CIS1921

Lecture 5: Mixed-Integer & Linear Programming

Today

- Moving away from SAT solving
  - But we will tie it back in later!
- Start looking at “high-level” solvers
- Specify constraints in something closer to mathematical language (as opposed to SAT clauses)

Reminders

- HW2 is due on Monday, October 7, 11:59PM
  - Start Early!
  - Make sure set-up works!
- Fall Break next week
  - No class?

Basic Linear Program

- You're deciding what to bring to a potluck and want a meal with ≥ 5000 calories but ≤ 200 mg sodium.
- You want to spend as little money as possible.

Item	Price/kg	Calories/kg	Sodium/kg
Rice	1.25	750	15
Pasta	1.65	1200	35
Couscous	1.35	1000	60

Linearity

- $f(x_1, \ldots, x_n)$ is a **linear function** if it is of the form

$$f(x_1, \ldots, x_n) = a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$$

- A **linear inequality** has the form:

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \geq b$$

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq b$$

Linear Programs

- A **linear program** is a special class of optimization problems with the goal: optimize a **linear function** subject to **linear (in)equalities**
  - Strict inequalities not allowed: $<$, $>$, $!=$
- Widely solved in industry for maximizing value, minimizing cost

Example LP

- LP formalization:

$$
\begin{aligned}
\text{minimize} \quad & 3x_1 + 2x_2 - 4x_3 + 5 \\
\text{subject to} \quad & x_1 \geq 2 \\
& x_2 + 2x_3 \leq 10 \\
& x_1, x_2, x_3 \geq 0
\end{aligned}
$$

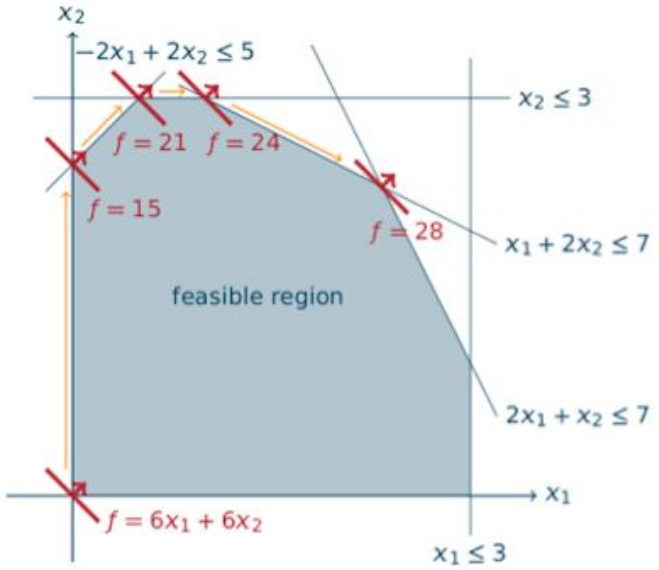
A More Complex Example LP

- LP formalization:

$$\begin{aligned} \text{maximize} \quad & 2x_1 + 5x_2 \\ \text{subject to} \quad & 0 \leq x_1, x_2 \leq 3 \\ & -2x_1 + 2x_2 \leq 5 \\ & x_1 + 2x_2 \leq 7 \\ & 2x_1 + x_2 \leq 7 \end{aligned}$$

Linear Programming Methods

- Major theory result: any LP is polytime solvable
- In practice: **Simplex algorithm**
  - George Dantzig, 1947
  - Worst-case exponential time
  - Practically fast for most problems
  - Check the corners!



Linear Programming Methods

- More recently: interior-point methods
- Karmarkar's algorithm (1984)
  - Polytime and practically fast

NYT (left)
TIME (right)

Breakthrough in Problem Solving

By JAMES GLEICK

A 28-year-old mathematician at A.T.&T. Bell Laboratories has made a startling theoretical breakthrough in the solving of systems of equations that often grow too vast and complex for the most powerful computers.

The discovery, which is to be formally published next month, is already circulating rapidly through the mathematical world. It has also set off a deluge of inquiries from brokerage houses, oil companies and airlines, industries with millions of dollars at stake in problems known as linear programming.

Faster Solutions Seen

These problems are fiendishly com-

"Science has its moments of great progress, and this may well be one of them."

Because problems in linear programming can have billions or more possible answers, even high-speed computers cannot check every one. So computers must use a special procedure, an algorithm, to examine as few answers as possible before finding the best one — typically the one that minimizes cost or maximizes efficiency.

A procedure devised in 1947, the simplex method, is now used for such problems,

Continued on Page A19, Column 1

Karmarkar at Bell Labs: an equation to find a new way through the maze

Folding the Perfect Corner

A young Bell scientist makes a major math breakthrough

Every day 1,200 American Airlines jets crisscross the U.S., Mexico, Canada and the Caribbean, stopping in 110 cities and bearing over 80,000 passengers. More than 4,000 pilots, copilots, flight personnel, maintenance

Indian-born mathematician at Bell Laboratories in Murray Hill, N.J., after only a years' work has cracked the puzzle of linear programming by devising a new algorithm, a step-by-step mathematical formula. He has

Basic Linear Program

- You're deciding what to bring to a potluck and want a meal with ≥ 5000 calories but ≤ 200 mg sodium.
- You want to spend as little money as possible.

Item	Price/kg	Calories/kg	Sodium/kg
Rice	1.25	750	15
Pasta	1.65	1200	35
Couscous	1.35	1000	60

Basic Linear Program

- LP formulation:

$$\begin{aligned} \text{minimize} \quad & price \\ \text{subject to} \quad & calories \geq 5000 \\ & sodium \leq 200 \end{aligned}$$

- Plus implicit constraint: can't buy negative amounts

Basic Linear Program

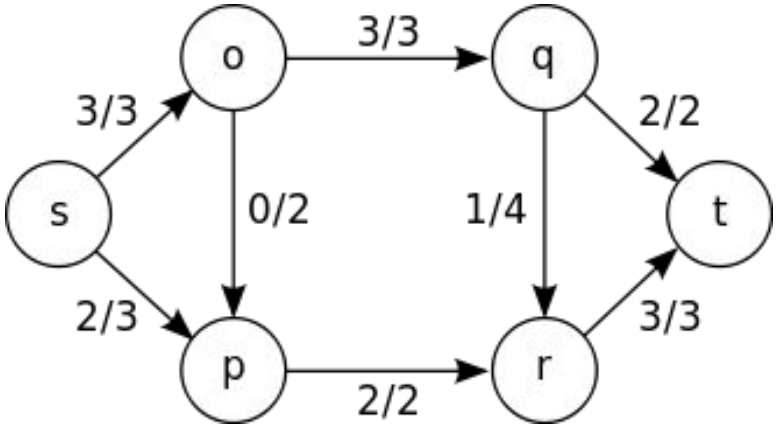
- LP formulation:

$$\begin{aligned} \text{minimize} \quad & 1.25r + 1.65p + 1.35c \\ \text{subject to} \quad & 750r + 1200p + 1000c \geq 5000 \\ & 15r + 35p + 60c \leq 200 \\ & r \geq 0,\ p \geq 0,\ c \geq 0 \end{aligned}$$

Demo

Max Flows with LP

- Max flow problem has a natural LP formulation
- Recall the problem: how much flow can we send along the edges from s to t?
  - Flow conservation
  - Capacity constraints



Max Flows with LP

- Variables:

  f_{uv} = total flow along edge (u, v)

- Objective:

  maximize $\sum_{v \in N_{\text{Out}}(s)} f_{sv}$

  - Alternatively: add ∞-capacity feedback edge (t, s) and maximize f_{ts}

Max Flows with LP

- Capacity constraints:

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

- Conservation constraints:

$$\sum_{v \in N_{\text{In}}(u)} f_{vu} - \sum_{v \in N_{\text{Out}}(u)} f_{uv} = 0 \quad \forall u \in V - \{s,t\}$$

  - If we added feedback edge, don't exclude s, t

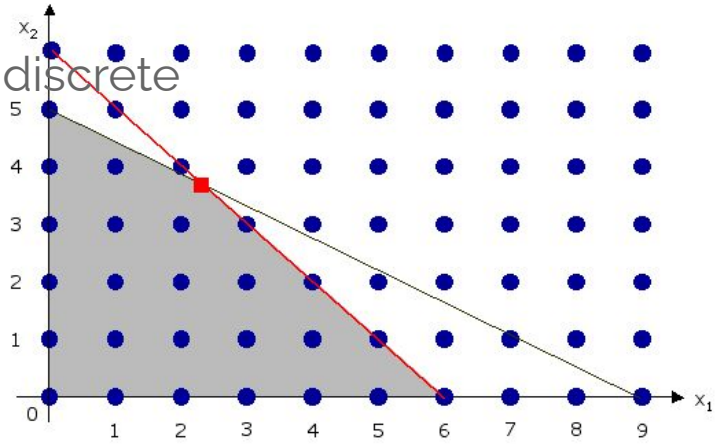
Demo

Assumptions in LP

- Linear programming assumes that the problem obeys (or is approximated by) the following laws
- **Proportionality:** The contribution of any decision variable to the objective function is proportional to its value
- **Divisibility:** fractional values are acceptable

Integer (Linear) Programming

- **Integer linear program (ILP):** a linear program with the additional constraint that variables must take integer values
  - aka **integer program (IP)**
  - In real life, items often come in discrete units



Integer (Linear) Programming

- **Ex:** What should we buy to maximize profit?
  - We have $45 to buy pies ($5) and cakes ($9).
  - We can resell pies for $5 profit and cakes for $8 profit.
  - But we can only carry 6 items total.
- IP formulation:

$$\begin{aligned} \text{maximize} \quad & 5p + 8c \\ \text{subject to} \quad & 5p + 9c \leq 45 \\ & p + c \leq 6 \\ & p, c \geq 0 \text{ are integers} \end{aligned}$$

Rounding LPs from ILPs?

- What if we just solve the **LP relaxation** and round?

Bad Theory News

- **Bad news 1:** we can construct ILPs whose rounded LP solution is arbitrarily far away
  - Sometimes, we can “round” in a clever way so that the rounded solution is not too far
- **Bad news 2:** integer programming is NP-complete!
- Good practical news: lots of work on robust solvers for real-world IPs

Mixed-Integer Programming

- **Mixed-integer program (MIP):** some variables may be constrained to be integers, and some may not
- Objectives & constraints are still linear!
- We'll just talk about MIP, since it generalizes IP

MIP in OR-Tools

- Nearly the same as LP! Only differences:
- **C**OIN-OR's **B**ranch-and-**Cut** solver
  - COIN-OR: Computational Optimization Infrastructure for Operations Research

```
from ortools.linear_solver.pywraplp import Solver
model = Solver('my_MIP_model', Solver.CBC_MIXED_INTEGER_PROGRAMMING)
```

- Declaring fractional or integer variables

```
x = model.NumVar(0, Solver.Infinity(), 'x')
n = model.IntVar(0, Solver.Infinity(), 'n')
b = model.BoolVar('b')
```

Capital-Budgeting Problem

- Common MIP modeling problem
- We have n possible investments, each with value v_i
- We have m resources, each with amount a_j
- Investment i costs c_{ij} units of resource j
- Want to maximize value

Capital-Budgeting Problem

- x_i is 0/1 variable indicating if we pick i^{th} investment
  - 0/1 variables are **very** common and useful in modeling

$$\begin{aligned} \text{max} \quad & \textstyle\sum_i v_i x_i \\ \text{s.t.} \quad & \textstyle\sum_i c_{ij} x_i \leq a_j \qquad \text{for each } j \end{aligned}$$

Capital-Budgeting Problem

- What if we need to invest in i in order to invest in j?

$$x_i \geq x_j$$

- What if i, j, k are conflicting investments?

$$x_i + x_j + x_k \leq 1$$

CPU Job Assignment Problem

- There are *n* jobs that must be completed
- There are *m* CPUs available to do the jobs
  - Each CPU can do at most one job
- There is a cost associated with running a particular job on a particular CPU
- How to assign jobs to CPUs, minimizing total cost?

Variables

$$x_{c,j} = \begin{cases} 1 & \text{if CPU } c \text{ gets job } j \\ 0 & \text{else} \end{cases}$$

```
for c in range(num_cpus):
    for j in range(num_jobs):
        x[c, j] = model.IntVar(0, 1, f'cpu {c} gets job {j}')
```

Constraint

- Each CPU gets at most 1 job

$$\text{For each CPU } c, \sum_{j \in \text{jobs}} x_{c,j} \leq 1$$

```
# Each cpu gets at most one job
for c in range(num_cpus):
    model.Add(
        sum(x[c, j] for j in range(num_jobs)) <= 1
    )
```

Constraint

- Each Job gets exactly one CPU

For a job $j^\star$, over all CPUs, exactly one of $x_{c_1,j^\star}, x_{c_2,j^\star}, ..., x_{c_n,j^\star}$ equals 1

```
# Each job gets exactly one CPU
for j in range(num_jobs):
    model.Add(
        sum(x[c, j] for c in range(num_cpus)) == 1
    )
```

Enjoy Fall Break!

"I would rather sit on a pumpkin, and have it all to myself, than be crowded on a velvet cushion." ~Henry David Thoreau