Linear Programming

DPV Chapter 7, Part 1

Jim Royer

March 25, 2019

Uncredited diagrams are from DPV or homemade.

Linear Programming: Introduction, 1

Ingredients of Linear Programming

- a set of variables
- a set of linear equations/inequalities over these variables
- a linear objective function to min/max

An Example Problem: Maximizing Profits

Item	# made per day	profit per unit	max # per day
1	x_1	\$1	200
2	x_2	\$6	300

We also cannot make more than a total of 400 units per day of items 1 and 2 combined.
Q: What should x_1 and x_2 be to maximize profits?

Sample Problem, Continued

An Example Problem: Maximizing Profits

Item	# made per day	profit per unit	max # per day
1	x_1	\$1	200
2	x_2	\$6	300

We also cannot make more than a total of 400 units per day of items 1 and 2 combined.

The problem expressed as a linear program

Objective function	max: $x_1 + 6x_2$
Constraints	$x_1 \leq 200$
	$x_2 \leq 300$
	$x_1 + x_2 \leq 400$
	$x_1, x_2 \geq 0$

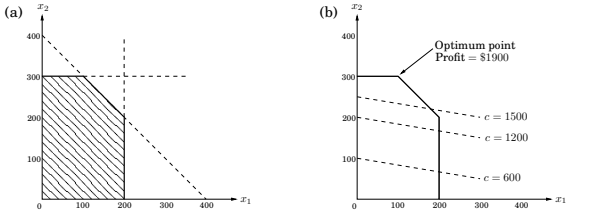
Sample Problem, Continued

The problem expressed as a linear program

Objective function
max: $x_1 + 6x_2$

Constraints
$x_1 \leq 200$
$x_2 \leq 300$
$x_1 + x_2 \leq 400$
$x_1, x_2 \geq 0$

Figure 7.1 (a) The feasible region for a linear program. (b) Contour lines of the objective function: $x_1 + 6x_2 = c$ for different values of the profit c.



Linear Programming, 2

- The constraints describe a convex polygon. *(Why?)*

Generally . . .

The optimum is achieved at a vertex of the feasible region. *(Why?)*

. . . except when there is no optimum. E.g.:

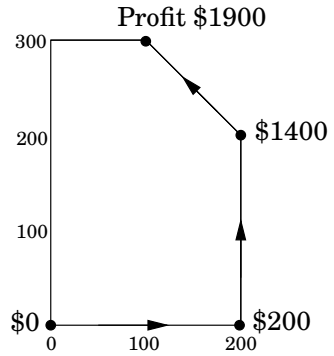
- the linear program is *infeasible*, meaning the constraints are so tight it is impossible to satisfy them all. E.g.: $x \leq 1, x \geq 2$.
- the feasible region is *unbounded*, meaning the constraints are so loose that the objective function takes on arbitrarily large values. E.g.: maximize $x_1 + x_2$ under $x_1, x_2 \geq 0$.

How to Solve Linear Programs

The Simplex Method

- Warning: Far from the whole story!!
- The feasible region is a polygon.
- Suppose p is a vertex of the feasible region polygon.
- p is a *local maximum* when the value of the objective function is larger at p and at any of its neighbors.
- The method *hill-climbs*.

```
p ← the origin
while p is not a local max do
    p ← the neighbor at which the
          obj. function is larger
return p
```



?? Why is the local max the true max?

Sample Problem, Continued

An Example Problem: Maximizing Profits

Item	# made per day	profit per unit	max # per day
1	x_1	\$1	200
2	x_2	\$6	300
3	x_3	\$13	

Other production constraints: $x_1 + x_2 + x_3 \leq 400$ and $x_2 + 3x_3 \leq 600$.

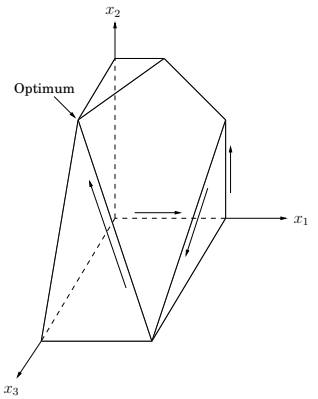
The problem expressed as a linear program

Objective function	max: $x_1 + 6x_2 + 13x_3$
Constraints	$x_1 \leq 200$
	$x_2 \leq 300$
	$x_1 + x_2 + x_3 \leq 400$
	$x_2 + 3x_3 \leq 400$
	$x_1, x_2, x_3 \geq 0$

Sample Problem, Continued

The problem expressed as a linear program

Objective function	max: $x_1 + 6x_2 + 13x_3$
Constraints	$x_1 \leq 200$
	$x_2 \leq 300$
	$x_1 + x_2 + x_3 \leq 400$
	$x_2 + 3x_3 \leq 400$
	$x_1, x_2, x_3 \geq 0$



$$(0,0,0) \xrightarrow{\$0} (200,0,0) \xrightarrow{\$200} (200,200,0) \xrightarrow{\$1400} (200,0,200) \xrightarrow{\$2800} (0,300,100) \xrightarrow{\$3100}$$

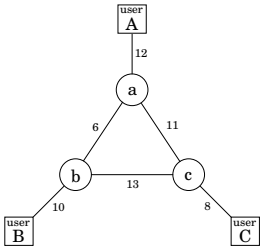
Notes

- At higher dimensions, things are harder to visualize, but the math still works!
- There are lots of professional packages that quickly and accurately solve linear programming problems.
- So the key thing to know is: how to precisely express a problem as a LP-problem.

Bandwidth Allocation, 1

- We have a network as shown in the figure.
- The numbers on edges are max-bandwidth capacities.
- We need connections between users A&B, A&C, and B&C.
- Each connection requires at least two units of bandwidth, but more can be used.
- Connection A–B pays \$3 per bandwidth unit; A–C pays \$4 and B–C pays \$2.
- Each connection can be routed via the short route, long route, or a combination of the two.

?? How do we route connections to maximize profit?

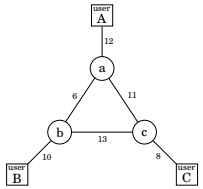


Bandwidth Allocation, 2

- Each connection needs at least two units of bandwidth.
- Connection A–B pays \$3 per bandwidth unit; A–C pays \$4 and B–C pays \$2.

?? How do we route connections to maximize profit?

$$
\begin{aligned}
\max\ & 3x_{AB} + 3x'_{AB} + 2x_{BC} + 2x'_{BC} + 4x_{AC} + 4x'_{AC} \\
& x_{AB} + x'_{AB} + x_{BC} + x'_{BC} \le 10 && \text{[edge } (b, B)\text{]} \\
& x_{AB} + x'_{AB} + x_{AC} + x'_{AC} \le 12 && \text{[edge } (a, A)\text{]} \\
& x_{BC} + x'_{BC} + x_{AC} + x'_{AC} \le 8 && \text{[edge } (c, C)\text{]} \\
& x_{AB} + x'_{BC} + x'_{AC} \le 6 && \text{[edge } (a, b)\text{]} \\
& x'_{AB} + x_{BC} + x'_{AC} \le 13 && \text{[edge } (b, c)\text{]} \\
& x'_{AB} + x'_{BC} + x_{AC} \le 11 && \text{[edge } (a, c)\text{]} \\
& x_{AB} + x'_{AB} \ge 2 \\
& x_{BC} + x'_{BC} \ge 2 \\
& x_{AC} + x'_{AC} \ge 2 \\
& x_{AB}, x'_{AB}, x_{BC}, x'_{BC}, x_{AC}, x'_{AC} \ge 0
\end{aligned}
$$



Solution via Simplex:

$$
\begin{aligned}
x_{AB} &= 0.0 \\
x'_{AB} &= 7.0 \\
x_{BC} &= 1.5 \\
x'_{BC} &= 1.5 \\
x_{AC} &= 0.5 \\
x'_{AC} &= 4.5
\end{aligned}
$$

★ x_{UV} = the **short path** allocation for U-V

★ x'_{UV} = the **long path** allocation for U-V

Variations on Linear Programming, 1

LP in Standard Form

- the objective function is minimized
- the variables are constrained to be ≥ 0
- the other constraints are equations

LP in General

- the objective function is minimized or maximized
- the variables may take on negative values
- the other constraints are equations or inequalities

Reduction to Standard Form

$$
\begin{aligned}
\max: f &\iff \min: (-f) \\
x \text{ positive/negative} &\iff x^+, x^- \ge 0 \ \ \& \ \ x = x^+ - x^- \\
a_1 x_1 + \cdots + a_n x_n \le b &\iff (\textstyle\sum a_i x_i) + s = b \ \ \& \ \ s \ge 0
\end{aligned}
$$

s as above is called a slack variable.

Variations on Linear Programming, 2

Reduction to Standard Form

$$\begin{aligned} \max: f &\iff \min: (-f) \\ x \text{ positive/negative} &\iff x^+, x^- \geq 0 \ \ \& \ \ x = x^+ - x^- \\ a_1x_1 + \cdots + a_nx_n \leq b &\iff (\sum a_ix_i) + s = b \ \ \& \ \ s \geq 0 \end{aligned}$$

Example

$$\begin{array}{rcl} \max: x_1 + 6x_2 & & \min: -x_1 - 6x_2 \\ x_1 \leq 200 & & x_1 + s_1 = 200 \\ x_2 \leq 300 & \Longrightarrow & x_2 + s_2 = 300 \\ x_1 + x_2 \leq 400 & & x_1 + x_2 + s_3 = 400 \\ x_1, x_2 \geq 0 & & x_1, x_2, s_1, s_2, s_3 \geq 0 \end{array}$$

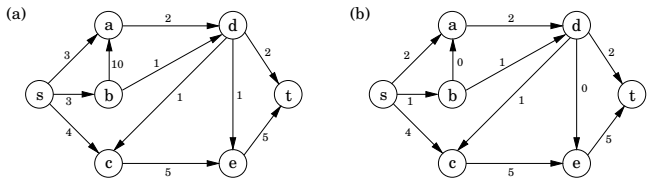
Linear Programming in Matrix Form

Example

$$\begin{array}{rcl} \max: x_1 + 6x_2 & & \max: \vec{c}^T \cdot \vec{x} \\ x_1 \leq 200 & \Longrightarrow & A\vec{x} \leq \vec{b} \\ x_2 \leq 300 & & \vec{x} \geq 0 \\ x_1 + x_2 \leq 400 & & \\ x_1, x_2 \geq 0 & & \end{array}$$

where $\vec{c} = \begin{pmatrix} 1 \\ 6 \end{pmatrix}$, $\vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, $\vec{b} = \begin{pmatrix} 200 \\ 300 \\ 400 \end{pmatrix}$, $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$

Network Flows



(a) A network of oil pipelines.
The number on each edge is its maximum capacity.
No storage in nodes. s = source node. t = target node.
Goal: Send as much oil from s to t as the network allows.

(b) A flow in the network, shipping 7 units.
Can we do better? How do we find out?

Network Flow as an LP Problem

- A variable for each edge: f_e = the flow on that edge.
  $0 \leq f_e \leq c_e$ = the capacity of edge e.
- Flow is conserved at each node except s and t:

$$\sum_{(w,u) \in E} f_{wu} = \sum_{(u,z) \in E} f_{uz}$$

- The size of the flow is the amount sent from s to t. By conservation:

$$size(f) = \sum_{(s,u) \in E} f_{su}.$$

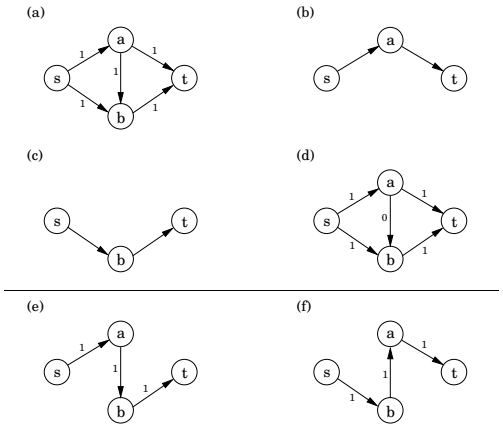
- **Goal:** Assign values to the f_{uv}'s that satisfy the constraints and maximize $size(f)$.

$\therefore$ This is a linear programming problem,
so Simplex can solve these problems for us.

From Simplex to a Direct Network Flow Algorithm, 1

If you analyze how Simplex solves Network Flow Problems, it is roughly equivalent to:

Start with zero flow.
Repeat
  Choose an appropriate path from s to t
  Increase the flow along the path's edges as much as possible
until there is no path that improves the flow



(a) Initial network.
(b) First path chosen.
(c) Second path chosen.
(d) Final flow.
(e) Alternative first path.
(f) Alternative second path.
The $b \to a$ edge cancels flow.

From Simplex to a Direct Network Flow Algorithm, 2

- Appropriate paths consists of two sorts of edges (u, v):
  1. (u, v) is in the network and not yet at full capacity.
     (u, v) can handle up to $c_{uv} - f_{uv}$ more flow.
  2. (v, u) is in the network with flow along it.
     (u, v) can handle up to f_{vu} more flow — by canceling flow along (v, u).
- $G^f = (V, E^f)$ is the *residual network* with two sorts of edges, (u, v), with *residual capacities:*

$$\begin{cases} c_{uv} - f_{uv}, & \text{if } (u,v) \in E \text{ and } f_{uv} < c_{uv}; \\ f_{vu}, & \text{if } (v,u) \in E \text{ and } f_{vu} > 0. \end{cases}$$

G^f = the network of unused capacities.

The "Simplex Strategy"

Start with zero flow.
Repeat
  Choose an appropriate path from s to t
  Increase the flow along the path's edges as much as possible
until there is no path that improves the flow

From Simplex to a Direct Network Flow Algorithm, 2

The "Simplex Strategy"

Start with zero flow.
Repeat
  Choose an appropriate path from s to t
  Increase the flow along the path's edges as much as possible
until there is no path that improves the flow

$G^f = (V, E^f)$ is the *residual network* with edges (u, v) with

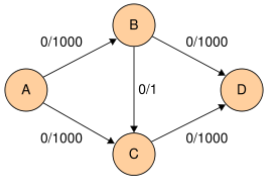
residual capacities: $\begin{cases} c_{uv} - f_{uv}, & \text{if } (u,v) \in E \text{ and } f_{uv} < c_{uv}; \\ f_{vu}, & \text{if } (v,u) \in E \text{ and } f_{vu} > 0. \end{cases}$

The "Simplex Strategy" in terms of G^f

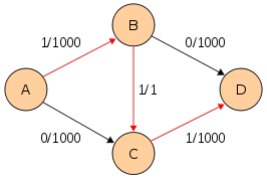
$G^f \leftarrow G$ // Initially, capacity = unused flow
Repeat forever
  Use BFS to find a path in G^f along which we can increase flow
  if there is no such path **then quit**
  if there is such a path **then** revise f and G^f

An Example

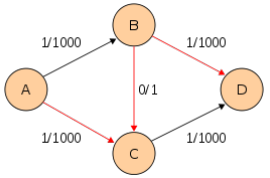
$G^f \leftarrow G$
// Initially, capacity = unused flow
Repeat forever
  Use BFS to find a path in G^f along which we can increase flow
  if there is such a path
    then revise f and G^f
    else quit



Initially



After Step 1



After Step 2

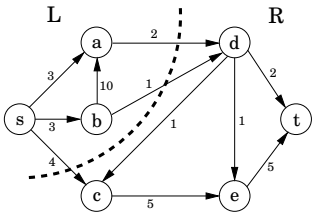
...

Flows and Cuts, 1

Definition

An s,t-*cut* is a partition of V into two sets L and R with $s \in L$ and $t \in R$.

$capacity(L,R)$
$= \sum\{ c_{uv} : (u,v) \in E,\ u \in L,\ v \in R \}$.



Claim

For any flow f and any s,t-cut (L,R):
$size(f) \le capacity(L,R)$.

The Max-Flow Min-Cut Theorem

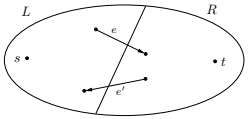
The max flow in a network = $\min\{ capacity(L,R) : (L,R)$ is an s,t-cut $\}$.

Also, our algorithm finds the min-cut.

Flows and Cuts, 2

The Max-Flow Min-Cut Theorem

The max flow in a network =
$\min\{ capacity(L,R) : (L,R)$ is an s,t-cut $\}$.



Proof: Let

- f = the algorithm's final flow
- L = nodes reachable (with positive capacity edges) from s in G^f. **Note:** $t \notin L$! *(Why?)*
- $R = V - L$.

Claim: $size(f) = capacity(L,R)$.
Proof of Claim:

- The L to R edges must be at full capacity in G^f. *(Why?)*
- The R to L edges must have 0 flow in f. *(Why?)*

Therefore, the claim follows.
Thus, (L,R) must be a min-cut. *(Why?)*
Therefore, the theorem follows.

Max Flow, Runtime

The "Simplex Strategy" in terms of G^f

$G^f \leftarrow G$
Repeat forever
  Use BFS to find a path in G^f along which we can increase flow
  if there is no such path **then quit**
  if there is such a path **then** revise f and G^f

- Each iteration takes $O(|E|)$ time. *(Why?)*
- With some work *(see Exercise 7.31)* one can show that there are at most $O(|V| \cdot |E|)$ many iterations.
- Therefore, the total run time is $O(|V| \cdot |E|^2)$ time.