Assignment 2 – Optimization under Uncertainty

Hariaksh Pandya- s2692608

March 4, 2025

Contents

1 Train Seat Allocation

1.1 Problem Overview

ScotRail is redesigning its train for the Edinburgh–Glasgow route. The train's interior can accommodate up to 200 passengers if it is entirely configured with Economy seats. In practice, however, the train is partitioned into a mix of Economy, Business, and First-Class seats. The design challenge arises because:

- A **Business seat** occupies 1.5 times the space of an Economy seat.
- A **First-Class seat** occupies 2 times the space of an Economy seat.

Thus, we measure the train's capacity in *Economy-equivalent* seats.

Demand is uncertain and represented by four scenarios (indexed by $\xi \in \{1, 2, 3, 4\}$), each with probability p_ξ, and each scenario provides the maximum number of seats that can be sold in each class:

$$(d_{E,\xi},\, d_{B,\xi},\, d_{F,\xi}).$$

The goal is to decide on the train partition (i.e., the number of seats of each type) and, after the scenario is revealed, the number of seats sold in each class, to maximize the expected profit.

1.2 Parameters and Data

We denote:

- r_E: Profit (or revenue) per Economy seat.
- $r_B = 2r_E$: Profit per Business seat.
- $r_F = 3r_E$: Profit per First-Class seat.

Often one sets $r_E = 1$, but we retain r_E symbolically.

The four demand scenarios are provided in Table 1.

Table 1: Demand Scenarios for Seat Sales

Scenario (ξ)	p_ξ	$d_{E,\xi}$	$d_{B,\xi}$	$d_{F,\xi}$
1	0.4	200	60	25
2	0.3	180	40	20
3	0.2	175	25	10
4	0.1	150	10	5

1.3 Decision Variables

1.3.1 First-Stage Decisions (Train Partition)

Before the demand is realised, the operator decides the train's configuration:

$$\begin{aligned} x_E &\geq 0, \quad \text{number of Economy seats allocated,} \\ x_B &\geq 0, \quad \text{number of Business seats allocated,} \\ x_F &\geq 0, \quad \text{number of First-Class seats allocated.} \end{aligned}$$

These choices determine the physical partition of the train.

1.3.2 Second-Stage Decisions (Sales Decisions)

After the scenario ξ is realised, the number of seats sold in each class is determined:

$$\begin{aligned} y_{E,\xi} &\geq 0, \quad \text{Economy seats sold in scenario } \xi, \\ y_{B,\xi} &\geq 0, \quad \text{Business seats sold in scenario } \xi, \\ y_{F,\xi} &\geq 0, \quad \text{First-Class seats sold in scenario } \xi. \end{aligned}$$

These decisions represent the recourse actions once the uncertain demand materialises.

1.4 Model Formulation

1.4.1 Capacity Constraint

Since Business seats consume 1.5 times and First-Class seats 2 times the space of an Economy seat, the total number of Economy-equivalent seats allocated must not exceed the train's capacity:

$$x_E + 1.5\,x_B + 2\,x_F \leq 200.$$

This constraint ensures that the physical limits of the train are respected.

1.4.2 Sales and Demand Constraints

For each scenario ξ, the following constraints are imposed:

No Overselling:

$$y_{E,\xi} \leq x_E, \quad y_{B,\xi} \leq x_B, \quad y_{F,\xi} \leq x_F.$$

This prevents selling more seats than have been allocated.

Demand Limits:

$$y_{E,\xi} \leq d_{E,\xi}, \quad y_{B,\xi} \leq d_{B,\xi}, \quad y_{F,\xi} \leq d_{F,\xi}.$$

This ensures that the number of seats sold does not exceed the demand in scenario ξ.

1.4.3 Objective Function

The profit in scenario ξ is given by:

$$\text{Profit}_\xi = r_E\, y_{E,\xi} + 2\, r_E\, y_{B,\xi} + 3\, r_E\, y_{F,\xi}.$$

Thus, the expected profit is:

$$\max \quad \sum_{\xi=1}^{4} p_\xi \left(r_E\, y_{E,\xi} + 2\, r_E\, y_{B,\xi} + 3\, r_E\, y_{F,\xi}\right).$$

If we assume $r_E = 1$, the objective simplifies to:

$$\max \quad \sum_{\xi=1}^{4} p_\xi \left(y_{E,\xi} + 2\, y_{B,\xi} + 3\, y_{F,\xi}\right).$$

1.4.4 First-Stage Cost Consideration

Typically, first-stage decisions might incur a setup cost (e.g., cost for installing seats or partitioning the train). However, the problem statement does not provide any such cost. We therefore assume that the cost of setting up or acquiring the seats is *zero*. In this model, the first-stage decisions influence the outcome only through the constraints on the second-stage decisions.

If setup costs were included, we introduce the parameter c_i, which represents the cost per seat type i. Let c_E, c_B, c_F denote the setup costs for Economy, Business, and First-Class seats.

setup cost:

$$\sum_{i\in\{E,B,F\}} c_i x_i$$

where x_i represents the number of seats allocated for seat type i.

Objective function:

$$\min \sum_{i\in\{E,B,F\}} c_i x_i - \sum_{\xi=1}^{4} p_\xi \Big(r_E y_{E,\xi} + 2 r_E y_{B,\xi} + 3 r_E y_{F,\xi} \Big)$$

where:

- The first term represents the first-stage setup cost, penalising more expensive train configurations.
- The second term represents the expected second-stage profit, with revenue contributions from Economy, Business, and First-Class ticket sales across all demand scenarios.

However, as previously mentioned and in accordance with the problem statement, there is no setup cost in this instance. Consequently, the first-stage objective reduces to zero due to the absence of any associated costs. This leaves us solely with second-stage decisions, also known as recourse decisions. Furthermore, since minimising a negative value is equivalent to maximising its positive counterpart, the problem ultimately simplifies to maximising the expected profit/revenue.

Since there are no setup costs in this case, the optimisation reduces to maximising revenue. Essentially, while the problem is initially framed as a profit maximisation model, the absence of setup costs simplifies it to a revenue maximisation problem.

$$c_E = c_B = c_F = 0$$

which effectively eliminates the first-stage cost term from the objective function. As a result, the optimisation focuses solely on maximising expected revenue in the second stage, leading to the final simplified objective:

$$\max \sum_{\xi=1}^{4} p_\xi \Big(y_{E,\xi} + 2y_{B,\xi} + 3y_{F,\xi} \Big).$$

1.5 Recourse Feasibility and Analysis

Any allocation (x_E, x_B, x_F) satisfying

$$x_E + 1.5\, x_B + 2\, x_F \leq 200$$

allows a feasible second-stage decision by setting, for each class $i \in \{E, B, F\}$ and scenario ξ,

$$y_{i,\xi} = \min\{x_i,\, d_{i,\xi}\}.$$

Thus, no matter the allocation, the second-stage problem is always feasible. This property is *relatively complete recourse.*

However, the same cannot be said for complete recourse. If we assign any negative values to the first stage variables, we cannot satisfy our constraints; this occurs because the variables, and subsequently the constraints, turn negative. Since demand cannot be negative, complete recourse is not possible in such scenarios. It's important to emphasise that if we define our variables and logically formulate our constraints, ensuring they remain positive, the constraints will also be positive and solvable for the first stage. In this case, we can manage to solve the second stage, which may lead to complete recourse.

Furthermore, in practice, solving the model with a solver such as Gurobi often yields an optimal partition that favors higher-revenue seats (Business and First-Class) when the demand and scenario probabilities justify it. By comparing the stochastic solution with a solution derived from average demand, one can compute the *Value of the Stochastic Solution (VSS).* Similarly, by comparing with the scenario-specific perfect-information solutions, the *Expected Value of Perfect Information (EVPI)* can be obtained.

1.6 Code - Gurobi

```
import gurobipy as gb
from gurobipy import GRB

# Define scenario data: Each scenario xi has a probability and demands.
scenarios = {
    1: {"prob": 0.4, "demE": 200, "demB": 60,  "demF": 25},
    2: {"prob": 0.3, "demE": 180, "demB": 40,  "demF": 20},
    3: {"prob": 0.2, "demE": 175, "demB": 25,  "demF": 10},
    4: {"prob": 0.1, "demE": 150, "demB": 10,  "demF": 5}
}

r_E = 1.0  # Base profit per Economy seat

##############################
# Full Two-Stage Model
##############################
m_full = gb.Model("Full_2Stage")
xE = m_full.addVar(lb=0, vtype=GRB.CONTINUOUS, name="xE")
xB = m_full.addVar(lb=0, vtype=GRB.CONTINUOUS, name="xB")
xF = m_full.addVar(lb=0, vtype=GRB.CONTINUOUS, name="xF")

yE = {}
yB = {}
yF = {}
for xi in scenarios:
    yE[xi] = m_full.addVar(lb=0, vtype=GRB.CONTINUOUS, name=f"yE_{xi}")
    yB[xi] = m_full.addVar(lb=0, vtype=GRB.CONTINUOUS, name=f"yB_{xi}")
    yF[xi] = m_full.addVar(lb=0, vtype=GRB.CONTINUOUS, name=f"yF_{xi}")

m_full.addConstr(xE + 1.5*xB + 2*xF <= 200, "Capacity")
for xi, data in scenarios.items():
    m_full.addConstr(yE[xi] <= xE, name=f"NoOversellE_{xi}")
    m_full.addConstr(yB[xi] <= xB, name=f"NoOversellB_{xi}")
    m_full.addConstr(yF[xi] <= xF, name=f"NoOversellF_{xi}")

    m_full.addConstr(yE[xi] <= data["demE"], name=f"DemE_{xi}")
    m_full.addConstr(yB[xi] <= data["demB"], name=f"DemB_{xi}")
    m_full.addConstr(yF[xi] <= data["demF"], name=f"DemF_{xi}")

obj_full = gb.LinExpr()
for xi, data in scenarios.items():
    p_xi = data["prob"]
    obj_full += p_xi * (r_E*yE[xi] + 2*r_E*yB[xi] + 3*r_E*yF[xi])
```

```
m_full.setObjective(obj_full, GRB.MAXIMIZE)
m_full.optimize()

# Print full model results
if m_full.status == GRB.OPTIMAL:
    print("\nOptimal Objective (Expected Profit): {:.4f}".format(m_full.objVal))
    print("Optimal seat allocation:")
    print("  xE = {:.4f}".format(xE.X))
    print("  xB = {:.4f}".format(xB.X))
    print("  xF = {:.4f}".format(xF.X))
    for xi in sorted(scenarios):
        print("\nScenario {} (prob = {}):".format(xi, scenarios[xi]['prob']))
        print("  yE_{} = {:.4f}".format(xi, yE[xi].X))
        print("  yB_{} = {:.4f}".format(xi, yB[xi].X))
        print("  yF_{} = {:.4f}".format(xi, yF[xi].X))
else:
    print("No optimal solution found for full model")

vS = m_full.objVal

#############################
# Mean-Value (MV) Model
#############################
# Compute weighted average demands
dE_MV = sum(data["prob"] * data["demE"] for data in scenarios.values())
dB_MV = sum(data["prob"] * data["demB"] for data in scenarios.values())
dF_MV = sum(data["prob"] * data["demF"] for data in scenarios.values())

m_MV = gb.Model("MV_Model")
xE_MV = m_MV.addVar(lb=0, vtype=GRB.CONTINUOUS, name="xE_MV")
xB_MV = m_MV.addVar(lb=0, vtype=GRB.CONTINUOUS, name="xB_MV")
xF_MV = m_MV.addVar(lb=0, vtype=GRB.CONTINUOUS, name="xF_MV")
yE_MV = m_MV.addVar(lb=0, vtype=GRB.CONTINUOUS, name="yE_MV")
yB_MV = m_MV.addVar(lb=0, vtype=GRB.CONTINUOUS, name="yB_MV")
yF_MV = m_MV.addVar(lb=0, vtype=GRB.CONTINUOUS, name="yF_MV")

m_MV.addConstr(xE_MV + 1.5*xB_MV + 2*xF_MV <= 200, "Capacity_MV")
m_MV.addConstr(yE_MV <= xE_MV, "NoOversellE_MV")
m_MV.addConstr(yB_MV <= xB_MV, "NoOversellB_MV")
m_MV.addConstr(yF_MV <= xF_MV, "NoOversellF_MV")
m_MV.addConstr(yE_MV <= dE_MV, "DemE_MV")
m_MV.addConstr(yB_MV <= dB_MV, "DemB_MV")
m_MV.addConstr(yF_MV <= dF_MV, "DemF_MV")

obj_MV = r_E*yE_MV + 2*r_E*yB_MV + 3*r_E*yF_MV
```

```
m_MV.setObjective(obj_MV, GRB.MAXIMIZE)
m_MV.optimize()

# Store the MV first-stage solution
xE_MV_val = xE_MV.X
xB_MV_val = xB_MV.X
xF_MV_val = xF_MV.X

# Evaluate the MV solution in the full model
vMV_eval = 0.0
for xi, data in scenarios.items():
    yE_val = min(xE_MV_val, data["demE"])
    yB_val = min(xB_MV_val, data["demB"])
    yF_val = min(xF_MV_val, data["demF"])
    vMV_eval += data["prob"] * (r_E*yE_val + 2*r_E*yB_val + 3*r_E*yF_val)

#############################
# Perfect Information (PI) Model
#############################
vPI = 0.0
for xi, data in scenarios.items():
    m_PI = gb.Model(f"PI_scenario_{xi}")
    yE_PI = m_PI.addVar(lb=0, vtype=GRB.CONTINUOUS, name="yE_PI")
    yB_PI = m_PI.addVar(lb=0, vtype=GRB.CONTINUOUS, name="yB_PI")
    yF_PI = m_PI.addVar(lb=0, vtype=GRB.CONTINUOUS, name="yF_PI")
    m_PI.addConstr(yE_PI + 1.5*yB_PI + 2*yF_PI <= 200, "Capacity_PI")
    m_PI.addConstr(yE_PI <= data["demE"], "DemE_PI")
    m_PI.addConstr(yB_PI <= data["demB"], "DemB_PI")
    m_PI.addConstr(yF_PI <= data["demF"], "DemF_PI")
    m_PI.setObjective(r_E*yE_PI + 2*r_E*yB_PI + 3*r_E*yF_PI, GRB.MAXIMIZE)
    m_PI.optimize()
    if m_PI.status == GRB.OPTIMAL:
        vPI += data["prob"] * m_PI.objVal

#############################
# Compute VSS and EVPI
#############################
VSS = vS - vMV_eval
EVPI = vPI - vS

print("\n----------------------------------------")
print("Final Results:")
print("Full 2-Stage Model Expected Profit: {:.4f}".format(vS))
print("Mean-Value Evaluated Profit: {:.4f}".format(vMV_eval))
print("Perfect Information Expected Profit: {:.4f}".format(vPI))
print("Value of the Stochastic Solution (VSS): {:.4f}".format(VSS))
print("Expected Value of Perfect Information (EVPI): {:.4f}".format(EVPI))
print("----------------------------------------")
```

1.7 Output

```
Optimal Objective (Expected Profit): 219.0000
Optimal seat allocation:
  xE = 122.5000
  xB = 25.0000
  xF = 20.0000

Scenario 1 (prob = 0.4):
  yE_1 = 122.5000
  yB_1 = 25.0000
  yF_1 = 20.0000

Scenario 2 (prob = 0.3):
  yE_2 = 122.5000
  yB_2 = 25.0000
  yF_2 = 20.0000

Scenario 3 (prob = 0.2):
  yE_3 = 122.5000
  yB_3 = 25.0000
  yF_3 = 10.0000

Scenario 4 (prob = 0.1):
  yE_4 = 122.5000
  yB_4 = 10.0000
  yF_4 = 5.0000

  ----------------------------------------
Final Results:
Full 2-Stage Model Expected Profit: 219.0000
Mean-Value Evaluated Profit: 215.9500
Perfect Information Expected Profit: 237.0000
Value of the Stochastic Solution (VSS): 3.0500
Expected Value of Perfect Information (EVPI): 18.0000
----------------------------------------
```

In practice, solving this model with the demands from Table will likely yield a seat mix that favors more Business and First-Class seats, especially under scenarios with relatively high Business/First demands and large probabilities.

2 Two-Stage Assignment Problem

2.1 Overview

In a standard assignment problem, we have n workers (V_1) and n tasks (V_2), forming a bipartite graph $(V_1 \cup V_2, E)$. Each edge $e = (i, j) \in E$ means worker $i \in V_1$ can be assigned to task $j \in V_2$ at some cost. The classical goal is to pick exactly one edge per worker and task (forming a perfect matching) at minimal cost.

2.2 Two-Stage Extension

We now separate the decision into two stages:

1. **First Stage (Partial Matching):** We must select exactly K edges before cost uncertainty is resolved, incurring a known cost $\mathrm{c1}_e$ per chosen edge e. We also ensure no worker or task is used more than once among these K edges.

2. **Second Stage (Completion):** After scenario $\mathrm{sc} \in \Omega$ is observed (each with probability π_{sc}), we add enough edges to form a full matching, incurring scenario-dependent costs $\mathrm{c2}_e^{\mathrm{sc}}$ for these newly added edges.

Hence, the objective is the sum of the first-stage cost plus the expected second-stage cost over all scenarios.

2.3 Parameters and Notation

Sets and Graph

- V_1: workers, $|V_1| = n$,
- V_2: tasks, $|V_2| = n$,
- $E \subseteq V_1 \times V_2$: feasible edges (i, j).

Costs and Scenarios

- $\mathrm{c1}_e$: first-stage cost for edge e.
- $\mathrm{c2}_e^{\mathrm{sc}}$: second-stage cost for edge e under scenario $\mathrm{sc} \in \Omega$.
- π_{sc}: probability of scenario sc, with $\sum_{\mathrm{sc}\in\Omega} \pi_{\mathrm{sc}} = 1$.
- $K < n$: number of edges chosen in the first stage.

2.4 Decision Variables

First Stage:

$$a_e \in \{0,1\} \quad \forall e \in E,$$

where $a_e = 1$ means edge e is picked in the first stage. We impose:

$$\sum_{e\in E} a_e = K, \quad \sum_{e\in\delta(v)} a_e \le 1 \quad \forall\, v \in V_1 \cup V_2.$$

This ensures exactly K edges, and no vertex is used more than once in the first stage.

Second Stage (scenario-based): For each scenario sc $\in \Omega$,

$$z_{e,\mathrm{sc}} \in \{0,1\} \quad \forall e \in E,$$

where $z_{e,\mathrm{sc}} = 1$ means edge e is chosen in the second stage for scenario sc. The constraints ensure a perfect matching across the union of first- and second-stage edges:

$$\sum_{e\in\delta(v)} \big(a_e + z_{e,\mathrm{sc}}\big) = 1, \quad \forall v \in V_1 \cup V_2,\ \forall\, \mathrm{sc} \in \Omega.$$

2.5 Mathematical Formulation

The two-stage stochastic program is:

$$\min\Big[\sum_{e\in E} \mathrm{c1}_e\, a_e\Big] + \sum_{\mathrm{sc}\in\Omega} \pi_{\mathrm{sc}}\Big(\min\sum_{e\in E} \mathrm{c2}_e^{\mathrm{sc}}\, z_{e,\mathrm{sc}}\Big),$$

subject to:

$$\sum_{e\in E} a_e = K, \quad \sum_{e\in\delta(v)} a_e \le 1 \quad \forall v, \quad a_e \in \{0,1\},$$

and for each scenario sc:

$$\sum_{e\in\delta(v)} \big(a_e + z_{e,\mathrm{sc}}\big) = 1 \quad \forall v \in V_1 \cup V_2, \quad z_{e,\mathrm{sc}} \in \{0,1\}.$$

If any feasible a_e leaves no way to pick $z_{e,\mathrm{sc}}$ for some scenario sc, then recourse fails. Otherwise, the second stage always completes a perfect matching. This model can exhibit or fail to exhibit *relatively complete recourse* depending on the bipartite graph structure.

2.6 Recourse Analysis

Let us build upon the previous statement regarding relative complete recourse. In the context of the two-stage assignment problem, relative complete recourse requires that, regardless of the first-stage decisions, a feasible second-stage solution must always exist in every scenario. However, this property does not always hold, and we provide a counterexample to illustrate a case where certain first-stage decisions result in infeasibility in the second stage.

Consider an assignment problem involving three workers, $W = \{A, B, C\}$, and three tasks, $T = \{1, 2, 3\}$. The set of feasible worker-task assignments is given by:

$$E = \{(A, 1), (A, 2), (B, 2), (B, 3), (C, 1), (C, 3)\}.$$

Each worker may only be assigned to tasks for which an edge exists in E. The problem is solved in two stages:

- In the **first stage**, we must select exactly two edges, effectively pre-assigning some workers to tasks.
- In the **second stage**, once the uncertainty is revealed, we complete the assignment while satisfying the constraints.

The uncertainty is introduced through two possible cost scenarios:

1. **Scenario 1**: The cost of assigning worker A to task 2 increases significantly.
2. **Scenario 2**: The cost of assigning worker B to task 2 increases significantly.

Each scenario occurs with equal probability, i.e., $P_1 = 0.5$ and $P_2 = 0.5$.

To demonstrate the lack of relative complete recourse, consider the following first-stage decision:

$$(A, 1) \quad \text{and} \quad (B, 3).$$

This pre-assigns:

- Worker A to task 1.
- Worker B to task 3.

This decision leaves only worker C to be assigned in the second stage. However, worker C can only be assigned to tasks 1 or 3, both of which are already occupied. This results in an infeasible second-stage problem, as no valid assignment exists for worker C.

Mathematically, the first-stage decisions are represented as:

$$a_{A,1} = 1, \quad a_{B,3} = 1, \quad a_{A,2} = 0, \quad a_{B,2} = 0, \quad a_{C,1} = 0, \quad a_{C,3} = 0.$$

In the second stage, the feasibility constraint requires that each worker be assigned to exactly one task. Specifically, worker C must satisfy:

$$z_{C,1} + z_{C,3} = 1.$$

However, since both tasks 1 and 3 are already occupied by workers A and B, there is no feasible solution for worker C, making the problem infeasible.

This counterexample clearly illustrates that the principle of relative complete recourse is not applicable in the two-stage assignment problem. By selecting specific first-stage decisions, one can create scenarios in which the second stage becomes infeasible due to limitations imposed on task allocations. Consequently, it is essential to exercise caution when devising first-stage decisions to guarantee the availability of feasible second-stage solutions across all potential realisations of uncertainty.

3 Two-Stage Supply Chain Model

3.1 Problem Overview

We have a supply chain problem where in the first stage, we decide how much capacity or inventory x_n to install at each facility or city n. In the second stage, uncertain demands arise in scenario k, and we pay additional costs for shipping, expansion, or shortages.

3.2 Code

```
#---Attempt no. I don't remember---
import gurobipy as gp
from gurobipy import GRB
# --- Read input data from provided modules ---
import ReadData  # ReadData.py loads data.py into variables
cities = ReadData.cities           # list of city names
scenarios = ReadData.scenarios     # list of scenario names
theta = ReadData.theta             # first-stage cost per unit to each city
theta_s = ReadData.theta_s         # second-stage cost per unit to each city
h = ReadData.h                     # cost per unit of leftover (unused inventory)
g = ReadData.g                     # cost per unit of shortage (unmet demand)
I = ReadData.I                     # initial central inventory
Yn = ReadData.Yn                   # initial inventory at each city n
demand = ReadData.demand           # demand[(n,k)] for city n under scenario k
prob = 1.0/len(scenarios)         # probability of each scenario (equal probability)
# --- Define optimization model ---
model = gp.Model("TwoStageStochasticInventory")
model.Params.OutputFlag = 0  # mute solver output for clarity
# First-stage decision variables: x[n] for shipments to cities
x = model.addVars(cities, name="x", lb=0)
# Second-stage variables for each scenario and city
y = model.addVars(cities, scenarios, name="y", lb=0)            # shipment in recourse
leftover = model.addVars(cities, scenarios, name="leftover", lb=0)
shortage = model.addVars(cities, scenarios, name="shortage", lb=0)
# --- Constraints ---
# 1. Central inventory usage in first stage
model.addConstr(gp.quicksum(x[n] for n in cities) <= I, name="FirstStageInvLimit")
# 2. Second-stage center inventory limit per scenario
total_first_stage = gp.quicksum(x[n] for n in cities)
for k in scenarios:
    model.addConstr(gp.quicksum(y[n,k] for n in cities) <= I - total_first_stage,
                    name=f"CenterInvLimit_{k}")

# 3. Demand satisfaction constraints for each city and scenario
for k in scenarios:
    for n in cities:
        # (If (initial + first-stage + second-stage) exceeds demand, the surplus
        #is leftover)
        model.addConstr(leftover[n,k] >= Yn[n] + x[n] + y[n,k] - demand[(n, k)],
                        name=f"LeftoverDef_{n}_{k}")
        # (If demand exceeds (initial + first-stage + second-stage), the shortfall
        #is shortage)
        model.addConstr(shortage[n,k] >= demand[(n, k)] - (Yn[n] + x[n] + y[n,k]),
                        name=f"ShortageDef_{n}_{k}")

# --- Objective: minimize total expected cost ---
# First-stage cost
first_stage_cost = gp.quicksum(theta[n] * x[n] for n in cities)
# Expected second-stage cost (sum over scenarios of probability * recourse cost)
second_stage_cost = gp.quicksum(prob * (theta_s[n]*y[n,k] + h*leftover[n,k] +
g*shortage[n,k])
                                for n in cities for k in scenarios)
model.setObjective(first_stage_cost + second_stage_cost, GRB.MINIMIZE)
# --- Solve the model ---
model.optimize()
# --- Output results ---
if model.Status == GRB.OPTIMAL:
    opt_obj = model.ObjVal
    print(f"Optimal objective value: {opt_obj:.2f}")
    print("Optimal first-stage decisions (shipments to cities):")
    for n in cities:
        print(f"  x[{n}] = {x[n].X:.2f}")
    print(f"Time taken to solve [s]: {model.Runtime:.4f}")
else:
    print("Optimization was not successful. Status code:", model.Status)
```

3.3 Output

```
Optimal objective value: 20200.57
Optimal first-stage decisions (shipments to cities):
  x[C0] = 12.10
  x[C1] = 24.79
  x[C2] = 15.84
  x[C3] = 13.16
  x[C4] = 14.12
  x[C5] = 27.07
  x[C6] = 27.03
  x[C7] = 29.81
  x[C8] = 30.70
  x[C9] = 26.56
```

```
Time taken to solve [s]: 0.1270
```

note: each $x[Cn]$ equals the minimum demand shortfall for city n across all scenarios.

4 Misinformation containment

4.1 Introduction and Overview

This report provides an in-depth explanation of a two-stage stochastic programming model used to minimise the spread of misinformation in social media networks. In this context, the goal is to select a subset of network edges to remove under a fixed budget so as to limit the propagation of harmful rumours. The model is built on two stages:

- In the **first stage**, we decide which edges to remove (the interdiction decision), incurring a cost for each removal.
- In the **second stage**, after uncertainty (different propagation scenarios) is realised, we determine whether each target node is protected or not.

The objective is to minimise the expected damage caused by rumours reaching key target nodes.

4.2 Model Formulation

4.2.1 Objective Function

Let:

- N be the set of scenarios (with $|N|$ scenarios, assumed equally likely so that each has weight $1/|N|$).
- R be the set of rumours.
- T be the set of target nodes.
- For each rumour $r \in R$, $d_r > 0$ denotes the damage incurred if r reaches a target.

For each scenario $k \in N$, and each target node $t \in T$ with respect to a given rumour r, let

$$z_t^{k,r} \in \{0,1\}$$

be a binary variable that equals 1 if, in scenario k, target t is reached by rumour r (i.e. the misinformation propagates to t); otherwise, it is 0.

Thus, the total damage in scenario k is

$$\sum_{r\in R}\sum_{t\in T} d_r\, z_t^{k,r}.$$

The overall objective function is to minimise the average damage across all scenarios:

$$\min \quad \frac{1}{|N|}\sum_{k\in N}\sum_{r\in R}\sum_{t\in T} d_r\, z_t^{k,r}.$$

This objective drives the model to choose edge removals such that as many target nodes as possible remain safe (i.e. $z_t^{k,r} = 0$) in every scenario.

4.2.2 Decision Variables

First-Stage Variables (Edge Removals): For every edge $e \in E$ in the social network, let

$$x_e \in \{0, 1\},$$

where $x_e = 1$ indicates that edge e is removed. Removing an edge effectively blocks any rumour path that passes through it. These are the decisions made before the actual propagation scenario is observed.

Second-Stage Variables (Outcome Indicators): For each scenario $k \in N$, for each rumour $r \in R$ and each target $t \in T$, define

$$z_t^{k,r} \in \{0, 1\}.$$

If $z_t^{k,r} = 0$, it means target t is safe from rumour r in scenario k; if $z_t^{k,r} = 1$, then the rumour reaches t and causes damage d_r.

4.2.3 Constraints

Path-Cutting Constraints: For every scenario k, for every rumour r, for every target t, and for every path $P \in P^k(r,t)$ (where $P^k(r,t)$ denotes the set of all paths from a rumour source $s \in S(r)$ to target t in scenario k), the following inequality must hold:

$$\sum_{e\in P} x_e \geq 1 - z_t^{k,r}.$$

This ensures that if we wish for $z_t^{k,r}$ to be 0 (i.e. t remains safe), then every path P must have at least one edge removed ($\sum_{e\in P} x_e \geq 1$). Conversely, if even one path is entirely intact (i.e. the sum is 0), then $z_t^{k,r}$ is forced to be 1 (indicating that the rumour reaches t).

Budget Constraint: Each edge removal incurs a cost c_e, and we have a total budget B available. This is represented by:

$$\sum_{e\in E} c_e\, x_e \leq B.$$

This constraint limits the overall number (or cost) of edges we may remove, forcing a trade-off between cost and containment effectiveness.

Integrality Constraints: The variables are binary:

$$x_e \in \{0,1\} \quad \forall e \in E, \qquad z_t^{k,r} \in \{0,1\} \quad \forall\, k \in N,\ r \in R,\ t \in T.$$

4.2.4 Complete Model

Bringing all the elements together, the final formulation is:

$$\begin{aligned}
\min \quad & \frac{1}{|N|} \sum_{k\in N} \sum_{r\in R} \sum_{t\in T} d_r\, z_t^{k,r} \\
\text{s.t.} \quad & \sum_{e\in P} x_e \geq 1 - z_t^{k,r}, \quad \forall k \in N,\ \forall r \in R,\ \forall t \in T,\ \forall P \in P^k(r,t), \\
& \sum_{e\in E} c_e\, x_e \leq B, \\
& x_e \in \{0,1\}, \quad \forall e \in E, \\
& z_t^{k,r} \in \{0,1\}, \quad \forall k \in N,\ \forall r \in R,\ \forall t \in T.
\end{aligned}$$

4.3 Interpretation of the Model

This model is a two-stage stochastic programme. The first stage involves making a single, irreversible decision on which network edges to remove (the x_e variables) under a limited budget. The second stage models the outcome in each scenario through the $z_t^{k,r}$ variables. In each scenario, for every path from a rumour source to a target, if the path is not completely cut, then the corresponding z variable is forced to 1 (indicating damage). The objective then minimises the average damage (weighted equally, if scenarios are assumed equally likely). In effect, we seek an interdiction strategy that robustly protects target nodes across a variety of possible rumour propagation scenarios.

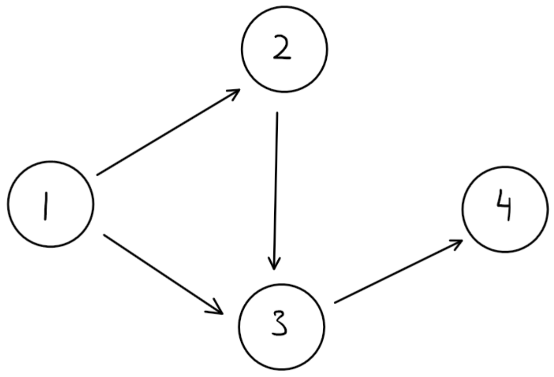

Figure 1: Simple Network

4.4 Example:

Imagine a tiny network with nodes $\{1,2,3,4\}$ and edges $E = \{(1,3),(1,2),(2,3),(3,4)\}$. Assume:

- Node 1 is the source of a rumour; node 4 is a critical target.
- Damage $d_1 = 10$ is incurred if the rumour reaches node 4.
- Each edge removal costs 1 unit, and the total budget is $B = 1$.
- We consider two scenarios:
    1. **Scenario 1:** The active propagation path is $1 \to 3 \to 4$ (the path $1 \to 2$ is inactive).
    2. **Scenario 2:** The active path is $1 \to 2 \to 3 \to 4$ (the direct edge $1 \to 3$ fails).

If we remove the edge $(3,4)$, then in both scenarios every possible path from node 1 to node 4 is interrupted. Consequently, we can set $z_4^{1,1} = 0$ and $z_4^{2,1} = 0$, leading to an expected damage of 0. On the other hand, removing edge $(1,3)$ would only block Scenario 1 (forcing $z_4^{1,1} = 0$) while leaving Scenario 2 unaffected ($z_4^{2,1} = 1$), yielding an expected damage of 5 (since $0.5 \times 0 + 0.5 \times 10 = 5$). Hence, the optimal decision is to remove edge $(3,4)$.

5 Gen - AI usage

5.1 ChatGPT

ChatGPT enhanced the quality of comments in the first code (Q1). Only the comments were inputted into ChatGPT, which subsequently provided 'improved' remarks. For instance, the comment "model result" was reformulated to "print the full model results".

Furthermore, ChatGPT's commenting methodology from the first question was applied to subsequent code analyses.

The data retention feature had been disabled, and the chats were subsequently deleted to ensure that no data could be utilised to train their model.

5.2 Grammarly

Grammarly was used to check for spelling errors and reduce grammatical inconsistencies. Additionally, word prompt suggestions were utilised to obtain synonyms and ensure that the report is not overly monotonous.

6 Reference

Yongjia Song and Thang N Dinh. Optimal containment of misinformation in social media: A scenario-based approach. In International Conference on Combinatorial Optimization and Applications, pages 547–556. Springer, 2014.