Integer Programming

Abrémod Training

July 1, 2020

Integer Programming (IP)

- Linear Programming Axioms
  - Additivity
  - Proportionality
  - Divisibility
  - Certainty
- Fractional solutions are not always okay.
- Binary decision variables greatly enrich our modeling capability.

Integer Programming (IP)

$$
\begin{aligned}
z^* = \min / \max \quad & c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \\
\text{subject to:} \quad & a_{i1} x_1 + a_{i2} x_2 + \cdots + a_{in} x_n \left\{ \begin{matrix} \leq \\ \geq \\ = \end{matrix} \right\} b_i, \quad i = 1, \ldots, m \\
& 0 \leq x_j \leq u_j, \quad j = 1, \ldots, n \\
& x_j \;\; \text{integer for some or all} \;\; j = 1, \ldots, n
\end{aligned}
$$

Transportation Problem, Revisited

- Sets and Indices
  - $i \in I$: warehouses
  - $j \in J$: demand centers (customers)
- Data
  - u_i: capacity for warehouse i (widgets)
  - d_j: demand at demand center j (widgets)
  - c_{ij}: shipping cost from warehouse i to demand center j ($/widget)
- Decision Variables
  - x_{ij}: number of widgets to ship from warehouse i to demand center j

Transportation Problem, Revisited

$$
\begin{aligned}
\min_{x} \quad & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \quad \text{(minimize shipping costs)} \\
\text{s.t.} \quad & \sum_{i \in I} x_{ij} = d_j, \quad j \in J \quad \text{(satisfy demand)} \\
& \sum_{j \in J} x_{ij} \le u_i, \quad i \in I \quad \text{(don't exceed capacity)} \\
& x_{ij} \ge 0, \quad i \in I,\ j \in J \quad \text{(ship nonnegative quantities)}
\end{aligned}
$$

Transportation Problem Extensions

- No more than half of customer 3's deliveries come from warehouses 1, 2, and 27 (LP)
- No more than half the warehouses can be built (IP)
- Each customer is served by a single warehouse (IP)
- Allow for unsatisfied demand, at a penalty
    - per unit shortage penalty (LP)
    - increasing unit penalties above threshold values (LP)

Transportation Problem Extensions

- Increasing marginal shipping costs
  - shipping cost (i, j) is $c_{ij}x_{ij}^2$ (easy nonlinear program)
  - marginal shipping cost is c_{ij} for $0 \leq x_{ij} \leq l_{ij}$ and $1.5c_{ij}$ for $x_{ij} > l_{ij}$ (LP)
- Decreasing marginal shipping costs (bulk discounts or economies of scale)
  - shipping cost (i, j) is $c_{ij}\sqrt{x_{ij}}$ (difficult nonlinear program)
  - marginal shipping cost is c_{ij} for $0 \leq x_{ij} \leq l_{ij}$ and $0.75c_{ij}$ for $x_{ij} > l_{ij}$ (IP)
- Fixed-charge to open a warehouse (IP)

Transportation Problem with Fixed Costs

Let f_i be the cost of opening a warehouse and y_i indicate whether we open warehouse i.

$$
\begin{aligned}
\min_{x} \quad & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in I} f_i y_i \\
\text{s.t.} \quad & \sum_{i \in I} x_{ij} = d_j, \quad j \in J \\
& \sum_{j \in J} x_{ij} \leq u_i y_i, \quad i \in I \\
& x_{ij} \geq 0, \quad i \in I, \ j \in J \\
& y_i \in \{0, 1\}, \quad i \in I
\end{aligned}
$$

Transportation Problem: Other Extensions

- No more than 10 warehouses built

$$\sum_{i \in I} y_i \leq 10$$

- Build at most one warehouse at locations $i = 1, 13, 101$

$$y_1 + y_{13} + y_{101} \leq 1$$

- A customer can receive widgets from only one warehouse. Let $w_{ij} = 1$ if customer j receives widgets from warehouse i, 0 otherwise

$$\sum_{i \in I} w_{ij} = 1, \quad j \in J$$

$$x_{ij} \leq d_j w_{ij}, \quad i \in I,\ j \in J$$

$$w_{ij} \in \{0, 1\}, \quad i \in I,\ j \in J$$

Transportation Problem: Other Extensions

- Marginal shipping cost is c_{ij} for $0 \leq x_{ij} \leq l_{ij}$ and $0.75c_{ij}$ for $x_{ij} > l_{ij}$
  z_{ij} indicates whether we ship at least l_{ij} widgets from i to j
  x^1_{ij} and x^2_{ij} are shipping volumes at level 1 and at level 2

$$
\begin{aligned}
\min_{x,z} \quad & \sum_{i \in I} \sum_{j \in J} (c_{ij} x^1_{ij} + 0.75 c_{ij} x^2_{ij}) \\
\text{s.t.} \quad & l_{ij} z_{ij} \leq x^1_{ij} \leq l_{ij}, \;\; i \in I, \; j \in J \\
& 0 \leq x^2_{ij} \leq (d_j - l_{ij}) z_{ij}, \;\; i \in I, \; j \in J \\
& \sum_{j \in J} (x^1_{ij} + x^2_{ij}) \leq u_i, \;\; i \in I \\
& \sum_{i \in I} (x^1_{ij} + x^2_{ij}) = d_j, \;\; j \in J \\
& z_{ij} \in \{0, 1\}, \;\; i \in I, \; j \in J \\
& \ldots
\end{aligned}
$$

Exercise

Extend the transportation problem to include a fixed costs for opening warehouses. (Set vtype = GRB.BINARY for the new decision variables).

How are IPs Solved?

- (Assuming minimization problem with binary variables)
- Relax integrality constraints and solve so-called *LP relaxation* to obtain $\hat{x}$.
- If $\hat{x}$ satisfies integrality constraints, return $\hat{x}$.
- Suppose not, and let i' be the index of a variable that should have been fractional but wasn't.
- Solve two subproblems, one with $x_{i'} = 0$ and one with $x_{i'} = 1$.
- Repeat.

How are IPs Solved?

- Along the way, you will eventually stumble upon feasible solutions. The cost of those solutions gives an upper bound on the optimal cost.
- Throw out a subproblem if
  - It is infeasible
  - Its optimal cost is not cheaper than the cost of a feasible solution you've already found.
- Minimum cost over all active subproblems gives a lower bound.
- Terminate when upper and lower bounds are within some tolerance.

How are IPs Solved?

- Gurobi will
  - Run heuristics to try to find good feasible solutions earlier (improves upper bound)
  - Add cutting planes to tighten LP relaxation (improves lower bound)
- You can help by
  - Specifying an initial feasible solution using GRBVar attribute Start.
  - Implement a callback to build heuristic solutions from relaxation solutions.
  - Formulate your problem intelligently.
- Always set reasonable stopping criteria for IPs.

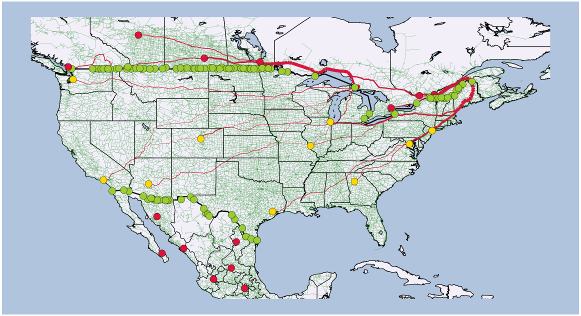
Adding Constraints Can Improve Performance

$$
\begin{aligned}
\min_{x} \quad & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in I} f_i y_i \\
\text{s.t.} \quad & \sum_{i \in I} x_{ij} = d_j, \;\; j \in J \\
& \sum_{j \in J} x_{ij} \leq u_i y_i, \;\; i \in I \\
& \color{red}{x_{ij} \leq d_j y_i, \;\; i \in I, \; j \in J} \\
& x_{ij} \geq 0, \;\; i \in I, \; j \in J \\
& y_i \in \{0, 1\}, \;\; i \in I
\end{aligned}
$$

Additional constraint is redundant, but can tighten the LP relaxation.

Interdicting Nuclear Material Smuggling

- Goal: Minimize probability of successful smuggling of nuclear material
- Approach: Install radiation sensors at key locations
- Question: How to select locations to achieve goal given limited resources?

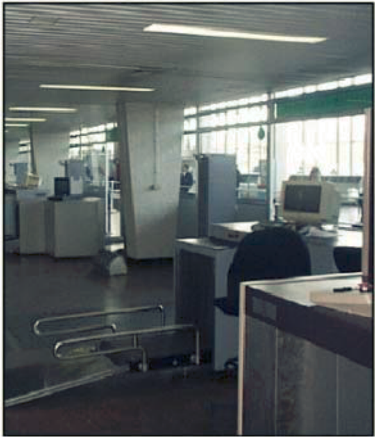


U.S. Land Border Crossings

Interdicting Nuclear Material Smuggling

Moscow's Sheremetyevo International Airport: September 1998

Radiation Sensors in Sheremetyevo Airport



SNIP: Stochastic Network Interdiction Problem

Structure:

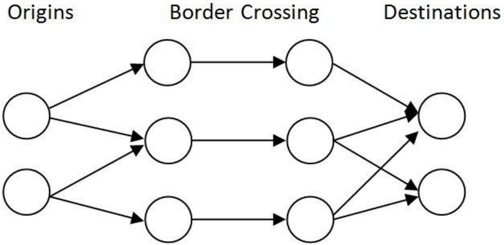
- Interdictor's decision: (First stage) Select locations to install sensors, subject to budget constraint
- Random event: Smuggler's origin-destination pair is realized
- Smuggler's decision: (Second stage) Select path from origin to destination to minimize probability of detection

Assumptions:

- Smuggler knows sensor locations, detection probabilities, chooses best path
- Interdictor and smuggler "see" the same network

One-Country Case

- Potential smugglers indexed by $\omega \in \Omega$, with associated probability p^ω
- Checkpoints indexed by $k \in K$
- Evasion probability p_k^ω if no sensor installed at checkpoint $k \in K$, 0 otherwise
- Cost of installing sensor at checkpoint c_k, $k \in K$
- Installation budget b



Formulation

Decision Variables:

- $x_k = 1$ if sensor installed at checkpoint k, 0 otherwise
- θ^ω : probability that smuggler ω evades detection, computed as $\theta^\omega = \max_{k \in K} p_k^\omega (1 - x_k)$

Model:

$$
\begin{aligned}
\min_{x,\theta} \quad & \sum_{\omega \in \Omega} p^\omega \theta^\omega \\
\text{s.t.} \quad & \theta^\omega \geq p_k^\omega - p_k^\omega x_k, \quad k \in K, \ \omega \in \Omega \\
& \sum_{k \in K} c_k x_k \leq b \\
& x_k \in \{0, 1\}, \quad k \in K
\end{aligned}
$$

Tightening the Formulation

Consider $\theta^\omega \geq p_k^\omega - p_k^\omega x_k$, for a smuggler with $p_1^\omega = 1$, $p_2^\omega = 0.8$, $p_3^\omega = 0.6$, and $p_4^\omega = 0.4$.

$$
\begin{aligned}
\theta^\omega &\geq 1 - x_1 \\
\theta^\omega &\geq 0.8 - 0.8x_2 \\
\theta^\omega &\geq 0.6 - 0.6x_3 \\
\theta^\omega &\geq 0.4 - 0.4x_4
\end{aligned}
$$

- Role of x_k in above is to make inequality non-binding if $x_k = 1$
- Making coefficients too large hurts solve time
- Suppose budget constraint is $\sum_{k \in K} x_k \leq 2$, can we make coefficients smaller?

Big-M Coefficient Tuning

- Rewrite $\theta^\omega \geq p_k^\omega - (p_k^\omega - \underline{\theta}^\omega)x_k$ where $\underline{\theta}^\omega$ is a lower bound on θ^ω.
- Find $\underline{\theta}^\omega$ by allocating sensors to smuggler ω's best checkpoints
- Wait-and-see bound

Valid Inequalities

Consider a smuggler with $p_1^\omega = 1$, $p_2^\omega = 0.8$, $p_3^\omega = 0.6$, and $p_4^\omega = 0.4$. How much does smuggler evasion probability decrease as we interdict checkpoints?

$$\theta^\omega \geq 1 - 0.2x_1 - 0.2x_2 - 0.2x_3 - 0.4x_4(1)$$

What if we ignore checkpoint 2?

$$\theta^\omega \geq 1 - 0.4x_1 - 0.2x_3 - 0.4x_4(2)$$

Both (1) and (2) are valid constraints to add.
Under what conditions is (1) stronger?
Under what conditions is (2) stronger?

Valid Inequalities

Consider smuggler ω, and let $\{k_1, k_2, \ldots, k_n\}$ satisfy:

$$r^{\omega}_{k_1} \geq r^{\omega}_{k_2} \geq \cdots \geq r^{\omega}_{k_n}$$

Then

$$\theta^{\omega} \geq r^{\omega}_{k_1} - (r^{\omega}_{k_1} - r^{\omega}_{k_2})x_{k_1} - \cdots - (r^{\omega}_{k_n} - 0)x_{k_l}$$

- The above "step inequality" can be written for any subset of checkpoints.
- If $x_{k_{i+1}} > x_{k_i}$, then we should leave checkpoint k_{i+1} out.
- Exponentially many subsets, can't enumerate all possible step inequalities.

Reformulation

- Let $v_k^\omega = 1$ if smuggler ω traverses checkpoint k
- Let $K_k^\omega = \{k' \in K : p_{k'}^\omega < p_k^\omega\}$ (i.e. checkpoints worse than k from ω's perspective)
- The following reformulation avoids big-M coefficients:

$$
\begin{aligned}
\min_{x,v,\theta} \quad & \sum_{\omega \in \Omega} p^\omega \theta^\omega \\
\text{s.t.} \quad & \theta^\omega = \sum_{k \in K} p_k^\omega v_k^\omega, \;\; \omega \in \Omega \\
& x_k \geq \sum_{k' \in K_k^\omega} v_{k'}^\omega, \;\; k \in K, \; \omega \in \Omega \\
& \sum_{k \in K} v_k^\omega = 1, \;\; \omega \in \Omega \\
& x_k \in \{0,1\}, \;\; k \in K \\
& v_k^\omega \in \{0,1\}, \;\; k \in K, \; \omega \in \Omega
\end{aligned}
$$

Gurobi Parameters

- Method : dual simplex, primal simplex, barrier
- Presolve, PrePasses
- Termination : IterationLimit, BarIterLimit, TimeLimit, NodeLimit, SolutionLimit, ...
- Tolerances : FeasibilityTol, IntFeasTol, MIPGap, ...
- MIP : Heuristics, MIPFocus, ImproveStartGap/Nodes/Time, ...
- MIPCuts : Cuts, CutPasses, MIRCuts, ...
- Some parameters are tied to a particular algorithm (i.e. barrier, simplex, MIP)
- GRBEnv.set(parameter, value), Model.SetParam(parameter, value) in Python

Metaparameters

- Setting value of Cuts parameter will change level of aggressiveness for all cut types.
  - Can be overridden for a particular type of cut by setting CliqueCuts, CoverCuts, etc.
- MIPFocus=1 (focus on feasiblity) sets CutPasses=5, Heuristics=0.2, and VarBranch=1
- MIPFocus=2 (focus on optimality) sets Cuts=2, Presolve=2,

Automatic Parameter Tuning

- Through API via GRBModel.Tune()
- Through command line via grbtune (grbtune [param=value] filename1 filname2 ...)
- Goal: Vary parameters that matter (where is solve time being spent?)
- Minimize runtime or minimize optimality gap
- Parameters that control the parameter tuning tool:
  - TuneTimeLimit
  - TuneTrials
  - TuneResults
  - TuneOutput
  - ResultFile
- Important parameters for difficult MIP models : MIPFocus, Presolve, Cuts, CutPasses, VarBranch, Method (if root relaxation difficult), Heuristics
- Mean improvement from best settings over 423 models : 2.91X

Callbacks

- Create a subclass of GRBCallback and implement a callback() method
- Call GRBModel.SetCallback(GRBCallback)
- callback() method is called periodically during optimization
- Query the protected member *where* to figure out where you are (PRESOLVE, MIPNODE, MIPSOL, etc.)
- GRBCallback methods
  - AddCut (if where is MIPNODE)
  - AddLazy (if where is MIPNODE or MIPSOL)
  - GetNodeRel (if where is MIPNODE and GRB.Callback.MIPNODE_STATUS is GRB.OPTIMAL)
  - GetSolution (if where is MIPSOL)

Custom Rounding Heuristics

- If where == GRB.Callback.MIPNODE and GRB.Callback.MIPNODE_STATUS == GRB.OPTIMAL
- Call GRBCallback.GetNodeRel(vars) to get LP relaxation solution
- Run heuristic
- Call GRBCallback.SetSolution(vars, soln) to pass back the heuristic solution