CS 473: Algorithms

Chandra Chekuri   Ruta Mehta

University of Illinois, Urbana-Champaign

Fall 2016

Introduction to Linear Programming

Lecture 18

October 26, 2016

Part I

Introduction to Linear Programming

A Factory Example

Problem

Suppose a factory produces two products **1** and **2** using resources **A, B, C**.

1. Making a unit of **1** requires a unit each of **A** and **C**.
2. A unit of **2** requires one unit of **B** and **C**.
3. We have 200 units of **A**, 300 units of **B**, and 400 units of **C**.
4. Product **1** can be sold for **\$1** and product **2** for **\$6**.

How many units of product **1** and product **2** should the factory manufacture to maximize profit?

A Factory Example

Problem

Suppose a factory produces two products **1** and **2** using resources **A, B, C**.

1. Making a unit of **1** requires a unit each of **A** and **C**.
2. A unit of **2** requires one unit of **B** and **C**.
3. We have 200 units of **A**, 300 units of **B**, and 400 units of **C**.
4. Product **1** can be sold for **$1** and product **2** for **$6**.

How many units of product **1** and product **2** should the factory manufacture to maximize profit?

Solution:

A Factory Example

Problem

Suppose a factory produces two products **1** and **2** using resources **A, B, C**.

1. Making a unit of **1** requires a unit each of **A** and **C**.
2. A unit of **2** requires one unit of **B** and **C**.
3. We have 200 units of **A**, 300 units of **B**, and 400 units of **C**.
4. Product **1** can be sold for **$1** and product **2** for **$6**.

How many units of product **1** and product **2** should the factory manufacture to maximize profit?

Solution: Formulate as a linear program.

A Factory Example

Problem

Suppose a factory produces two products **1** and **2**, using resources **A, B, C**.

1. Making a unit of **1**: Req. one unit of **A**, **C**.
2. Making unit of **2**: Req. one unit of **B, C**.
3. Have **A**: 200, **B**: 300 , and **C**: 400.
4. Price of **1**: **\$1**, and **2**: **\$6**.

How many units of **1** and **2** to manufacture to max profit?

A Factory Example

Problem

Suppose a factory produces two products **1** and **2**, using resources **A, B, C**.

1. Making a unit of **1**: Req. one unit of **A**, **C**.
2. Making unit of **2**: Req. one unit of **B, C**.
3. Have **A**: 200, **B**: 300 , and **C**: 400.
4. Price of **1**: **$1**, and **2**: **$6**.

How many units of **1** and **2** to manufacture to max profit?

$$
\begin{array}{lll}
\textbf{max} & x_1 + 6x_2 & \\
\text{s.t.} & x_1 \leq 200 & \textbf{(A)} \\
 & x_2 \leq 300 & \textbf{(B)} \\
 & x_1 + x_2 \leq 400 & \textbf{(C)} \\
 & x_1 \geq 0 & \\
 & x_2 \geq 0 &
\end{array}
$$

Linear Programming Formulation

Let us produce $\mathbf{x_1}$ units of product 1 and $\mathbf{x_2}$ units of product 2. Our profit can be computed by solving

$$\begin{array}{ll} \text{maximize} & \mathbf{x_1 + 6x_2} \\ \text{subject to} & \mathbf{x_1 \le 200 \quad x_2 \le 300 \quad x_1 + x_2 \le 400} \\ & \mathbf{x_1, x_2 \ge 0} \end{array}$$

Linear Programming Formulation

Let us produce $\mathbf{x_1}$ units of product 1 and $\mathbf{x_2}$ units of product 2. Our profit can be computed by solving

$$\begin{array}{ll} \text{maximize} & \mathbf{x_1 + 6x_2} \\ \text{subject to} & \mathbf{x_1 \le 200} \quad \mathbf{x_2 \le 300} \quad \mathbf{x_1 + x_2 \le 400} \\ & \mathbf{x_1, x_2 \ge 0} \end{array}$$

What is the solution?

$-x_1 \le 0$

$a_{41} = -1 \quad a_{42} = 0 \quad b_4 = 0$

max: $\sum c_j x_j$

$c_1 = 1, \quad c_2 = 6$

$\forall i = 1..n \quad \sum a_{ij} x_j \le b_i$

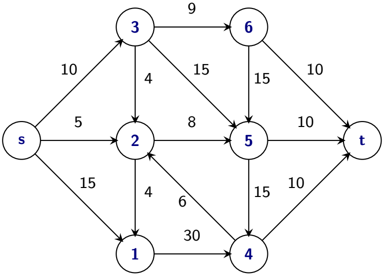
$a_{11} = 1 \quad a_{12} = 0 \quad b_1 = 200$

$a_{21} = 0 \quad a_{22} = 1 \quad b_2 = 300$

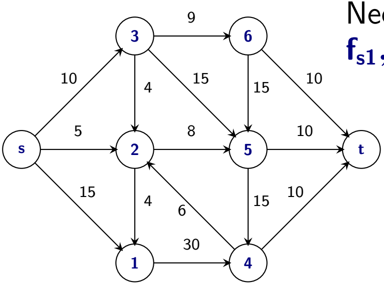
$a_{31} = 1 \quad a_{32} = 1 \quad b_3 = 400$

max: $[1, 6] \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ s.t. $\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \le \begin{bmatrix} 200 \\ 300 \\ 400 \end{bmatrix}$

Maximum Flow in Network

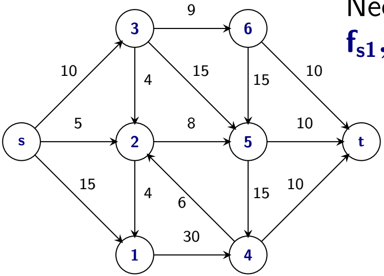


Maximum Flow in Network



Need to compute values $f_{s1}, f_{s2}, \ldots f_{25}, \ldots f_{5t}, f_{6t}$ such that

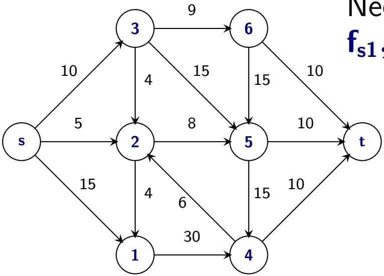
Maximum Flow in Network



Need to compute values $f_{s1}, f_{s2}, \ldots f_{25}, \ldots f_{5t}, f_{6t}$ such that

$f_{s1} \leq 15$	$f_{s2} \leq 5$	$f_{s3} \leq 10$
$f_{14} \leq 30$	$f_{21} \leq 4$	$f_{25} \leq 8$
$f_{32} \leq 4$	$f_{35} \leq 15$	$f_{36} \leq 9$
$f_{42} \leq 6$	$f_{4t} \leq 10$	$f_{54} \leq 15$
$f_{5t} \leq 10$	$f_{65} \leq 15$	$f_{6t} \leq 10$

Maximum Flow in Network



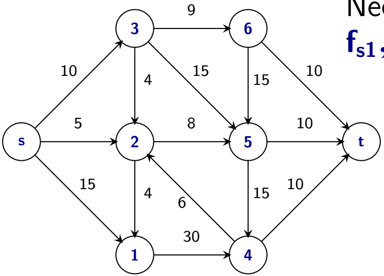
Need to compute values $f_{s1}, f_{s2}, \ldots f_{25}, \ldots f_{5t}, f_{6t}$ such that

$$
\begin{array}{lll}
f_{s1} \leq 15 & f_{s2} \leq 5 & f_{s3} \leq 10 \\
f_{14} \leq 30 & f_{21} \leq 4 & f_{25} \leq 8 \\
f_{32} \leq 4 & f_{35} \leq 15 & f_{36} \leq 9 \\
f_{42} \leq 6 & f_{4t} \leq 10 & f_{54} \leq 15 \\
f_{5t} \leq 10 & f_{65} \leq 15 & f_{6t} \leq 10
\end{array}
$$

and

$$
\begin{array}{lll}
f_{s1} + f_{21} = f_{14} & f_{s2} + f_{32} = f_{21} + f_{25} & f_{s3} = f_{32} + f_{35} + f_{36} \\
f_{14} + f_{54} = f_{42} + f_{4t} & f_{25} + f_{35} + f_{65} = f_{54} + f_{5t} & f_{36} = f_{65} + f_{6t}
\end{array}
$$

Maximum Flow in Network



Need to compute values $\mathbf{f_{s1}, f_{s2}, \ldots f_{25}, \ldots f_{5t}, f_{6t}}$ such that

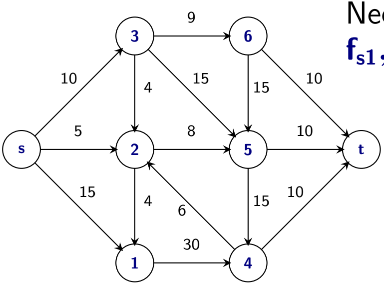
$$\begin{array}{lll} f_{s1} \leq 15 & f_{s2} \leq 5 & f_{s3} \leq 10 \\ f_{14} \leq 30 & f_{21} \leq 4 & f_{25} \leq 8 \\ f_{32} \leq 4 & f_{35} \leq 15 & f_{36} \leq 9 \\ f_{42} \leq 6 & f_{4t} \leq 10 & f_{54} \leq 15 \\ f_{5t} \leq 10 & f_{65} \leq 15 & f_{6t} \leq 10 \end{array}$$

and

$$\begin{array}{lll} f_{s1} + f_{21} = f_{14} & f_{s2} + f_{32} = f_{21} + f_{25} & f_{s3} = f_{32} + f_{35} + f_{36} \\ f_{14} + f_{54} = f_{42} + f_{4t} & f_{25} + f_{35} + f_{65} = f_{54} + f_{5t} & f_{36} = f_{65} + f_{6t} \end{array}$$

$$f_{s1} \geq 0 \quad f_{s2} \geq 0 \quad f_{s3} \geq 0 \quad \cdots \quad f_{4t} \geq 0 \quad f_{5t} \geq 0 \quad f_{6t} \geq 0$$

Maximum Flow in Network



Need to compute values $f_{s1}, f_{s2}, \ldots f_{25}, \ldots f_{5t}, f_{6t}$ such that

$$
\begin{array}{lll}
f_{s1} \leq 15 & f_{s2} \leq 5 & f_{s3} \leq 10 \\
f_{14} \leq 30 & f_{21} \leq 4 & f_{25} \leq 8 \\
f_{32} \leq 4 & f_{35} \leq 15 & f_{36} \leq 9 \\
f_{42} \leq 6 & f_{4t} \leq 10 & f_{54} \leq 15 \\
f_{5t} \leq 10 & f_{65} \leq 15 & f_{6t} \leq 10
\end{array}
$$

and

$$
\begin{array}{lll}
f_{s1} + f_{21} = f_{14} & f_{s2} + f_{32} = f_{21} + f_{25} & f_{s3} = f_{32} + f_{35} + f_{36} \\
f_{14} + f_{54} = f_{42} + f_{4t} & f_{25} + f_{35} + f_{65} = f_{54} + f_{5t} & f_{36} = f_{65} + f_{6t}
\end{array}
$$

$$
f_{s1} \geq 0 \quad f_{s2} \geq 0 \quad f_{s3} \geq 0 \quad \cdots \quad f_{4t} \geq 0 \quad f_{5t} \geq 0 \quad f_{6t} \geq 0
$$

and $f_{s1} + f_{s2} + f_{s3}$ is maximized.

Maximum Flow as a Linear Program

For a general flow network $\mathbf{G = (V, E)}$ with capacities $\mathbf{c_e}$ on edge $\mathbf{e} \in \mathbf{E}$, we have variables $\mathbf{f_e}$ indicating flow on edge $\mathbf{e}$

$$
\begin{array}{rll}
\text{Maximize} & \displaystyle\sum_{\text{e out of s}} f_e & \\
\text{subject to} & f_e \leq c_e & \text{for each } e \in E \\
& \displaystyle\sum_{\text{e out of v}} f_e - \sum_{\text{e into v}} f_e = 0 & \forall v \in V \setminus \{s, t\} \\
& f_e \geq 0 & \text{for each } e \in E.
\end{array}
$$

Maximum Flow as a Linear Program

For a general flow network $\mathbf{G = (V, E)}$ with capacities $\mathbf{c_e}$ on edge $\mathbf{e} \in \mathbf{E}$, we have variables $\mathbf{f_e}$ indicating flow on edge $\mathbf{e}$

$$\begin{aligned} \text{Maximize} \quad & \sum_{\mathbf{e} \text{ out of } \mathbf{s}} \mathbf{f_e} & \\ \text{subject to} \quad & \mathbf{f_e} \le \mathbf{c_e} & \text{for each } \mathbf{e} \in \mathbf{E} \\ & \sum_{\mathbf{e} \text{ out of } \mathbf{v}} \mathbf{f_e} - \sum_{\mathbf{e} \text{ into } \mathbf{v}} \mathbf{f_e} = \mathbf{0} & \forall \mathbf{v} \in \mathbf{V} \setminus \{\mathbf{s}, \mathbf{t}\} \\ & \mathbf{f_e} \ge \mathbf{0} & \text{for each } \mathbf{e} \in \mathbf{E}. \end{aligned}$$

Number of variables: $\mathbf{m}$, one for each edge.
Number of constraints: $\mathbf{m + n - 2 + m}$.

Minimum Cost Flow with Lower Bounds

... as a Linear Program

For a general flow network $\mathbf{G = (V, E)}$ with capacities $\mathbf{c_e}$, lower bounds $\ell_\mathbf{e}$, and costs $\mathbf{w_e}$, we have variables $\mathbf{f_e}$ indicating flow on edge $\mathbf{e}$. Suppose we want a min-cost flow of value at least $\mathbf{v}$.

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{e \in E} w_e f_e \\
\text{subject to} \quad & \sum_{e \text{ out of } s} f_e \geq v \\
& f_e \leq c_e \qquad f_e \geq \ell_e && \text{for each } e \in E \\
& \sum_{e \text{ out of } v} f_e - \sum_{e \text{ into } v} f_e = 0 && \text{for each } v \in V - \{s, t\} \\
& f_e \geq 0 && \text{for each } e \in E.
\end{aligned}
$$

Minimum Cost Flow with Lower Bounds

... as a Linear Program

For a general flow network $\mathbf{G = (V, E)}$ with capacities $\mathbf{c_e}$, lower bounds $\boldsymbol{\ell_e}$, and costs $\mathbf{w_e}$, we have variables $\mathbf{f_e}$ indicating flow on edge $\mathbf{e}$. Suppose we want a min-cost flow of value at least $\mathbf{v}$.

$$\text{Minimize} \sum_{e \in E} w_e f_e$$

$$\text{subject to} \sum_{e \text{ out of } s} f_e \geq v$$

$$f_e \leq c_e \qquad f_e \geq \ell_e \qquad \text{for each } e \in E$$

$$\sum_{e \text{ out of } v} f_e - \sum_{e \text{ into } v} f_e = 0 \qquad \text{for each } v \in V - \{s, t\}$$

$$f_e \geq 0 \qquad \text{for each } e \in E.$$

Number of variables: $\mathbf{m}$, one for each edge

Number of constraints: $\mathbf{1 + m + m + n - 2 + m = 3m + n - 1}$.

Linear Programs

Problem

Find a vector $\mathbf{x} \in \mathbb{R}^{\mathbf{d}}$ that

$$\begin{array}{lll} \text{maximize/minimize} & \sum_{j=1}^{d} c_j x_j & \\ \text{subject to} & \sum_{j=1}^{d} a_{ij} x_j \leq b_i & \text{for } i = 1 \ldots p \\ & \sum_{j=1}^{d} a_{ij} x_j = b_i & \text{for } i = p+1 \ldots q \\ & \sum_{j=1}^{d} a_{ij} x_j \geq b_i & \text{for } i = q+1 \ldots n \end{array}$$

Linear Programs

Problem

Find a vector $\mathbf{x} \in \mathbb{R}^d$ that

$$\begin{array}{lll} \text{maximize/minimize} & \sum_{j=1}^{d} c_j x_j & \\ \text{subject to} & \sum_{j=1}^{d} a_{ij} x_j \leq b_i & \text{for } i = 1 \ldots p \\ & \sum_{j=1}^{d} a_{ij} x_j = b_i & \text{for } i = p+1 \ldots q \\ & \sum_{j=1}^{d} a_{ij} x_j \geq b_i & \text{for } i = q+1 \ldots n \end{array}$$

Input is matrix $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{n \times d}$, column vector $\mathbf{b} = (b_i) \in \mathbb{R}^n$, and row vector $\mathbf{c} = (c_j) \in \mathbb{R}^d$

Canonical Form of Linear Programs

Canonical Form

A linear program is in canonical form if it has the following structure

$$\begin{array}{ll} \text{maximize} & \sum_{j=1}^{d} c_j x_j \\ \text{subject to} & \sum_{j=1}^{d} a_{ij} x_j \leq b_i \quad \text{for } i = 1 \ldots n \end{array}$$

Canonical Form of Linear Programs

max: x_1

Canonical Form

A linear program is in canonical form if it has the following structure

$$
\begin{array}{ll}
\text{maximize} & \sum_{j=1}^{d} c_j x_j \\
\text{subject to} & \sum_{j=1}^{d} a_{ij} x_j \le b_i \quad \text{for } i = 1 \ldots n
\end{array}
$$

$0 \le x_1 \le 1$

$x_1 < 1$

Conversion to Canonical Form

1. Replace $\sum_j a_{ij} x_j = b_i$ by $\sum_j a_{ij} x_j \le b_i$ and $-\sum_j a_{ij} x_j \le -b_i$
2. Replace $\sum_j a_{ij} x_j \ge b_i$ by $-\sum_j a_{ij} x_j \le -b_i$

Matrix Representation of Linear Programs

A linear program in canonical form can be written as

$$\begin{array}{ll} \text{maximize} & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} \leq \mathbf{b} \end{array}$$

where $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{n \times d}$, column vector $\mathbf{b} = (b_i) \in \mathbb{R}^n$, row vector $\mathbf{c} = (c_j) \in \mathbb{R}^d$, and column vector $\mathbf{x} = (x_j) \in \mathbb{R}^d$

1. Number of variable is $\mathbf{d}$
2. Number of constraints is $\mathbf{n}$

$x = \lambda x_1 + (1-\lambda) x_2$

$0 \leq \lambda \leq 1$

$\left.\begin{array}{l} \lambda (A x_1 \leq b) \\ (1-\lambda)(A x_2 \leq b) \end{array}\right\}$ Given

$$\begin{aligned} A \cdot x &= A(\lambda x_1 + (1-\lambda) x_2) \\ &= \lambda (A x_1) + (1-\lambda)(A x_2) \\ &\leq \lambda \bar{b} + (1-\lambda) \bar{b} = \bar{b}(\lambda + 1 - \lambda) = \bar{b} \end{aligned}$$

Other Standard Forms for Linear Programs

$$\begin{array}{ll} \text{maximize} & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

$$\begin{array}{ll} \text{minimize} & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

$$\begin{array}{ll} \text{minimize} & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

Linear Programming: A History

1. First formal application to problems in economics by Leonid Kantorovich in the 1930s
    1. However, work was ignored behind the Iron Curtain and unknown in the West
2. Rediscovered by Tjalling Koopmans in the 1940s, along with applications to economics
3. First algorithm (Simplex) to solve linear programs by George Dantzig in 1947
4. Kantorovich and Koopmans receive Nobel Prize for economics in 1975 ; Dantzig, however, was ignored
    1. Koopmans contemplated refusing the Nobel Prize to protest Dantzig's exclusion, but Kantorovich saw it as a vindication for using mathematics in economics, which had been written off as "a means for apologists of capitalism"

Back to the Factory example

Produce x_1 units of product 1 and x_2 units of product 2. Our profit can be computed by solving

$$\begin{array}{ll} \text{maximize} & x_1 + 6x_2 \\ \text{subject to} & x_1 \le 200 \quad x_2 \le 300 \quad x_1 + x_2 \le 400 \\ & x_1, x_2 \ge 0 \end{array}$$

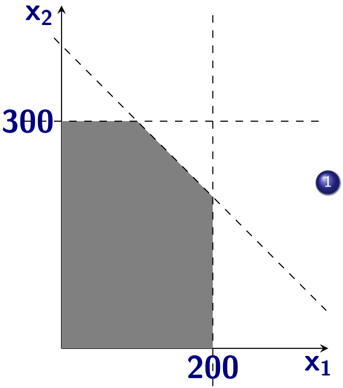
Back to the Factory example

Produce $\mathbf{x_1}$ units of product 1 and $\mathbf{x_2}$ units of product 2. Our profit can be computed by solving

$$\begin{array}{ll} \text{maximize} & \mathbf{x_1 + 6x_2} \\ \text{subject to} & \mathbf{x_1 \leq 200 \quad x_2 \leq 300 \quad x_1 + x_2 \leq 400} \\ & \mathbf{x_1, x_2 \geq 0} \end{array}$$

What is the solution?

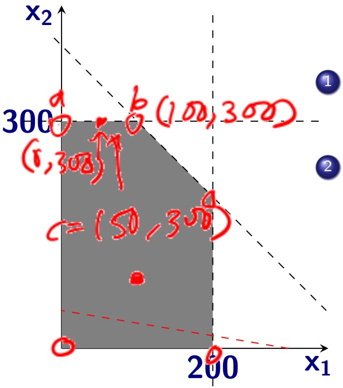
Solving the Factory Example



1. Feasible values of x_1 and x_2 are shaded region.

$$\begin{array}{llll} \text{maximize} & & x_1 + 6x_2 & \\ \text{subject to} & x_1 \leq 200 & x_2 \leq 300 & x_1 + x_2 \leq 400 \\ & & x_1, x_2 \geq 0 & \end{array}$$

Solving the Factory Example



1. Feasible values of x_1 and x_2 are shaded region.
2. Objective (Cost) function is a direction — the line represents all points with same value of the function

$c = \frac{1}{2} a + \frac{1}{2} b$

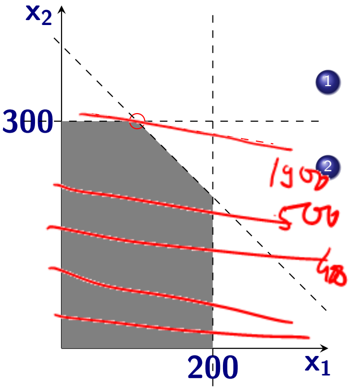
$\tau = \lambda\, 100 + (1-\lambda) 200$

$1 \geq \lambda \geq 0$

$100 \leq \tau \leq 200$

maximize		$x_1 + 6x_2$		
subject to	$x_1 \leq 200$	$x_2 \leq 300$	$x_1 + x_2 \leq 400$	
		$x_1, x_2 \geq 0$		

Solving the Factory Example



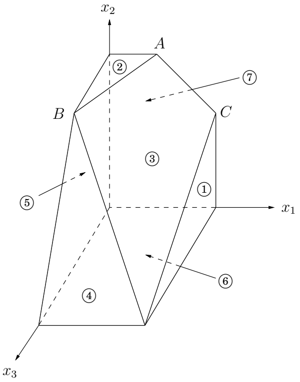
1. Feasible values of x_1 and x_2 are shaded region.
2. Objective (Cost) function is a direction — the line represents all points with same value of the function; moving the line until it just leaves the feasible region, gives optimal values.

$$\begin{aligned} \text{maximize} \quad & x_1 + 6x_2 \\ \text{subject to} \quad & x_1 \leq 200 \quad x_2 \leq 300 \quad x_1 + x_2 \leq 400 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Linear Programming in 2-d

1. Each constraint a half plane
2. Feasible region is intersection of finitely many half planes — it forms a polygon
3. For a fixed value of objective function, we get a line. Parallel lines correspond to different values for objective function.
4. Optimum achieved when objective function line just leaves the feasible region

An Example in 3-d



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

Figure from Dasgupta etal book.

Factory Example: Alternate View

Original Problem

Recall we have,

$$\begin{array}{ll} \text{maximize} & \mathbf{x_1 + 6x_2} \\ \text{subject to} & \mathbf{x_1 \leq 200 \quad x_2 \leq 300 \quad x_1 + x_2 \leq 400} \\ & \mathbf{x_1, x_2 \geq 0} \end{array}$$

Factory Example: Alternate View

Original Problem

Recall we have,

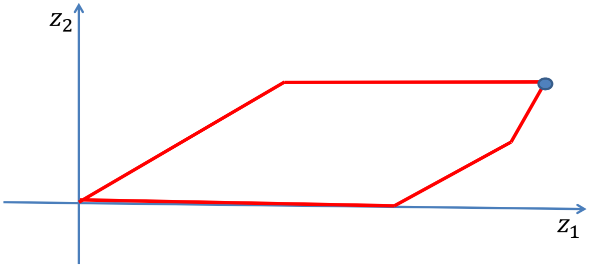
$$\begin{array}{ll} \text{maximize} & x_1 + 6x_2 \\ \text{subject to} & x_1 \le 200 \quad x_2 \le 300 \quad x_1 + x_2 \le 400 \\ & x_1, x_2 \ge 0 \end{array}$$

Transformation

Consider new variable z_1 and z_2, such that $z_1 = x_1 + 6x_2$ and $z_2 = x_2$. Then $x_1 = z_1 - 6z_2$. In terms of the new variables we have

$$\begin{array}{ll} \text{maximize} & z_1 \\ \text{subject to} & z_1 - 6z_2 \le 200 \quad z_2 \le 300 \quad z_1 - 5z_2 \le 400 \\ & z_1 - 6z_2 \ge 0 \qquad z_2 \ge 0 \end{array}$$

Transformed Picture



Feasible region rotated, and optimal value at the right-most point on polygon

Observations about the Transformation

Observations

1. Linear program can always be transformed to get a linear program where the optimal value is achieved at the point in the feasible region with highest $\mathbf{x}$-coordinate
2. Optimum value attained at a vertex of the polygon
3. Since feasible region is convex, and objective function linear, every local optimum is a global optimum

A Simple Algorithm in 2-d

1. optimum solution is at a vertex of the feasible region
2. a vertex is defined by the intersection of two lines (constraints)

A Simple Algorithm in 2-d

1. optimum solution is at a vertex of the feasible region
2. a vertex is defined by the intersection of two lines (constraints)

Algorithm:

1. find all intersections between the $\mathbf{n}$ lines — $\mathbf{n^2}$ points
2. for each intersection point $\mathbf{p = (p_1, p_2)}$
    1. check if $\mathbf{p}$ is in feasible region (how?)
    2. if $\mathbf{p}$ is feasible evaluate objective function at $\mathbf{p}$:
       $\mathbf{val(p) = c_1 p_1 + c_2 p_2}$
3. Output the feasible point with the largest value

A Simple Algorithm in 2-d

1. optimum solution is at a vertex of the feasible region
2. a vertex is defined by the intersection of two lines (constraints)

Algorithm:

1. find all intersections between the $\mathbf{n}$ lines — $\mathbf{n^2}$ points
2. for each intersection point $\mathbf{p = (p_1, p_2)}$
    1. check if $\mathbf{p}$ is in feasible region (how?)
    2. if $\mathbf{p}$ is feasible evaluate objective function at $\mathbf{p}$: $\mathbf{val(p) = c_1p_1 + c_2p_2}$
3. Output the feasible point with the largest value

Running time: $\mathbf{O(n^3)}$.

Simple Algorithm in $\mathbf{d}$ Dimensions

Real problem: $\mathbf{d}$-dimensions

Simple Algorithm in $\mathbf{d}$ Dimensions

Real problem: $\mathbf{d}$-dimensions

1. optimum solution is at a vertex of the feasible region
2. a vertex is defined by the intersection of $\mathbf{d}$ hyperplanes
3. number of vertices can be $\mathbf{\Omega(n^d)}$

Running time: $\mathbf{O(n^{d+1})}$ which is not polynomial since problem size is at least $\mathbf{nd}$. Also not practical.

How do we find the intersection point of $\mathbf{d}$ hyperplanes in $\mathbb{R}^{\mathbf{d}}$?

Simple Algorithm in $\mathbf{d}$ Dimensions

Real problem: $\mathbf{d}$-dimensions

1. optimum solution is at a vertex of the feasible region
2. a vertex is defined by the intersection of $\mathbf{d}$ hyperplanes
3. number of vertices can be $\Omega(\mathbf{n}^{\mathbf{d}})$

Running time: $O(\mathbf{n}^{\mathbf{d}+1})$ which is not polynomial since problem size is at least $\mathbf{nd}$. Also not practical.

How do we find the intersection point of $\mathbf{d}$ hyperplanes in $\mathbb{R}^{\mathbf{d}}$? Using Gaussian elimination to solve $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is a $\mathbf{d} \times \mathbf{d}$ matrix and $\mathbf{b}$ is a $\mathbf{d} \times 1$ matrix.

Linear Programming in **d**-dimensions

1. Each linear constraint defines a **halfspace**.
2. Feasible region, which is an intersection of halfspaces, is a convex **polyhedron**.
3. Every local optimum is a global optimum.
4. Optimal value attained at a vertex of the polyhedron.

Simplex Algorithm

Simplex: Vertex hoping algorithm

Simplex Algorithm

Simplex: Vertex hoping algorithm

Moves from a vertex to its neighboring vertex

Simplex Algorithm

Simplex: Vertex hoping algorithm

Moves from a vertex to its neighboring vertex

Questions

- Which neighbor to move to?
- When to stop?
- How much time does it take?

Observations

For Simplex

Suppose we are at a non-optimal vertex $\hat{\mathbf{x}} = (\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_d)$ and optimal is $\mathbf{x}^* = (\mathbf{x}_1^*, \ldots, \mathbf{x}_d^*)$, then $\mathbf{c} \cdot \mathbf{x}^* > \mathbf{c} \cdot \hat{\mathbf{x}}$.

Observations

For Simplex

Suppose we are at a non-optimal vertex $\hat{\mathbf{x}} = (\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_d)$ and optimal is $\mathbf{x}^* = (\mathbf{x}_1^*, \ldots, \mathbf{x}_d^*)$, then $\mathbf{c} \cdot \mathbf{x}^* > \mathbf{c} \cdot \hat{\mathbf{x}}$.

How does $(\mathbf{c} \cdot \mathbf{x})$ change as we move from $\hat{\mathbf{x}}$ to $\mathbf{x}^*$ on the line joining the two?

Observations

For Simplex

Suppose we are at a non-optimal vertex $\hat{\mathbf{x}} = (\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_d)$ and optimal is $\mathbf{x}^* = (\mathbf{x}_1^*, \ldots, \mathbf{x}_d^*)$, then $\mathbf{c} \cdot \mathbf{x}^* > \mathbf{c} \cdot \hat{\mathbf{x}}$.

How does $(\mathbf{c} \cdot \mathbf{x})$ change as we move from $\hat{\mathbf{x}}$ to $\mathbf{x}^*$ on the line joining the two?

Strictly increases!

Observations

For Simplex

Suppose we are at a non-optimal vertex $\hat{\mathbf{x}} = (\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_d)$ and optimal is $\mathbf{x}^* = (\mathbf{x}_1^*, \ldots, \mathbf{x}_d^*)$, then $\mathbf{c} \cdot \mathbf{x}^* > \mathbf{c} \cdot \hat{\mathbf{x}}$.

How does $(\mathbf{c} \cdot \mathbf{x})$ change as we move from $\hat{\mathbf{x}}$ to $\mathbf{x}^*$ on the line joining the two?

Strictly increases!

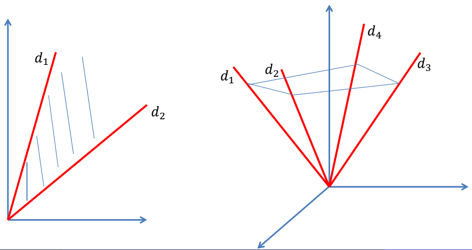
- $\mathbf{d} = \mathbf{x}^* - \hat{\mathbf{x}}$ is the direction from $\hat{\mathbf{x}}$ to $\mathbf{x}^*$.
- $(\mathbf{c} \cdot \mathbf{d}) = (\mathbf{c} \cdot \mathbf{x}^*) - (\mathbf{c} \cdot \hat{\mathbf{x}}) > \mathbf{0}$.
- In $\mathbf{x} = \hat{\mathbf{x}} + \delta \mathbf{d}$, as δ goes from $\mathbf{0}$ to $\mathbf{1}$, we move from $\hat{\mathbf{x}}$ to $\mathbf{x}^*$.
- $\mathbf{c} \cdot \mathbf{x} = \mathbf{c} \cdot \hat{\mathbf{x}} + \delta(\mathbf{c} \cdot \mathbf{d})$. Strictly increasing with δ!
- Due to convexity, all of these are feasible points.

Cone

Definition

Given a set of vectors $\mathbf{D} = \{\mathbf{d_1}, \ldots, \mathbf{d_k}\}$, the cone spanned by them is just their positive linear combinations, i.e.,

$$\mathbf{cone(D)} = \{\mathbf{d} \mid \mathbf{d} = \sum_{\mathbf{i=1}}^{\mathbf{k}} \boldsymbol{\lambda}_\mathbf{i}\mathbf{d}_\mathbf{i}, \text{ where } \boldsymbol{\lambda}_\mathbf{i} \geq \mathbf{0}, \forall \mathbf{i}\}$$



Cone (Contd.)

Lemma

If $\mathbf{d} \in \mathbf{cone(D)}$ *and* $(\mathbf{c} \cdot \mathbf{d}) > \mathbf{0}$, *then there exists* $\mathbf{d_i}$ *such that* $(\mathbf{c} \cdot \mathbf{d_i}) > \mathbf{0}$.

Proof.

To the contrary suppose $(\mathbf{c} \cdot \mathbf{d_i}) \leq \mathbf{0}, \ \forall \mathbf{i} \leq \mathbf{k}$.
Since $\mathbf{d}$ is a positive linear combination of $\mathbf{d_i}$'s,

$$
\begin{aligned}
(\mathbf{c} \cdot \mathbf{d}) &= (\mathbf{c} \cdot \textstyle\sum_{i=1}^{k} \lambda_i \mathbf{d_i}) \\
&= \textstyle\sum_{i=1}^{k} \lambda_i (\mathbf{c} \cdot \mathbf{d_i}) \\
&\leq \mathbf{0}
\end{aligned}
$$

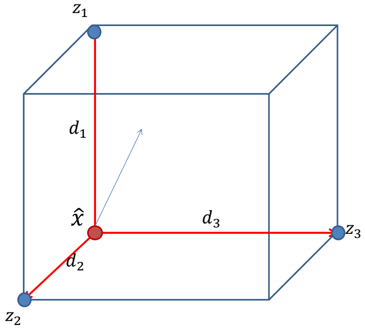
A contradiction! □

Improving Direction Implies Improving Neighbor

Let $\mathbf{z_1}, \ldots, \mathbf{z_k}$ be the neighboring vertices of $\hat{\mathbf{x}}$. And let $\mathbf{d_i} = \mathbf{z_i} - \hat{\mathbf{x}}$ be the direction from $\hat{\mathbf{x}}$ to $\mathbf{z_i}$.

Lemma

Any feasible direction of movement $\mathbf{d}$ *from* $\hat{\mathbf{x}}$ *is in the* $\mathbf{cone(\{d_1, \ldots, d_k\})}$.



Observations

For Simplex

Suppose we are at a non-optimal vertex $\hat{\mathbf{x}} = (\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_d)$ and optimal is $\mathbf{x}^* = (\mathbf{x}_1^*, \ldots, \mathbf{x}_d^*)$, then $\mathbf{c} \cdot \mathbf{x}^* > \mathbf{c} \cdot \hat{\mathbf{x}}$.

- $\mathbf{d} = \mathbf{x}^* - \hat{\mathbf{x}}$ is the direction from $\hat{\mathbf{x}}$ to $\mathbf{x}^*$.
- $(\mathbf{c} \cdot \mathbf{d}) = (\mathbf{c} \cdot \mathbf{x}^*) - (\mathbf{c} \cdot \hat{\mathbf{x}}) > \mathbf{0}$.

Observations

For Simplex

Suppose we are at a non-optimal vertex $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_d)$ and optimal is $x^* = (x_1^*, \ldots, x_d^*)$, then $c \cdot x^* > c \cdot \hat{x}$.

- $d = x^* - \hat{x}$ is the direction from $\hat{x}$ to x^*.
- $(c \cdot d) = (c \cdot x^*) - (c \cdot \hat{x}) > 0$.
- Let d_i be the direction towards neighbor z_i.
- $d \in \text{Cone}(\{d_1, \ldots, d_k\}) \Rightarrow \exists d_i,\ (c \cdot d_i) > 0$.

Observations

For Simplex

Suppose we are at a non-optimal vertex $\hat{\mathbf{x}} = (\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_d)$ and optimal is $\mathbf{x}^* = (\mathbf{x}_1^*, \ldots, \mathbf{x}_d^*)$, then $\mathbf{c} \cdot \mathbf{x}^* > \mathbf{c} \cdot \hat{\mathbf{x}}$.

- $\mathbf{d} = \mathbf{x}^* - \hat{\mathbf{x}}$ is the direction from $\hat{\mathbf{x}}$ to $\mathbf{x}^*$.
- $(\mathbf{c} \cdot \mathbf{d}) = (\mathbf{c} \cdot \mathbf{x}^*) - (\mathbf{c} \cdot \hat{\mathbf{x}}) > \mathbf{0}$.
- Let $\mathbf{d}_i$ be the direction towards neighbor $\mathbf{z}_i$.
- $\mathbf{d} \in \mathbf{Cone}(\{\mathbf{d}_1, \ldots, \mathbf{d}_k\}) \Rightarrow \exists \mathbf{d}_i,\ (\mathbf{c} \cdot \mathbf{d}_i) > \mathbf{0}$.

Theorem

If vertex $\hat{\mathbf{x}}$ is not optimal then it has a neighbor where cost improves.

How Many Neighbors a Vertex Has?

Geometric view...

$\mathbf{A} \in \mathbf{R}^{\mathbf{n} \times \mathbf{d}}$ ($\mathbf{n} > \mathbf{d}$), $\mathbf{b} \in \mathbf{R}^{\mathbf{n}}$, the constraints are: $\mathbf{A}\mathbf{x} \leq \mathbf{b}$

Faces

- $\mathbf{n}$ constraints/inequalities. Each defines a hyperplane.
- Vertex: 0-dimensional face. Edge: 1D face. $\ldots$ Hyperplane: $(\mathbf{d} - \mathbf{1})$D face.

How Many Neighbors a Vertex Has?

Geometric view...

$\mathbf{A} \in \mathbf{R}^{\mathbf{n} \times \mathbf{d}}$ ($\mathbf{n} > \mathbf{d}$), $\mathbf{b} \in \mathbf{R}^{\mathbf{n}}$, the constraints are: $\mathbf{Ax} \leq \mathbf{b}$

Faces

- $\mathbf{n}$ constraints/inequalities. Each defines a hyperplane.
- Vertex: 0-dimensional face. Edge: 1D face. $\ldots$ Hyperplane: $(\mathbf{d} - \mathbf{1})$D face.
- $\mathbf{r}$ linearly independent hyperplanes forms $\mathbf{d} - \mathbf{r}$ dimensional face.

How Many Neighbors a Vertex Has?

Geometric view...

$\mathbf{A} \in \mathbf{R}^{\mathbf{n}\times\mathbf{d}}$ ($\mathbf{n} > \mathbf{d}$), $\mathbf{b} \in \mathbf{R}^{\mathbf{n}}$, the constraints are: $\mathbf{Ax} \leq \mathbf{b}$

Faces

- $\mathbf{n}$ constraints/inequalities. Each defines a hyperplane.
- Vertex: 0-dimensional face. Edge: 1D face. $\ldots$ Hyperplane: $(\mathbf{d}-\mathbf{1})$D face.
- $\mathbf{r}$ linearly independent hyperplanes forms $\mathbf{d}-\mathbf{r}$ dimensional face.
- Vertices being of $\mathbf{0}$D, $\mathbf{d}$ L.I. hyperplanes form a vertex.

How Many Neighbors a Vertex Has?

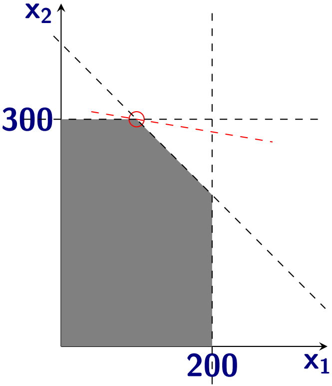
Geometric view...

$\mathbf{A} \in \mathbf{R}^{\mathbf{n}\times\mathbf{d}}$ ($\mathbf{n} > \mathbf{d}$), $\mathbf{b} \in \mathbf{R}^{\mathbf{n}}$, the constraints are: $\mathbf{Ax} \leq \mathbf{b}$

Faces

- $\mathbf{n}$ constraints/inequalities. Each defines a hyperplane.
- Vertex: 0-dimensional face. Edge: 1D face. $\ldots$ Hyperplane: $(\mathbf{d}-\mathbf{1})$D face.
- $\mathbf{r}$ linearly independent hyperplanes forms $\mathbf{d}-\mathbf{r}$ dimensional face.
- Vertices being of $\mathbf{0}$D, $\mathbf{d}$ L.I. hyperplanes form a vertex.

In 2-dimension ($\mathbf{d} = \mathbf{2}$)



How Many Neighbors a Vertex Has?

Geometric view...

$\mathbf{A} \in \mathbf{R}^{\mathbf{n} \times \mathbf{d}}$ $(\mathbf{n} > \mathbf{d})$, $\mathbf{b} \in \mathbf{R}^{\mathbf{n}}$, the constraints are: $\mathbf{Ax} \leq \mathbf{b}$

Faces

- $\mathbf{n}$ constraints/inequalities. Each defines a hyperplane.
- Vertex: 0-dimensional face. Edge: 1D face. $\ldots$ Hyperplane: $(\mathbf{d} - \mathbf{1})$D face.
- $\mathbf{r}$ linearly independent hyperplanes forms $\mathbf{d} - \mathbf{r}$ dimensional face.
- Vertices being of $\mathbf{0}$D, $\mathbf{d}$ L.I. hyperplanes form a vertex.

In 3-dimension $(\mathbf{d} = \mathbf{3})$

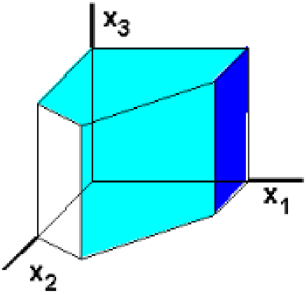


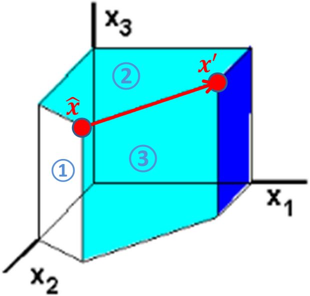
image source: webpage of Prof. Forbes W. Lewis

How Many Neighbors a Vertex Has?

Geometry view...

One neighbor per tight hyperplane. Therefore typically $\mathbf{d}$.

- Suppose $\mathbf{x'}$ is a neighbor of $\hat{\mathbf{x}}$, then on the edge joining the two $\mathbf{d-1}$ hyperplanes are tight.
- These $\mathbf{d-1}$ are also tight at both $\hat{\mathbf{x}}$ and $\mathbf{x'}$.
- In addition one more hyperplane, say $\mathbf{(Ax)_i = b_i}$, is tight at $\hat{\mathbf{x}}$. "Relaxing" this at $\hat{\mathbf{x}}$ leads to $\mathbf{x'}$.



Simplex Algorithm

Simplex: Vertex hoping algorithm

Moves from a vertex to its neighboring vertex

Questions + Answers

- Which neighbor to move to? One where objective value increases.

Simplex Algorithm

Simplex: Vertex hoping algorithm

Moves from a vertex to its neighboring vertex

Questions + Answers

- Which neighbor to move to? One where objective value increases.
- When to stop? When no neighbor with better objective value.

Simplex Algorithm

Simplex: Vertex hoping algorithm

Moves from a vertex to its neighboring vertex

Questions + Answers

- Which neighbor to move to? One where objective value increases.
- When to stop? When no neighbor with better objective value.
- How much time does it take? At most **d** neighbors to consider in each step.

Simplex in 2-d

Simplex Algorithm

1. Start from some vertex of the feasible polygon.
2. Compare value of objective function at current vertex with the value at **2** "neighboring" vertices of polygon.
3. If neighboring vertex improves objective function, move to this vertex, and repeat step 2.
4. If no improving neighbor (local optimum), then stop.

Simplex in Higher Dimensions

Simplex Algorithm

1. Start at a vertex of the polytope.
2. Compare value of objective function at each of the **d** "neighbors".
3. Move to neighbor that improves objective function, and repeat step 2.
4. If no improving neighbor, then stop.

Simplex in Higher Dimensions

Simplex Algorithm

1. Start at a vertex of the polytope.
2. Compare value of objective function at each of the **d** "neighbors".
3. Move to neighbor that improves objective function, and repeat step 2.
4. If no improving neighbor, then stop.

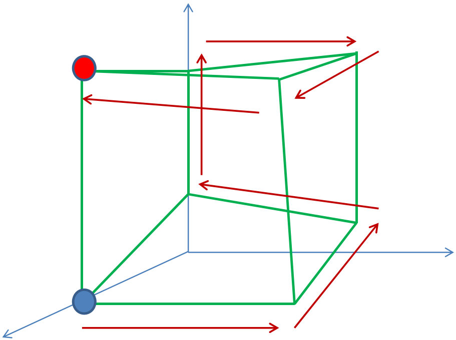
Simplex is a greedy local-improvement algorithm! Works because a local optimum is also a global optimum — convexity of polyhedra.

Solving Linear Programming in Practice

1. Naïve implementation of Simplex algorithm can be very inefficient

Solving Linear Programming in Practice

1. Naïve implementation of Simplex algorithm can be very inefficient – Exponential number of steps!



Solving Linear Programming in Practice

1. Naïve implementation of Simplex algorithm can be very inefficient
   1. Choosing which neighbor to move to can significantly affect running time
   2. Very efficient Simplex-based algorithms exist
   3. Simplex algorithm takes exponential time in the worst case but works extremely well in practice with many improvements over the years
2. Non Simplex based methods like interior point methods work well for large problems.

Polynomial time Algorithm for Linear Programming

Major open problem for many years: is there a polynomial time algorithm for linear programming?

Polynomial time Algorithm for Linear Programming

Major open problem for many years: is there a polynomial time algorithm for linear programming?

Leonid Khachiyan in 1979 gave the first polynomial time algorithm using the Ellipsoid method.

1. major theoretical advance
2. highly impractical algorithm, not used at all in practice
3. routinely used in theoretical proofs.

Polynomial time Algorithm for Linear Programming

Major open problem for many years: is there a polynomial time algorithm for linear programming?

Leonid Khachiyan in 1979 gave the first polynomial time algorithm using the Ellipsoid method.

1. major theoretical advance
2. highly impractical algorithm, not used at all in practice
3. routinely used in theoretical proofs.

Narendra Karmarkar in 1984 developed another polynomial time algorithm, the interior point method.

1. very practical for some large problems and beats simplex
2. also revolutionized theory of interior point methods

Polynomial time Algorithm for Linear Programming

Major open problem for many years: is there a polynomial time algorithm for linear programming?

Leonid Khachiyan in 1979 gave the first polynomial time algorithm using the Ellipsoid method.

1. major theoretical advance
2. highly impractical algorithm, not used at all in practice
3. routinely used in theoretical proofs.

Narendra Karmarkar in 1984 developed another polynomial time algorithm, the interior point method.

1. very practical for some large problems and beats simplex
2. also revolutionized theory of interior point methods

Following interior point method success, Simplex has been improved enormously and is the method of choice.

Degeneracy

1. The linear program could be infeasible: No points satisfy the constraints.
2. The linear program could be unbounded: Polygon unbounded in the direction of the objective function.
3. More than **d** hyperplanes could be tight at a vertex, forming more than **d** neighbors.

Infeasibility: Example

$$\begin{array}{ll} \text{maximize} & x_1 + 6x_2 \\ \text{subject to} & x_1 \leq 2 \quad x_2 \leq 1 \quad x_1 + x_2 \geq 4 \\ & x_1, x_2 \geq 0 \end{array}$$

Infeasibility has to do only with constraints.

Infeasibility: Example

$$\begin{array}{ll} \text{maximize} & x_1 + 6x_2 \\ \text{subject to} & x_1 \leq 2 \quad x_2 \leq 1 \quad x_1 + x_2 \geq 4 \\ & x_1, x_2 \geq 0 \end{array}$$

Infeasibility has to do only with constraints.

No starting vertex for Simplex.

Infeasibility: Example

$$\begin{array}{ll} \text{maximize} & x_1 + 6x_2 \\ \text{subject to} & x_1 \leq 2 \quad x_2 \leq 1 \quad x_1 + x_2 \geq 4 \\ & x_1, x_2 \geq 0 \end{array}$$

Infeasibility has to do only with constraints.

No starting vertex for Simplex. How to detect this?

Unboundedness: Example

$$\begin{aligned} \text{maximize} \quad & x_2 \\ x_1 + x_2 & \geq 2 \\ x_1, x_2 & \geq 0 \end{aligned}$$

Unboundedness depends on both constraints and the objective function.

Unboundedness: Example

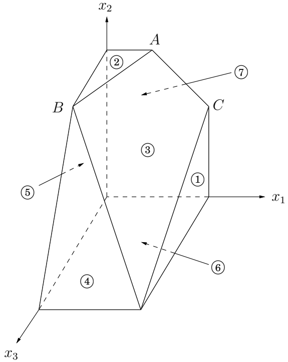
$$\begin{aligned} \text{maximize} \quad & \mathbf{x_2} \\ \mathbf{x_1 + x_2} &\geq \mathbf{2} \\ \mathbf{x_1, x_2} &\geq \mathbf{0} \end{aligned}$$

Unboundedness depends on both constraints and the objective function.

If unbounded in the direction of objective function, then Simplex detects it.

Degeneracy and Cycling

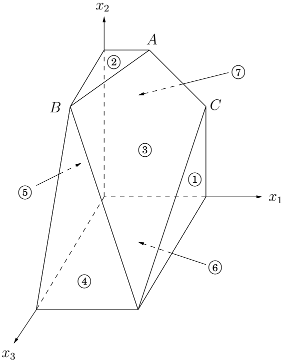
More than **d** inequalities tight at a vertex.



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

Degeneracy and Cycling

More than **d** inequalities tight at a vertex.



$$
\begin{aligned}
\max \quad & x_1 + 6x_2 + 13x_3 & \\
& x_1 \le 200 & (1)\\
& x_2 \le 300 & (2)\\
& x_1 + x_2 + x_3 \le 400 & (3)\\
& x_2 + 3x_3 \le 600 & (4)\\
& x_1 \ge 0 & (5)\\
& x_2 \ge 0 & (6)\\
& x_3 \ge 0 & (7)
\end{aligned}
$$

Depending on how Simplex is implemented, it may cycle at this vertex.

We will see how in the next lecture.