vogels_approximation_method(
            cost_matrix.copy(), supply.copy(), demand.copy()
        )
        self.results_text.insert(tk.END, f"\nInitial Cost: ${vam_cost:,.2f}\n\n")

        # STEP 2: UV Method
        self.results_text.insert(tk.END, "STEP 2: Optimization (UV/MODI
Method)\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n\n")
        optimal_allocation, optimal_cost, iterations = self.uv_method(
            cost_matrix.copy(), allocation.copy(), supply.copy(), demand.copy()
        )
        self.results_text.insert(tk.END, f"\nOptimized in {iterations}
iterations\n\n")

        # Display solution
        solution_text = """
═══════════════════════════════════════════════════════════════════════════
                        OPTIMAL TRANSPORTATION PLAN
═══════════════════════════════════════════════════════════════════════════

NON-ZERO SHIPMENTS:
"""
        for i in range(len(factories)):
            for j in range(len(warehouses)):
                if optimal_allocation[i, j] > 0.01:
                    cost = cost_matrix[i, j] * optimal_allocation[i, j]
                    solution_text += (
                        f"  {factories[i]:<12s} → {warehouses[j]:<12s}: "
```

```
                        f"{optimal_allocation[i, j]:>6.0f} units
(${cost:>8.2f})\n"
                    )

        solution_text += f"\n{'=' * 79}\n"
        solution_text += f"MINIMUM TOTAL COST: ${optimal_cost:,.2f}\n"
        solution_text += f"Cost Reduction: ${vam_cost - optimal_cost:,.2f}\n"
        solution_text += f"{'=' * 79}\n\n"

        self.results_text.insert(tk.END, solution_text)
        self.perform_transportation_sensitivity(
            cost_matrix, optimal_allocation, supply, demand, factories,
warehouses, optimal_cost
        )

    def perform_transportation_sensitivity(self, cost, allocation, supply,
demand,
                                           factories, warehouses, total_cost):
        """Sensitivity analysis for transportation"""

        self.results_text.insert(tk.END, "\n╔" + "=" * 78 + "╗\n")
        self.results_text.insert(tk.END, "║" + " " * 20 + "SENSITIVITY ANALYSIS"
+ " " * 38 + "║\n")
        self.results_text.insert(tk.END, "╚" + "=" * 78 + "╝\n\n")

        # 1. DUAL VARIABLES (u and v)
        self.results_text.insert(tk.END, "1. DUAL VARIABLES (u and v values)\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n\n")

        n, m = len(factories), len(warehouses)
        u = np.zeros(n)
        v = np.zeros(m)

        # Calculate dual variables
        basic_cells = [(i, j) for i in range(n) for j in range(m) if
allocation[i, j] > 0.01]

        for _ in range(n + m):
            for i, j in basic_cells:
                if u[i] == 0 and v[j] != 0:
                    u[i] = cost[i, j] - v[j]
                elif v[j] == 0 and u[i] != 0:
                    v[j] = cost[i, j] - u[i]
                elif u[i] == 0 and v[j] == 0:
                    v[j] = cost[i, j]
```

```
        self.results_text.insert(tk.END, "Factory Dual Variables (u):\n")
        for i, factory in enumerate(factories):
            self.results_text.insert(tk.END, f"  {factory:<12s}: u{i + 1} =
${u[i]:>6.2f}\n")

        self.results_text.insert(tk.END, "\nWarehouse Dual Variables (v):\n")
        for j, warehouse in enumerate(warehouses):
            self.results_text.insert(tk.END, f"  {warehouse:<12s}: v{j + 1} =
${v[j]:>6.2f}\n")

        # 2. REDUCED COSTS
        self.results_text.insert(tk.END, "\n2. REDUCED COSTS FOR NON-BASIC
ROUTES\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n")
        self.results_text.insert(tk.END, "Negative values indicate potential for
cost improvement\n\n")

        non_basic_routes = []
        for i in range(n):
            for j in range(m):
                if allocation[i, j] < 0.01:
                    reduced_cost = cost[i, j] - u[i] - v[j]
                    if reduced_cost < 10:  # Show interesting ones
                        non_basic_routes.append((factories[i], warehouses[j],
                                                 cost[i, j], reduced_cost))

        non_basic_routes.sort(key=lambda x: x[3])

        self.results_text.insert(tk.END, "Top 10 Alternative Routes (lowest
reduced cost):\n\n")
        self.results_text.insert(
            tk.END,
            f"{'Factory':<12s} {'→ Warehouse':<12s} {'Cost':>8s} {'Reduced
Cost':>15s}\n"
        )
        self.results_text.insert(tk.END, "-" * 79 + "\n")

        for factory, warehouse, c_val, rc in non_basic_routes[:10]:
            self.results_text.insert(
                tk.END,
                f"{factory:<12s} → {warehouse:<10s} ${c_val:>7.2f}
${rc:>14.2f}\n"
            )
```

```
        # 3. ROUTE UTILIZATION
        self.results_text.insert(tk.END, "\n3. ROUTE UTILIZATION ANALYSIS\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n\n")

        total_units = np.sum(allocation)
        avg_cost_per_unit = total_cost / total_units

        self.results_text.insert(tk.END, f"Average cost per unit:
${avg_cost_per_unit:.2f}\n\n")
        self.results_text.insert(
            tk.END,
            f"{'Factory':<12s} {'→ Warehouse':<12s} {'Units':>8s} {'$/unit':>8s}
{'Efficiency':>12s}\n"
        )
        self.results_text.insert(tk.END, "-" * 79 + "\n")

        for i in range(n):
            for j in range(m):
                if allocation[i, j] > 0.01:
                    unit_cost = cost[i, j]
                    efficiency = "Excellent" if unit_cost < avg_cost_per_unit *
0.9 else \
                        ("Good" if unit_cost < avg_cost_per_unit * 1.1 else "High
Cost")

                    self.results_text.insert(
                        tk.END,
                        f"{factories[i]:<12s} → {warehouses[j]:<10s}
{allocation[i, j]:>8.0f} "
                        f"${unit_cost:>7.2f} {efficiency:>12s}\n"
                    )

        # 4. SUPPLY/DEMAND SENSITIVITY
        self.results_text.insert(tk.END, "\n4. SUPPLY/DEMAND SENSITIVITY\n")
        self.results_text.insert(tk.END, "=" * 79 + "\n\n")

        self.results_text.insert(tk.END, "Factory Supply Utilization:\n")
        self.results_text.insert(
            tk.END,
            f"{'Factory':<12s} {'Shipped':>10s} {'Capacity':>10s} {'%
Used':>10s}\n"
        )
        self.results_text.insert(tk.END, "-" * 79 + "\n")

        for i in range(n):
```

```
            shipped = np.sum(allocation[i, :])
            pct = (shipped / supply[i]) * 100
            self.results_text.insert(
                tk.END,
                f"{factories[i]:<12s} {shipped:>10.0f} {supply[i]:>10.0f}
{pct:>9.1f}%\n"
            )

        self.results_text.insert(tk.END, "\nWarehouse Demand Fulfillment:\n")
        self.results_text.insert(
            tk.END,
            f"{'Warehouse':<12s} {'Received':>10s} {'Demand':>10s} {'%
Filled':>10s}\n"
        )
        self.results_text.insert(tk.END, "-" * 79 + "\n")

        for j in range(m):
            received = np.sum(allocation[:, j])
            pct = (received / demand[j]) * 100
            self.results_text.insert(
                tk.END,
                f"{warehouses[j]:<12s} {received:>10.0f} {demand[j]:>10.0f}
{pct:>9.1f}%\n"
            )

        # FIXED: compute active routes safely
        active_routes = sum(1 for i in range(n) for j in range(m) if
allocation[i, j] > 0.01)

        insights = f"""

{'=' * 79}
                        MANAGERIAL INSIGHTS
{'=' * 79}

KEY FINDINGS:

1. OPTIMAL ROUTES:
   - {active_routes} active routes out of {n * m} possible
   - Routes with negative reduced costs should be considered if situation changes
   - High-cost routes may need renegotiation or alternative suppliers

2. ALTERNATIVE ROUTES:
   - Routes with reduced cost near 0 are good backup options
   - Large positive reduced costs indicate routes to avoid
```

```
3. CAPACITY PLANNING:
   - All facilities at 100% utilization indicates optimal resource use
   - Consider expanding facilities with consistent high demand

RECOMMENDATIONS:

1. Implement the optimal shipping plan immediately
2. Negotiate lower rates for high-volume, high-cost routes
3. Establish backup routes from alternatives with low reduced costs
4. Monitor fuel prices and recalculate if costs change >5%
5. Consider warehouse locations - clusters of high-cost routes suggest need
6. Re-optimize monthly or when supply/demand changes by >10%

{'=' * 79}
"""
        self.results_text.insert(tk.END, insights)

    def vogels_approximation_method(self, cost, supply, demand):
        """VAM for initial basic feasible solution"""
        rows, cols = cost.shape
        allocation = np.zeros((rows, cols))

        while np.sum(supply) > 0.01 and np.sum(demand) > 0.01:
            # Calculate penalties
            row_penalties = []
            for i in range(rows):
                if supply[i] > 0.01:
                    available = [cost[i, j] for j in range(cols) if demand[j] >
0.01]
                    if len(available) >= 2:
                        available.sort()
                        row_penalties.append((available[1] - available[0], i,
'row'))
                    elif len(available) == 1:
                        row_penalties.append((available[0], i, 'row'))

            col_penalties = []
            for j in range(cols):
                if demand[j] > 0.01:
                    available = [cost[i, j] for i in range(rows) if supply[i] >
0.01]
                    if len(available) >= 2:
                        available.sort()
```

```
                        col_penalties.append((available[1] - available[0], j,
'col'))
                    elif len(available) == 1:
                        col_penalties.append((available[0], j, 'col'))

            if not row_penalties and not col_penalties:
                break

            # Find max penalty
            all_penalties = row_penalties + col_penalties
            all_penalties.sort(reverse=True)
            penalty, idx, ptype = all_penalties[0]

            # Allocate to min cost cell
            if ptype == 'row':
                i = idx
                min_cost = float('inf')
                j_min = -1
                for j in range(cols):
                    if demand[j] > 0.01 and cost[i, j] < min_cost:
                        min_cost = cost[i, j]
                        j_min = j
                j = j_min
            else:
                j = idx
                min_cost = float('inf')
                i_min = -1
                for i in range(rows):
                    if supply[i] > 0.01 and cost[i, j] < min_cost:
                        min_cost = cost[i, j]
                        i_min = i
                i = i_min

            # Allocate
            allocated = min(supply[i], demand[j])
            allocation[i, j] = allocated
            supply[i] -= allocated
            demand[j] -= allocated

        total_cost = np.sum(allocation * cost)
        return allocation, total_cost

    def uv_method(self, cost, allocation, supply, demand):
        """UV/MODI method for optimization"""
        rows, cols = cost.shape
```

```
        iteration = 0
        max_iterations = 100

        while iteration < max_iterations:
            iteration += 1

            basic_cells = [(i, j) for i in range(rows) for j in range(cols)
                           if allocation[i, j] > 0.01]

            u = [None] * rows
            v = [None] * cols
            u[0] = 0

            for _ in range(rows + cols):
                for i, j in basic_cells:
                    if u[i] is not None and v[j] is None:
                        v[j] = cost[i, j] - u[i]
                    elif v[j] is not None and u[i] is None:
                        u[i] = cost[i, j] - v[j]

            min_opp = 0
            entering_cell = None

            for i in range(rows):
                for j in range(cols):
                    if allocation[i, j] < 0.01:
                        if u[i] is not None and v[j] is not None:
                            opp_cost = cost[i, j] - u[i] - v[j]
                            if opp_cost < min_opp:
                                min_opp = opp_cost
                                entering_cell = (i, j)

            if min_opp >= -0.01:
                break

            loop = self.find_loop(allocation, entering_cell, basic_cells)

            if loop and len(loop) >= 4:
                theta = float('inf')
                for idx in range(1, len(loop), 2):
                    i, j = loop[idx]
                    theta = min(theta, allocation[i, j])

                for idx, (i, j) in enumerate(loop):
                    if idx % 2 == 0:
```

```
                        allocation[i, j] += theta
                    else:
                        allocation[i, j] -= theta
            else:
                break

        total_cost = np.sum(allocation * cost)
        return allocation, total_cost, iteration

    def find_loop(self, allocation, entering_cell, basic_cells):
        """Find closed loop for pivot operation"""
        rows, cols = allocation.shape
        entering_i, entering_j = entering_cell

        row_neighbors = {i: [] for i in range(rows)}
        col_neighbors = {j: [] for j in range(cols)}

        for i, j in basic_cells:
            row_neighbors[i].append(j)
            col_neighbors[j].append(i)

        row_neighbors[entering_i].append(entering_j)
        col_neighbors[entering_j].append(entering_i)

        def dfs(ci, cj, path, visited, is_horiz):
            if len(path) >= 4 and ci == entering_i and cj == entering_j:
                return path

            if (ci, cj) in visited and (ci, cj) != entering_cell:
                return None

            visited.add((ci, cj))

            if is_horiz:
                for nj in row_neighbors[ci]:
                    if nj != cj and (len(path) < 2 or nj != path[-2][1]):
                        result = dfs(ci, nj, path + [(ci, nj)], visited.copy(),
False)
                        if result:
                            return result
            else:
                for ni in col_neighbors[cj]:
                    if ni != ci and (len(path) < 2 or ni != path[-2][0]):
                        result = dfs(ni, cj, path + [(ni, cj)], visited.copy(),
True)
```

```
                        if result:
                            return result

            return None

        return dfs(entering_i, entering_j, [(entering_i, entering_j)], set(),
True)


# ======================================================================
# MAIN EXECUTION
# ======================================================================
if __name__ == "__main__":
    root = tk.Tk()
    app = ORSolverApp(root)
    root.mainloop()
```

Conclusion

This project demonstrated how Operations Research techniques can significantly improve decision-making for TechElectro Manufacturing across production, workforce assignment, and distribution. The Simplex Method produced a profit-maximizing production plan, showing that only Smart Watches and Keyboards should be manufactured under current resource constraints. Sensitivity analysis further identified critical bottlenecks and evaluated the impact of new constraints and new products on feasibility and profitability.

The Assignment model minimized total assembly time by optimally pairing workers with tasks, while efficiency and opportunity-cost analyses revealed areas for skill development and operational flexibility. The Transportation model successfully reduced logistics cost using VAM and UV/MODI, providing an optimized shipment plan and detailed sensitivity insights that highlight cost-effective and high-risk routes.

Overall, the integrated OR system offers a reliable analytical framework that enhances production planning, labor utilization, and distribution efficiency. By supporting data-driven decisions, the solution strengthens TechElectro's operational performance and competitiveness in the electronics manufacturing industry.