CS498: Algorithmic Engineering

Lecture 3: Sensitivity Analysis & Network Models

Elfarouk Harb

University of Illinois Urbana-Champaign

01/27/2026

Outline

Theme of Today

Last time:

- Simplex walks along vertices (BFS).
- Duality: each constraint $\leftrightarrow$ a dual variable.

Today: "What is a constraint worth?"

- Dual variables = **shadow prices** for constraints.
- Reduced costs = **value of turning on** a variable.
- Network flow models as a structured LP family where these ideas are very tangible.

Goal: Use LP duals to answer: *"If I loosen this constraint a bit, how much better can I do?"*

Warm-up: Single-Constraint Example

Consider the toy LP:

$$\max 3x \quad \text{s.t. } x \leq 10,\ x \geq 0.$$

Primal solution:

$$x^* = 10, \quad z_P^* = 3 \cdot 10 = 30.$$

Increase RHS from 10 to 11:

$$x' = 11, \quad z' = 3 \cdot 11 = 33 \ \Rightarrow\ \Delta z_P^* = 3.$$

Dual:

$$\min 10y \quad \text{s.t. } y \geq 3,\ y \geq 0 \ \Rightarrow\ y^* = 3,\ z_D^* = 30.$$

Dual optimal $y^* = 3$ is exactly **"$3 per extra unit of the constraint RHS."**

Slightly Less Toy: 2 Resources, 1 Product

Suppose we produce a single product x:

- Each unit of x uses:
  - 2 hours of labor,
  - 1 kg of raw.
- Each unit yields \$5 profit.

Primal LP:

$$\begin{aligned} \max \quad & 5x \\ \text{s.t.} \quad & 2x \le 40 \quad \text{(labor: 40 hours)} \\ & 1x \le 30 \quad \text{(raw: 30 kg)} \\ & x \ge 0 \end{aligned}$$

Constraints imply $x \le 20$ and $x \le 30 \Rightarrow x^* = 20$.

$$z_P^* = 5 \cdot 20 = 100.$$

Labor is binding, raw is slack.

Dual of the 2-Resource Example

Primal:

$$\max 5x \quad \text{s.t.} \begin{cases} 2x \leq 40 & \text{(labor)} \\ 1x \leq 30 & \text{(raw)} \\ x \geq 0 \end{cases}$$

Introduce dual variables:

$$y_1 \geq 0 \text{ for labor}, \quad y_2 \geq 0 \text{ for raw.}$$

Dual:

$$\begin{aligned} \min \quad & 40y_1 + 30y_2 \\ \text{s.t.} \quad & 2y_1 + 1y_2 \ \geq \ 5 \quad \text{(one constraint per primal variable)} \\ & y_1, y_2 \geq 0 \end{aligned}$$

Optimum at $y_1^* = 2.5$, $y_2^* = 0$.

Shadow Prices in the 2-Resource Example

From the dual solution:

$$y_1^* = 2.5 \quad \text{(labor)}, \qquad y_2^* = 0 \quad \text{(raw)}.$$

Check against the primal: increase labor from 40 to 41.

- New labor constraint: $2x \leq 41 \Rightarrow x \leq 20.5$.
- Raw constraint: $x \leq 30$ still non-binding.
- New optimum: $x' = 20.5$, $z' = 5 \cdot 20.5 = 102.5$.

$$\Delta z_P^* = 102.5 - 100 = 2.5 \approx y_1^*.$$

Raw is slack, so $y_2^* = 0$: extra raw does not improve the optimum.
Only the binding labor constraint has nonzero shadow price.

Economic Interpretation: Shadow Prices

What does a dual variable y_i represent?

$$\text{Dual objective:} \quad \min \; b_1 y_1 + b_2 y_2 + \cdots + b_m y_m.$$

If we increase resource i from b_i to $b_i + 1$ (one more unit), the optimal dual objective changes by roughly y_i:

$$\Delta z_D^* \approx y_i.$$

By **strong duality**, at optimum

$$z_P^*(b) = z_D^*(b),$$

so the same change shows up in the **primal** optimum.

- y_i is the **marginal value** (shadow price) of resource i.
- It answers: **"How much does the optimal objective change if I get one more unit of this resource?"**

Complementary Slackness

Concept: You shouldn't pay for something you don't use.

- If a primal constraint is **slack** (not binding, leftover resource), then its dual variable must be **zero**.
- *Logic:* If we already have unused resource i, an extra unit adds \$0 to the optimum $\Rightarrow y_i = 0$.
- Conversely, if $y_i > 0$ (resource is valuable), then the constraint must be **tight**: we are using all of resource i.

Shadow prices light up exactly the bottleneck constraints.

Shadow Prices in Gurobi: A Small Factory Model

Toy factory: 2 products, 2 resources, plus shadow prices.

```python
import gurobipy as gp

m = gp.Model("toy_factory")

# Products and data
products = ["standard", "deluxe"]
profit   = {"standard": 5.0, "deluxe": 9.0}

resources = ["labor", "raw"]
capacity  = {"labor": 40.0, "raw": 30.0}
use = {("standard", "labor"): 2.0, ("standard", "raw"): 1.0, ("deluxe", "labor"): 3.5, ("deluxe",   "raw"): 3.0}

# Decision vars
x = {p: m.addVar(lb=0, name=f"x_{p}") for p in products}


# Resource constraints
cons = {}
for r in resources:
    cons[r] = m.addConstr(gp.quicksum(use[(p, r)] * x[p] for p in products) <= capacity[r], name=f"cap_{r}")


# Objective: max profit
m.setObjective(gp.quicksum(profit[p] * x[p] for p in products), gp.GRB.MAXIMIZE)
m.optimize()


print("Primal solution:")
for p in products: print(p, x[p].X)

print("\nShadow prices:")
for r in resources: print(r, cons[r].Pi)
```

Interpreting the Toy Factory Gurobi Output

Run the previous code:

- Primal: some optimal $(x^*_{\text{standard}}, x^*_{\text{deluxe}}) = (6, 8)$.
- Duals:

$$\Pi_{\text{labor}} = 2.4, \quad \Pi_{\text{raw}} = 0.$$

Interpretation:

- Labor is fully used $\Rightarrow$ it is a bottleneck $\Rightarrow$ positive price.
- Raw has slack $\Rightarrow$ $y_{\text{raw}} = 0$ (extra raw is locally worthless).

Engineering Application: Bottleneck Analysis

Scenario: You manage a cloud cluster and want to maximize profit of your running jobs:

- Variables x_j: which jobs / workloads to run.
- Constraints: CPU hours, RAM, GPU hours, network bandwidth.

You run the LP and look at duals:

- CPU dual (y_{cpu}) = 0.05
- RAM dual (y_{ram}) = 0.00
- GPU dual (y_{gpu}) = 50.0

Decision:

- Don't buy RAM. You already have slack.
- **Buy GPUs**: each extra GPU-hour is worth roughly $50 of objective.

Shadow prices = **shopping list** for infrastructure.

Shadow Prices for Variable Bounds in Gurobi

When we write an explicit constraint in Gurobi (e.g., `const = m.addConstr(...)`), its shadow price appears as `const.Pi` (dual value).

But every variable also has **implicit bound constraints:**

$$x_j \geq \mathrm{LB}_j, \quad x_j \leq \mathrm{UB}_j.$$

Gurobi stores their shadow prices as **reduced costs**:

- `x.RC` = dual value of the variable's bound constraint.
- If x_j is at its **upper bound**, `x.RC` is the shadow price of $x_j \leq \mathrm{UB}_j$.
- If x_j is at its **lower bound**, `x.RC` is the shadow price of $x_j \geq \mathrm{LB}_j$.

Flows: Think Water in Pipes

To introduce network flows, imagine:

- Nodes = junctions in a water network.
- Pipes = directed connections from one junction to another.
- At each pipe we can send some amount of water per minute.

Questions we might care about:

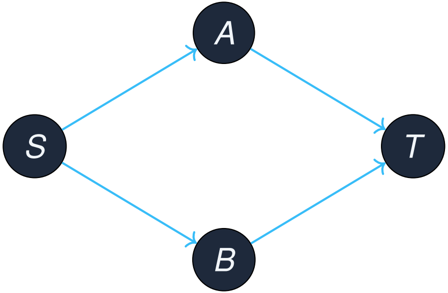
- How much water can we send from a reservoir to a city?
- What is the cheapest way to route water, if different pipes have different energy costs?
- Which pipe is the bottleneck or is important?

This intuition transfers directly to **cars on roads**, **packets on links**, **electricity on lines**, ...

From Pipes to Graphs

We represent the network as a directed graph $G = (V, E)$.

- V = set of nodes (junctions).
- E = set of directed arcs (pipes, roads, links).



Goal (informally): pick flows on arcs to move "stuff" from S to T respecting the network.

Flows and Capacities

For each arc $(u, v) \in E$ we define:

- Flow variable $f_{uv} \geq 0$: amount of flow on arc (u, v).
- Capacity $u_{uv} \geq 0$: maximum allowed flow on that arc.

Capacity constraint:

$$0 \leq f_{uv} \leq u_{uv} \quad \forall (u, v) \in E.$$

"The pipe can carry at most u_{uv} units per unit time."

Flow Conservation at Intermediate Nodes

Except for sources/sinks, a node just passes flow through.
Flow conservation at node n:

$$\sum_{(n,v)\in E} f_{nv} - \sum_{(u,n)\in E} f_{un} = 0.$$

- Sum of flows **leaving** n minus sum of flows **entering** n is zero.
- What comes in must go out.

Adding Supply and Demand

Some nodes create or consume flow.
Supply/demand vector b:

$$b_n = \begin{cases} +(\text{supply}) & \text{if } n \text{ is a source} \\ -(\text{demand}) & \text{if } n \text{ is a sink} \\ 0 & \text{otherwise} \end{cases}$$

General flow conservation:

$$\sum_{(n,v)\in E} f_{nv} - \sum_{(u,n)\in E} f_{un} = b_n \quad \forall n \in V.$$

Positive b_n: net outflow (supply). Negative b_n: net inflow (demand).

Costs on Flow

Often **each arc has a cost** c_{uv} per unit of flow.

- Could be money: tolls, fuel, energy.
- Could be time: latency, travel time.

Total cost:

$$\text{Cost} = \sum_{(u,v)\in E} c_{uv} f_{uv}.$$

We want to send the required flow **with minimum total cost.**

Minimum-Cost Flow as an LP

Put it all together:

Decision variables: $f_{uv} \geq 0$ for each arc $(u, v) \in E$.

Objective:

$$\min \sum_{(u,v)\in E} c_{uv} f_{uv}.$$

Constraints:

- Capacity:

$$0 \leq f_{uv} \leq u_{uv} \quad \forall (u, v) \in E.$$

- Flow conservation at each node n:

$$\sum_{(n,v)\in E} f_{nv} - \sum_{(u,n)\in E} f_{un} = b_n.$$

This is a standard LP with a lot of structure.

The Gridlock Gambit Scenario

Story: You've joined **TransiLogic AI**, optimizing urban traffic flow. **Mission:** Find the cheapest way to route 10 vehicles from hub S to destination T through a congested road network.

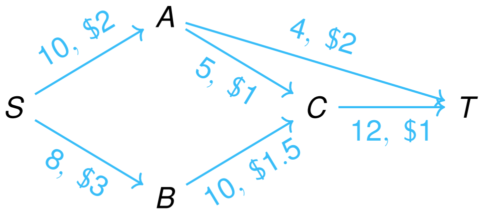
- Each directed road segment (u, v) has:
  - a **capacity**: max cars per minute,
  - a **cost**: energy / delays / tolls per car.
- We must:
  - send 10 units from S to T,
  - respect all capacities and conservation,
  - minimize total cost.

Gridlock LP Formulation

Decision variables: $f_{uv} \geq 0$ for each arc (u, v).

Objective:

$$\min \; 2f_{SA} + 3f_{SB} + 1f_{AC} + 1.5f_{BC} + 2f_{AT} + 1f_{CT}.$$



Capacity constraints:

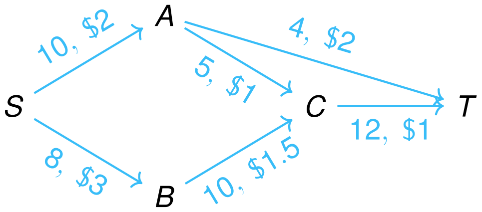
$$0 \leq f_{SA} \leq 10, 0 \leq f_{SB} \leq 8, 0 \leq f_{AC} \leq 5, \ldots$$

Flow conservation:

$$\begin{aligned} \text{At } S: &\quad f_{SA} + f_{SB} - 0 = 10, \\ \text{At } A: &\quad f_{AC} + f_{AT} - f_{SA} = 0, \\ \text{At } B: &\quad f_{BC} - f_{SB} = 0, \ldots \end{aligned}$$

Solving Gridlock in Gurobi

```python
import gurobipy as gp
import numpy as np

arcs = [('S','A',10,2.0), ('S','B',8,3.0), ('A','C',5,1.0),
    ('B','C',10,1.5), ('A','T',4,2.0), ('C','T',12,1.0)]

nodes = ['S', 'A', 'B', 'C', 'T']
supply = {'S': 10, 'A': 0, 'B': 0, 'C': 0, 'T': -10}


m = gp.Model("gridlock_gambit")

flow = {
    (u, v) : m.addVar(lb=0, ub=cap, name=f"f_{u}_{v}") for u, v, cap, cost in arcs
}
for n in nodes:
    outflow = gp.quicksum(flow[(u, v)]
                          for u, v, _, _ in arcs if u == n)
    inflow = gp.quicksum(flow[(u, v)]
                         for u, v, _, _ in arcs if v == n)
    m.addConstr(outflow - inflow == supply[n],
                name=f"flow_{n}")


m.setObjective(gp.quicksum(flow[(u, v)] * cost for u, v, _, cost in arcs), gp.GRB.MINIMIZE)

m.optimize()


print(f"Optimal Cost: {m.ObjVal:.2f}")
for (u, v), var in flow.items():
    if not np.isclose(var.X, 0):
        print(f"  {u} -> {v}: {var.X:.1f}")
```



Baseline Gridlock Results

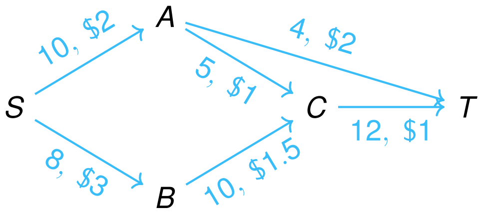
Optimal Flows:

$$f_{S\to A} = 9, \quad f_{S\to B} = 1, \quad f_{A\to C} = 5$$
$$f_{A\to T} = 4, \quad f_{B\to C} = 1, \quad f_{C\to T} = 6.$$

Total Cost: \$41.50

Cost Breakdown:

- $S \to A$: $9 \times 2.0 = 18.0$
- $S \to B$: $1 \times 3.0 = 3.0$
- $A \to C$: $5 \times 1.0 = 5.0$ (at capacity!)
- $A \to T$: $4 \times 2.0 = 8.0$ (at capacity!)
- $B \to C$: $1 \times 1.5 = 1.5$
- $C \to T$: $6 \times 1.0 = 6.0$



Shadow Prices on Capacities from Gurobi (Complementary Slackness)

Access capacity-related dual info in Gurobi:

```
print("\nShadow prices on capacities (from reduced costs):")
for u, v, cap, cost in arcs:
    var = flow[(u, v)]
    # For a binding upper bound, var.RC encodes a shadow price
    if np.isclose(var.X, cap):  # at (or very near) capacity
        print(f"  {u}->{v}: pi  {var.RC:.2f}")
```

Results:

- $\pi_{A\to C} = -1.5$ (binding!)
- $\pi_{A\to T} = -1.5$ (binding!)
- All others $= 0$

Interpretation: Adding 1 unit of capacity on $A \to C$ or $A \to T$ would *reduce* total cost by \$1.50.

Why Shadow Price ≈ -1.50?

Intuition: Look at competing routes.

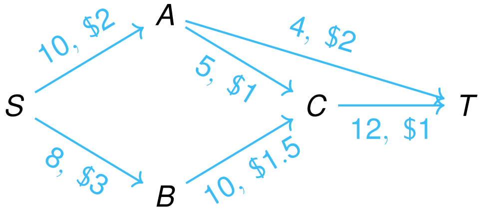
Cheap routes via A:

- $S \to A \to C \to T$: $2.0 + 1.0 + 1.0 = 4.0$/unit.
- $S \to A \to T$: $2.0 + 2.0 = 4.0$/unit.
- Both saturated.

Next-best detour:

- $S \to B \to C \to T$: $3.0 + 1.5 + 1.0 = 5.5$/unit.
- Penalty: $5.5 - 4.0 = 1.5$ per unit.

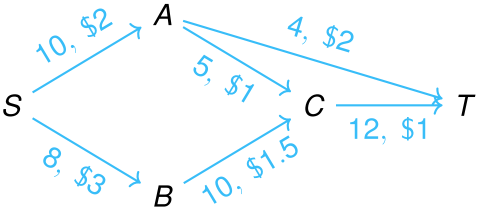
If we add 1 unit of capacity on $A \to C$, we can reroute 1 unit from the expensive detour to the cheap path, saving \$1.5.



Scenario Analysis: Closing $A \to C$

What if arc $A \to C$ is blocked (capacity $= 0$)?

```python
# Close A->C by setting capacity to 0 and re-optimizing
flow['A', 'C'].UB = 0
m.optimize()

print(f"New Cost after closing A->C: {m.ObjVal:.2f}")
```



Results:

- New optimal cost: \$49.00
- Increase vs baseline: $49.0 - 41.5 = 7.5$

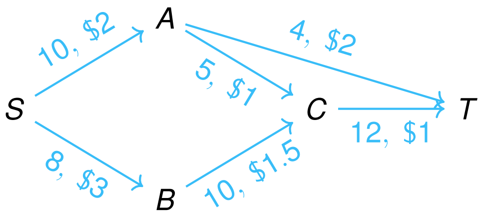
Dual Prediction:

- Shadow price: $\pi_{A \to C} \approx -1.5$
- Change in capacity: $\Delta = -5$.
- Predicted cost change: $\pi \cdot \Delta = (-1.5) \cdot (-5) = +7.5$ ✓

Economic Interpretation

The story of shadow prices in this network:

- **Bottlenecks:** Arcs $A \to C$ and $A \to T$ are scarce and valuable
  - $|\pi| = 1.5$ means high leverage.
  - Each extra unit of capacity is worth \$1.50.
- **Resilience:** When a cheap link fails:
  - The network reroutes flow to more expensive corridors.
  - Total cost rises by (shadow price) $\times$ (lost capacity).



Investment Recommendation: If you can widen only one or two roads, pick $A \to C$ and $A \to T$.

Important Caveat on Shadow Prices: Only Valid Locally For Local "Small" Changes.

Shadow prices are *local sensitivities*.

Their interpretation is only exact while the optimal basis stays the same after the small perturbation.
That's why we say: shadow prices are **locally valid**, not global.

When the Shadow Price Rule Fails

Example: Recall the Gridlock Gambit.
From the baseline solution:

$$\pi_{A \to C} = -1.5, \quad \pi_{A \to T} = -1.5$$

Each extra unit of capacity saves \$1.50 locally.
What if we add 5 new lanes on $A \to C$?

- Predicted improvement (linear model): $5 \times -1.5 = -7.5$.
- True improvement (re-solve LP): -1.5.

Why?

- Once $A \to C$ isn't tight anymore, it's no longer a bottleneck.
- The active set (tight constraints) changed.
- New shadow prices = new slopes.

Lesson: Shadow prices hold only **until the binding set changes.**

How Small Is “Small Enough”?

Shadow prices apply only while the same constraints stay binding: No change in the active (tight) constraint set

Geometric view:

- The LP optimum lies at a vertex of the feasible region.
- Small moves that keep you on the same vertex $\Rightarrow$ same slope (shadow price).
- Large moves switch to a new corner $\Rightarrow$ new slope (new shadow price).

In practice:

- Solvers like Gurobi can report **sensitivity ranges**:
  - `constraint.SARHSLow`, `constraint.SARHSUp`: how far you can change a RHS before basis changes.
  - `x.SALBLow`, `x.SALBUp`: same for shadow prices on variable bounds.
- Within that range, the dual values (y_i, `.RC`, `.Pi`) are valid.

Takeaway: Shadow prices are local. Once a new constraint becomes tight or slack, the slope changes.

Summary: Sensitivity & Networks

What we did today:

- Interpreted dual variables y_i as **shadow prices**.
- Saw simple scalar and 2-resource examples where $\Delta z_P^* \approx y_i \cdot \Delta b_i$.
- Used **strong duality** to link dual changes to primal changes.
- Related **complementary slackness** to “don’t pay for unused resources”.
- Built network flow models from first principles.
- Used the **Gridlock Gambit** problem to:
  - identify bottleneck arcs and do scenario analysis,
  - predict the cost of closures / capacity expansions.

Tagline: Duals are not just math; they quantify *which constraints really matter*.