

# Linear Programming

## DPV Chapter 7, Part 2

Jim Royer

Part 2

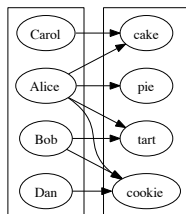
March 25, 2019

Uncredited diagrams are from DPV or homemade.

# Bipartite matching, 1

## Definition

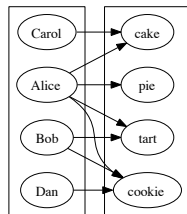
- (a) A graph  $G = (V, E)$  is *bipartite* when  $V$  can be partitioned into two nonempty sets  $L$  and  $R$  and each edge in  $E$  has one endpoint in  $L$  and the other in  $R$ .
- (b) Given a bipartite graph  $G$ , a *bipartite matching* is a set of edges which give an 1-1 correspondence between  $L$  and  $R$ . (E.g.,  $\{(Alice, pie), (Bob, tart), (Carol, cake), (Dan, cookie)\}$ )



# Bipartite matching, 1

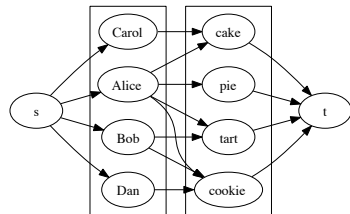
## Definition

- (a) A graph  $G = (V, E)$  is *bipartite* when  $V$  can be partitioned into two nonempty sets  $L$  and  $R$  and each edge in  $E$  has one endpoint in  $L$  and the other in  $R$ .
- (b) Given a bipartite graph  $G$ , a *bipartite matching* is a set of edges which give an 1-1 correspondence between  $L$  and  $R$ . (E.g.,  $\{(Alice, pie), (Bob, tart), (Carol, cake), (Dan, cookie)\}$ )



We can turn the problem of finding a bipartite matching into a network flow problem by:

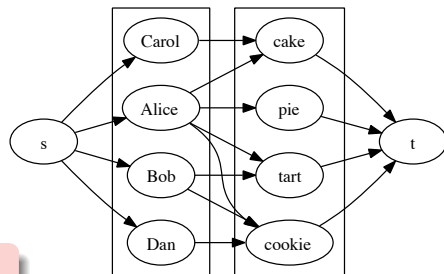
- Adding a source and target vertices  $s$  and  $t$ .
- Adding edges from  $s$  to each  $u \in L$ .
- Adding edges from each  $v \in R$  to  $t$ .
- Setting the capacity of each edge 1.



# Bipartite matching, 2

bipartite matching  $\rightarrow$  network flow

- Add a source and target vertices  $s$  and  $t$ .
- Add edges from  $s$  to each  $u \in L$ .
- Add edges from each  $v \in R$  to  $t$ .
- Set the capacity of each edge 1.



## Problem

We do **not** want fractional flows.  
(E.g., *Dan wants a whole cookie!*)

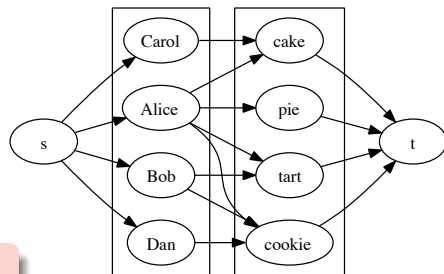
## Saving Fact

If all edge capacities are integers, then our algorithm finds an integral optimal flow.

# Bipartite matching, 2

bipartite matching  $\rightarrow$  network flow

- Add a source and target vertices  $s$  and  $t$ .
- Add edges from  $s$  to each  $u \in L$ .
- Add edges from each  $v \in R$  to  $t$ .
- Set the capacity of each edge 1.



## Problem

We do **not** want fractional flows.  
(E.g., *Dan wants a whole cookie!*)

## Saving Fact

If all edge capacities are integers, then our algorithm finds an integral optimal flow.

## BUT!

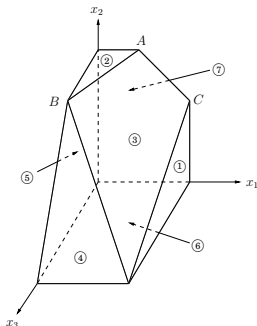
In general it is hard to find solutions to LP problems if you demand all variables have integer values.  
(Set up for Chapter 8.)

# Simplex at a very high level, 1

$v \leftarrow$  any vertex in the **feasible region**

**while** there is a neighbor  $v'$  of  $v$  with a better objective value **do**  $v \leftarrow v'$

- With  $n$  variables, we are working in  $\mathbb{R}^n$ .
- The **feasible region** is defined by the constraints:
  - ▶  $a_1x_1 + \dots + a_nx_n = b$  defines a hyperplane in  $\mathbb{R}^n$
  - ▶  $a_1x_1 + \dots + a_nx_n \leq b$  and  $a_1x_1 + \dots + a_nx_n \geq b$  define halfspaces in  $\mathbb{R}^n$ .
  - ▶ The intersection of halfspaces gives a convex polyhedron.



$$\begin{array}{ll} \max & x_1 + 6x_2 + 13x_3 \\ & x_1 \leq 200 & \textcircled{1} \\ & x_2 \leq 300 & \textcircled{2} \\ & x_1 + x_2 + x_3 \leq 400 & \textcircled{3} \\ & x_2 + 3x_3 \leq 600 & \textcircled{4} \\ & x_1 \geq 0 & \textcircled{5} \\ & x_2 \geq 0 & \textcircled{6} \\ & x_3 \geq 0 & \textcircled{7} \end{array}$$

# Simplex at a very high level, 2

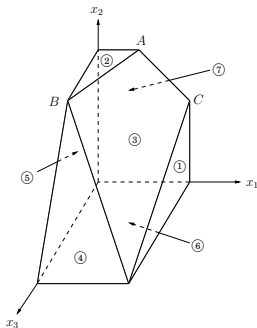
$v \leftarrow$  any **vertex** in the feasible region

**while** there is a neighbor  $v'$  of  $v$  with a better objective value **do**  $v \leftarrow v'$

**Q:** What is a vertex?    **A:** A “corner.”

- ▶ In 2-D, two lines intersect in a point\*.
- ▶ In 3-D, three planes intersect in a point\*.  
E.g., ①, ②, and ⑦ with equality meet at point A.
- ▶ In  $n$ -D,  $n$  hyperplanes intersect in a point\*.

**\*Watch out for degeneracy!**



$$\begin{aligned} \max \quad & x_1 + 6x_2 + 13x_3 \\ & x_1 \leq 200 & \text{①} \\ & x_2 \leq 300 & \text{②} \\ & x_1 + x_2 + x_3 \leq 400 & \text{③} \\ & x_2 + 3x_3 \leq 600 & \text{④} \\ & x_1 \geq 0 & \text{⑤} \\ & x_2 \geq 0 & \text{⑥} \\ & x_3 \geq 0 & \text{⑦} \end{aligned}$$

# Simplex at a very high level, 3

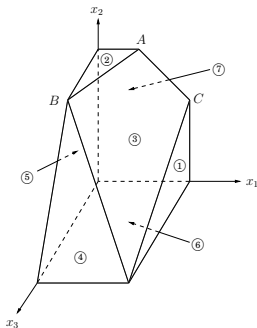
$v \leftarrow$  any vertex in the feasible region

**while** there is a neighbor  $v'$  of  $v$  with a better objective value **do**  $v \leftarrow v'$

Q: What is a neighbor of a vertex?

A: They have  $n - 1$  defining constraints in common.

E.g., A and C share ③ and ⑦.



$$\max x_1 + 6x_2 + 13x_3$$

$$x_1 \leq 200 \quad \textcircled{1}$$

$$x_2 \leq 300 \quad \textcircled{2}$$

$$x_1 + x_2 + x_3 \leq 400 \quad \textcircled{3}$$

$$x_2 + 3x_3 \leq 600 \quad \textcircled{4}$$

$$x_1 \geq 0 \quad \textcircled{5}$$

$$x_2 \geq 0 \quad \textcircled{6}$$

$$x_3 \geq 0 \quad \textcircled{7}$$



# Simplex: Towards the Algorithm

## The algorithm has two basic jobs

- job 1: See if the current vertex is optimal. (If so, halt.)
- job 2: Determine where to move next.

Both jobs are easy at the origin:  $\vec{0} = (0, \dots, 0)$ .

## Why is the origin so special?

$$\begin{aligned} \max : & \vec{c}^T \vec{x} \\ A\vec{x} & \leq \vec{b} \\ \vec{x} & \geq \vec{0} \end{aligned}$$

Suppose  $\vec{0}$  is feasible.

- job 1:  $\vec{0}$  is optimal  $\iff c_i \leq 0$  for each  $i$ . (Why?)
- job 2: If  $\vec{0}$  is not optimal, pick some  $i$  with  $c_i > 0$  and increase  $x_i$  until we hit another constraint.

## What do we do if we are not at the origin?

Transform coordinates to move the origin to where we are.

## Simplex: Moving the origin

- Suppose we want to move the origin to  $\vec{u}$ .
- Suppose  $\vec{u}$  is given by the constraints:  $\vec{a}_i \cdot \vec{x} \leq b_i$  where  $i = 1, \dots, n$ .
- For  $i = 1, \dots, n$ , let  $y_i = b_i - \vec{a}_i \cdot \vec{x}$ . Then:

in $\vec{x}$ -space		in $\vec{y}$ -space
$\vec{a}_i \cdot \vec{x} \leq b_i$	$\rightsquigarrow$	$y_i \geq 0$
$\vec{u}$	$\rightsquigarrow$	$\vec{0}$
max: $\vec{c}^T \cdot \vec{x}$	$\rightsquigarrow$	max: $(c_u + \vec{c}'^T \cdot \vec{y})$

where  $c_u$  = the cost at  $\vec{u}$  and  
 $\vec{c}'$  = the transformed cost vector

# Example from DPV, 1

Initial LP:

$$\begin{aligned} \max \quad & 2x_1 + 5x_2 \\ 2x_1 - x_2 & \leq 4 & \textcircled{1} \\ x_1 + 2x_2 & \leq 9 & \textcircled{2} \\ -x_1 + x_2 & \leq 3 & \textcircled{3} \\ x_1 & \geq 0 & \textcircled{4} \\ x_2 & \geq 0 & \textcircled{5} \end{aligned}$$

Current vertex:  $\{\textcircled{4}, \textcircled{5}\}$  (origin).

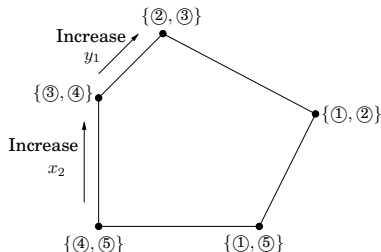
Objective value: 0.

Move: increase  $x_2$ .

$\textcircled{5}$  is released,  $\textcircled{3}$  becomes tight. Stop at  $x_2 = 3$ .

New vertex  $\{\textcircled{4}, \textcircled{3}\}$  has local coordinates  $(y_1, y_2)$ :

$$y_1 = x_1, \quad y_2 = 3 + x_1 - x_2$$



# Example from DPV, 2

Rewritten LP:

$$\begin{aligned} \max \quad & 15 + 7y_1 - 5y_2 \\ & y_1 + y_2 \leq 7 \quad \textcircled{1} \\ & 3y_1 - 2y_2 \leq 3 \quad \textcircled{2} \\ & y_2 \geq 0 \quad \textcircled{3} \\ & y_1 \geq 0 \quad \textcircled{4} \\ & -y_1 + y_2 \leq 3 \quad \textcircled{5} \end{aligned}$$

*Current vertex:*  $\{\textcircled{4}, \textcircled{3}\}$ .

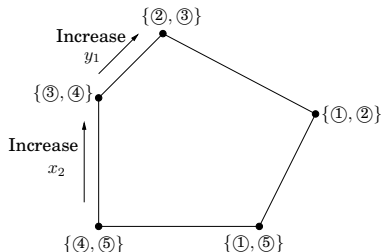
*Objective value:* 15.

*Move:* increase  $y_1$ .

$\textcircled{4}$  is released,  $\textcircled{2}$  becomes tight. Stop at  $y_1 = 1$ .

New vertex  $\{\textcircled{2}, \textcircled{3}\}$  has local coordinates  $(z_1, z_2)$ :

$$z_1 = 3 - 3y_1 + 2y_2, \quad z_2 = y_2$$



# Example from DPV, 3

Rewritten LP:

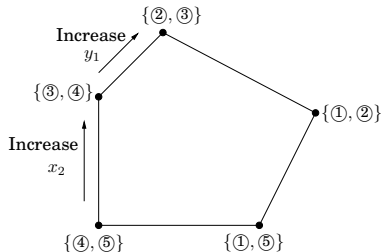
$$\begin{aligned} \max \quad & 22 - \frac{7}{3}z_1 - \frac{1}{3}z_2 \\ -\frac{1}{3}z_1 + \frac{5}{3}z_2 & \leq 6 & \textcircled{1} \\ z_1 & \geq 0 & \textcircled{2} \\ z_2 & \geq 0 & \textcircled{3} \\ \frac{1}{3}z_1 - \frac{2}{3}z_2 & \leq 1 & \textcircled{4} \\ \frac{1}{3}z_1 + \frac{1}{3}z_2 & \leq 4 & \textcircled{5} \end{aligned}$$

*Current vertex:*  $\{\textcircled{2}, \textcircled{3}\}$ .

*Objective value:* 22.

*Optimal:* all  $c_i < 0$ .

Solve  $\textcircled{2}, \textcircled{3}$  (in original LP) to get optimal solution  $(x_1, x_2) = (1, 4)$ .



## Linear Programming

2019-03-25

└ Example from DPV, 3

maximize LP: $\max z = 2x_1 + 3x_2$ $x_1 + 2x_2 \leq 6$ ① $x_1 \geq 0$ ② $x_2 \geq 0$ ③ $x_1 + x_2 \leq 4$ ④ $x_1 + 3x_2 \leq 4$ ⑤	Current vertex: ②, ③: Objective value: 12. Optimal: all $x_1 < 6$ . Solve ②, ③ (in original LP) to get optimal solution $(x_1, x_2) = (1, 4)$ .
--	---



$$(x_1, x_2) = (1, 4)$$

## Simplex Issues: *Finding the starting vertex*

- The origin is not always in the feasible region.

**Q:** So how to find a starting vertex?

**A:** It turns out this is another LP problem:

Suppose we have  $\min: \vec{c}^T \cdot \vec{x} \quad \ni \quad A\vec{x} = \vec{b}$  and  $\vec{x} \geq \vec{0}$ .

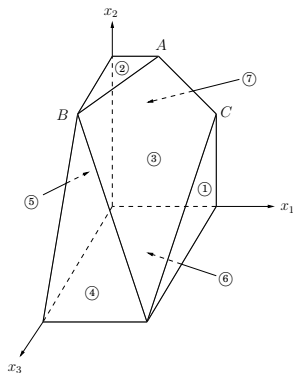
Then:

- ▶ Make sure each  $b_i \geq 0$ .  
If  $b_i < 0$ , multiply both sides of the  $i$ -th eqn. by  $-1$ .
- ▶ Create new variables  $z_1, \dots, z_m \geq 0$  where  $m = \#$  of equations.
- ▶ Add  $z_i$  to the left-hand side of equation  $i$ .
- ▶  $\min: z_1 + \dots + z_m$  is the objective function.
- ▶  $z_1 = b_1, \dots, z_m = b_m$  and all other variables = 0 is a starting vertex.

**Case:** Simplex reports 0 is the optimal value of the objective function.  
Then ignore the  $z_i$ 's and the value of the other vars gives a starting vertex.

**Case:** Simplex reports 0 is **not** the opt. value of the objective function.  
Then the original LP problem must be infeasible. *(Why?)*

# Simplex Issues: *Degeneracy*



$$\begin{aligned} \max \quad & x_1 + 6x_2 + 13x_3 \\ & x_1 \leq 200 & \textcircled{1} \\ & x_2 \leq 300 & \textcircled{2} \\ & x_1 + x_2 + x_3 \leq 400 & \textcircled{3} \\ & x_2 + 3x_3 \leq 600 & \textcircled{4} \\ & x_1 \geq 0 & \textcircled{5} \\ & x_2 \geq 0 & \textcircled{6} \\ & x_3 \geq 0 & \textcircled{7} \end{aligned}$$

Vertex B is degenerate in that it is given by any one of:

- ②, ③, ④
- ②, ③, ⑤
- ②, ④, ⑤
- ③, ④, ⑤



# Simplex Issues: *Degeneracy & Unboundedness*

- Degeneracy can confuse poor Simplex — causing it to return suboptimal answers.
- One way out is to perturb the  $b_i$ 's by a small amount, i.e.,  $b_i \mapsto b_i \pm \epsilon_i$ .
- This jolts each plan a bit and splits degenerate vertices into multiple verts.

## *Unboundedness*

- Simplex can detect if the feasible region is unbounded in which case it halts and fusses.

# Simplex Running Time

- With some cleverness (see DPV) each iteration can be gotten down to  $O(m \cdot n)$  time, where

$m$  = the number of inequalities

$n$  = number of variables

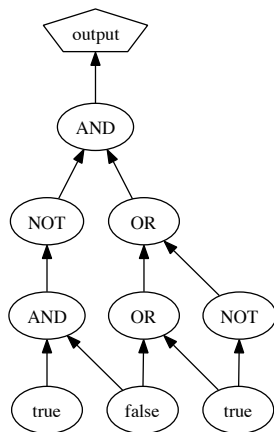
- There are  $\binom{m+n}{n}$  many vertices, which is exponential size.
- There are cases where Simplex takes exponential time!!
- In practice, Simplex usually runs much faster than any other LP algorithm — including Karmarkar's poly-time LP algorithm.

# Boolean Circuit Evaluation

## Boolean Circuit

A directed, acyclic graph of gates of the following sorts:

- inputs gates: indegree 0, labeled True or False
- AND gates, OR gates: indegree 2
- NOT gates: indegree 1
- One of the gates is chosen as the *output gate*



# Boolean Circuit Evaluation as an LP problem

- For each gate, create a variable  $x_g$  constrained:  
$$0 \leq x_g \leq 1.$$
- If  $g$  is a True-input gate, add:  
$$x_g = 1.$$
- If  $g$  is a False-input gate, add:  
$$x_g = 0.$$
- If  $g$  is an Or-gate with inputs from  $h$  and  $h'$ , add:  
$$x_g \geq x_h, \quad x_g \geq x_{h'}, \quad x_g \leq x_h + x_{h'}$$
- If  $g$  is an And-gate with inputs from  $h$  and  $h'$ , add:  
$$x_g \leq x_h, \quad x_g \leq x_{h'}, \quad x_g \geq x_h + x_{h'} - 1$$
- If  $g$  is a Not-gate with input from  $h$ , add:  
$$x_g = 1 - x_h$$

# Boolean Circuit Evaluation as an LP problem—So?

- Boolean Circuit Evaluation is, in a certain sense (see Chapter 8), the most general problem solvable in polytime.

I.e., every polytime problem reduces to a Boolean Circuit Evaluation.

- Boolean Circuit Evaluation reduces to LP.

- • Every polytime problem reduces to LP!!

- Similarly, every polytime problem reduces to a Dynamic Programming problem.