Introduction to Linear Programming

Lecture 25

December 4, 2014

Easy or not easy?

Let $x_1, \ldots, x_n \in \{0, 1\}$ be boolean variables. You are given m constraints of the form:

$$1 + x_i + x_j - x_k \geq 1.$$

That is, each variable might have $+1$ or -1 as a coefficient, and each inequality has three variables, and a constant additive term. Deciding if such a problem has a feasible solution is

(A) NP-Complete.

(B) NP-Hard.

(C) P.

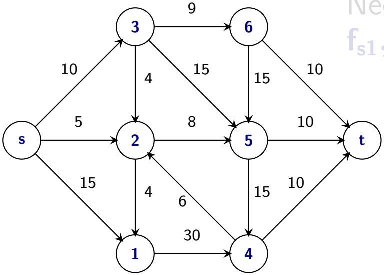
(D) Not a well defined question.

(E) Doable in polynomial time if Riemann's hypothesis is true.

Part I

Introduction to Linear Programming

Maximum Flow in Network



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

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

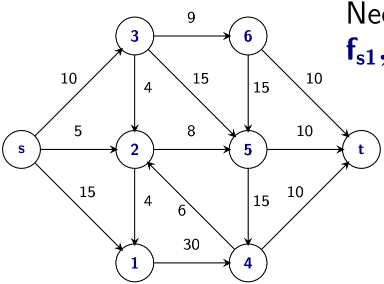
and

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

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

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

Maximum Flow in Network



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

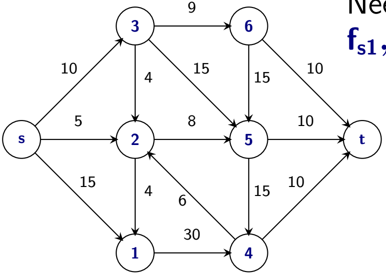
$$f_{s1} \le 15 \quad f_{s2} \le 5 \quad f_{s3} \le 10$$
$$f_{14} \le 30 \quad f_{21} \le 4 \quad f_{25} \le 8$$
$$f_{32} \le 4 \quad f_{35} \le 15 \quad f_{36} \le 9$$
$$f_{42} \le 6 \quad f_{4t} \le 10 \quad f_{54} \le 15$$
$$f_{5t} \le 10 \quad f_{65} \le 15 \quad f_{6t} \le 10$$

and

$$f_{s1} + f_{21} = f_{14} \quad f_{s2} + f_{32} = f_{21} + f_{25} \quad f_{s3} = f_{32} + f_{35} + f_{36}$$
$$f_{14} + f_{54} = f_{42} + f_{4t} \quad f_{25} + f_{35} + f_{65} = f_{54} + f_{5t} \quad f_{36} = f_{65} + f_{6t}$$
$$f_{s1} \ge 0 \quad f_{s2} \ge 0 \quad f_{s3} \ge 0 \quad \cdots \quad f_{4t} \ge 0 \quad f_{5t} \ge 0 \quad f_{6t} \ge 0$$

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

Maximum Flow in Network



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

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

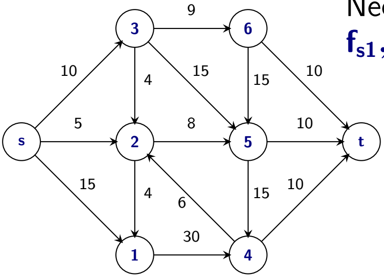
and

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

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

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

Maximum Flow in Network



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

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

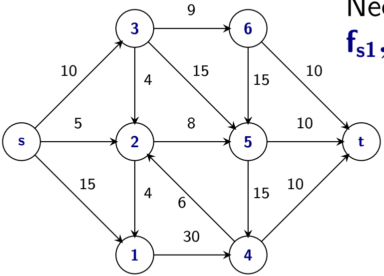
and

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

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

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

Maximum Flow in Network



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

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

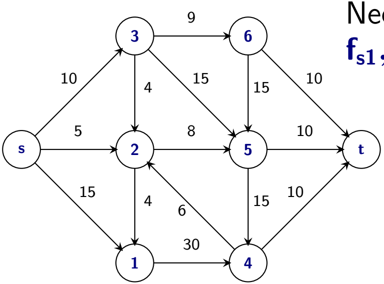
and

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

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

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

Maximum Flow 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 $\mathbf{e \in E}$, we have variables $\mathbf{f_e}$ indicating flow on edge $\mathbf{e}$

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

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

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

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 $\mathbf{e}$

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

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 $\mathbf{v}$.

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

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

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

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

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

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

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

Minimum Cost Flow with Lower Bounds

... as a Linear Program

For a general flow network $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ 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 $\mathbf{v}$.

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

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

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

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

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

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

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

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 $\mathbf{c} = (c_j) \in \mathbb{R}^{d}$

Linear Programs

Problem

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

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

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

Canonical Form of Linear Programs

Canonical Form

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

$$\begin{array}{lll} \text{maximize} & \sum_{j=1}^{d} c_j x_j & \\ \text{subject to} & \sum_{j=1}^{d} a_{ij} x_j \leq b_i & \text{for } i = 1 \ldots n \\ & x_j \geq 0 & \text{for } j = 1 \ldots d \end{array}$$

Conversion to Canonical Form

1. Replace each variable x_j by $x_j^+ - x_j^-$ and inequalities $x_j^+ \geq 0$ and $x_j^- \geq 0$
2. Replace $\sum_j a_{ij} x_j = b_i$ by $\sum_j a_{ij} x_j \leq b_i$ and $-\sum_j a_{ij} x_j \leq -b_i$
3. Replace $\sum_j a_{ij} x_j \geq b_i$ by $-\sum_j a_{ij} x_j \leq -b_i$

Canonical Form of Linear Programs

Canonical Form

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

$$\begin{array}{lll} \text{maximize} & \sum_{j=1}^{d} c_j x_j & \\ \text{subject to} & \sum_{j=1}^{d} a_{ij} x_j \leq b_i & \text{for } i = 1 \ldots n \\ & x_j \geq 0 & \text{for } j = 1 \ldots d \end{array}$$

Conversion to Canonical Form

1. Replace each variable x_j by $x_j^+ - x_j^-$ and inequalities $x_j^+ \geq 0$ and $x_j^- \geq 0$
2. Replace $\sum_j a_{ij} x_j = b_i$ by $\sum_j a_{ij} x_j \leq b_i$ and $-\sum_j a_{ij} x_j \leq -b_i$
3. Replace $\sum_j a_{ij} x_j \geq b_i$ by $-\sum_j a_{ij} x_j \leq -b_i$

Matrix Representation of Linear Programs

A linear program in canonical form can be written as

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

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

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

Other Standard Forms for Linear Programs

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

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

Linear Programming: A History

1. First formalized applied 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"

Shortest path as a LP

Let G be a directed graph with weights on the edges, and a vertices $\mathbf{s}$ and $\mathbf{t}$. For a vertex $\mathbf{v} \in V(G)$, let $\mathbf{x_v}$ be the length of the shortest path from $\mathbf{s}$ to $\mathbf{v}$. For all $(\mathbf{u} \rightarrow \mathbf{v}) \in E(G)$, we must have that

(A) $\mathbf{x_u + w(u \rightarrow v) \leq x_v}$.

(B) $\mathbf{x_u + x_v - w(u \rightarrow v) \geq 0}$.

(C) $\mathbf{x_u + w(u \rightarrow v) \geq x_v}$.

(D) $\mathbf{x_u + x_v + w(u \rightarrow v) \geq 0}$.

(E) All of the above.

Computing shortest path from s to t is the LP...

(A)
$$\begin{array}{rl} \max & x_t \\ \forall (u \to v) \in E & x_u + w(u \to v) \geq x_v \\ & x_s = 0. \end{array}$$

(B)
$$\begin{array}{rl} \min & x_s \\ \forall (u \to v) \in E & x_u + w(u \to v) \geq x_v \\ & x_t = 0. \end{array}$$

(C)
$$\begin{array}{rl} \min & x_t \\ \forall (u \to v) \in E & x_u + w(u \to v) \geq x_v \\ & x_s = 0. \end{array}$$

(D)
$$\begin{array}{rl} \max & x_t \\ \forall (u \to v) \in E & x_u + w(u \to v) \geq x_v \\ & x_s = 0. \end{array}$$

A Factory Example

Problem

Suppose a factory produces two products **I** and **II**. Each requires three resources **A, B, C**.

1. Producing one unit of Product I requires 1 unit each of resources **A** and **C**.
2. One unit of Product II requires 1 unit of resource **B** and 1 units of resource **C**.
3. We have 200 units of **A**, 300 units of **B**, and 400 units of **C**.
4. Product I can be sold for **$1** and product II for **$6**.

How many units of product I and product II should the factory manufacture to maximize profit?

Solution: Formulate as a linear program.

A Factory Example

Problem

Suppose a factory produces two products **I** and **II**. Each requires three resources **A, B, C**.

1. Producing one unit of Product I requires 1 unit each of resources **A** and **C**.
2. One unit of Product II requires 1 unit of resource **B** and 1 units of resource **C**.
3. We have 200 units of **A**, 300 units of **B**, and 400 units of **C**.
4. Product I can be sold for **$1** and product II for **$6**.

How many units of product I and product II should the factory manufacture to maximize profit?

Solution: Formulate as a linear program.

A Factory Example

Problem

Suppose a factory produces two products **I** and **II**. Each requires three resources **A, B, C**.

1. Producing one unit of Product I requires 1 unit each of resources **A** and **C**.
2. One unit of Product II requires 1 unit of resource **B** and 1 units of resource **C**.
3. We have 200 units of **A**, 300 units of **B**, and 400 units of **C**.
4. Product I can be sold for **$1** and product II for **$6**.

How many units of product I and product II should the factory manufacture to maximize profit?

Solution: Formulate as a linear program.

A Factory Example

Problem

Suppose a factory produces two products **I** and **II**. Each requires three resources **A, B, C**.

1. Producing unit I: Req. 1 unit of **A**, **C**.
2. Producing unit II: Requ. 1 unit of **B, C**.
3. Have **A**: 200, **B**: 300 , and **C**: 400.
4. Price I: **$1**, and II: **$6**.

How many units of I and II to manufacture to max profit?

A Factory Example

Problem

Suppose a factory produces two products **I** and **II**. Each requires three resources **A, B, C**.

1. Producing unit I: Req. 1 unit of **A**, **C**.
2. Producing unit II: Requ. 1 unit of **B, C**.
3. Have **A**: 200, **B**: 300 , and **C**: 400.
4. Price I: **$1**, and II: **$6**.

How many units of I and II to manufacture to max profit?

$$\begin{aligned} \textbf{max} \quad & \mathbf{x_I + 6x_{II}} & \\ \text{s.t.} \quad & \mathbf{x_I \leq 200} & \textbf{(A)} \\ & \mathbf{x_{II} \leq 300} & \textbf{(B)} \\ & \mathbf{x_I + x_{II} \leq 400} & \textbf{(C)} \\ & \mathbf{x_I \geq 0} & \\ & \mathbf{x_{II} \geq 0} & \end{aligned}$$

Linear Programming Formulation

Let us produce x_1 units of product I and x_2 units of product II. Our profit can be computed by solving

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

What is the solution?

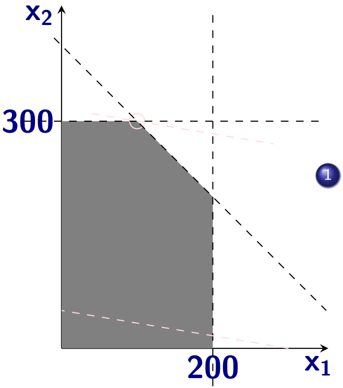
Linear Programming Formulation

Let us produce x_1 units of product I and x_2 units of product II. Our profit can be computed by solving

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

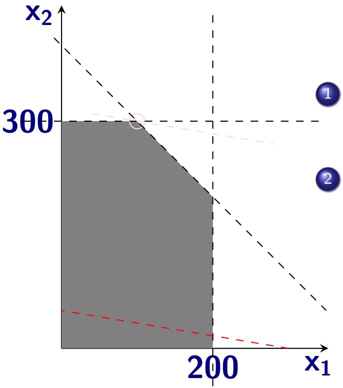
What is the solution?

Solving the Factory Example



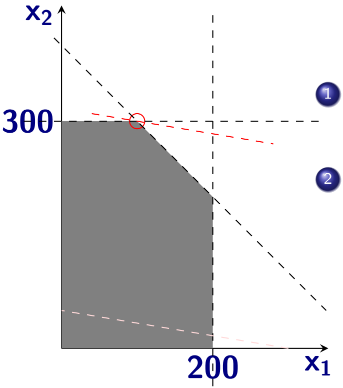
1. Feasible values of $\mathbf{x_1}$ and $\mathbf{x_2}$ are shaded region.

Solving the Factory Example



1. Feasible values of $\mathbf{x_1}$ and $\mathbf{x_2}$ are shaded region.
2. Objective 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.

Solving the Factory Example

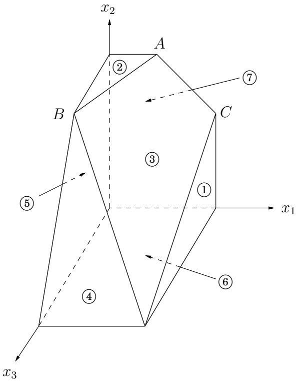


1. Feasible values of $\mathbf{x_1}$ and $\mathbf{x_2}$ are shaded region.
2. Objective function is a direction — the line represents all points with same value of the function; moving the line until it just leaves the feasible region, gives optimal values.

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



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

Figure from Dasgupta etal book.

Factory Example: Alternate View

Original Problem

Recall we have,

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

Transformation

Consider new variable x'_1 and x'_2, such that $x_1 = -6x'_1 + x'_2$ and $x_2 = x'_1 + 6x'_2$. Then in terms of the new variables we have

$$\begin{array}{ll} \text{maximize} & 37x'_2 \\ \text{subject to} & -6x'_1 + x'_2 \leq 200 \quad x'_1 + 6x'_2 \leq 300 \quad -5x'_1 + 7x_2 \leq 40 \\ & -6x'_1 + x'_2 \geq 0 \quad x'_1 + 6x'_2 \geq 0 \end{array}$$

Factory Example: Alternate View

Original Problem

Recall we have,

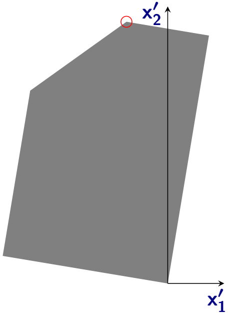
$$\begin{array}{ll} \text{maximize} & x_1 + 6x_2 \\ \text{subject to} & x_1 \leq 200 \quad x_2 \leq 300 \quad x_1 + x_2 \leq 400 \\ & x_1, x_2 \geq 0 \end{array}$$

Transformation

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

$$\begin{array}{ll} \text{maximize} & 37x_2' \\ \text{subject to} & -6x_1' + x_2' \leq 200 \quad x_1' + 6x_2' \leq 300 \quad -5x_1' + 7x_2 \leq 40 \\ & -6x_1' + x_2' \geq 0 \quad x_1' + 6x_2' \geq 0 \end{array}$$

Transformed Picture



Feasible region rotated, and optimal value at the highest 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 $\mathbf{y}$-coordinate
2. Optimum value attained at a vertex of the polygon
3. Since feasible region is convex, 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)

Algorithm:

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

Running time: $O(n^3)$.

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

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

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 General Case

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 $\mathbb{R}^{\mathbf{d}}$? Using Gaussian elimination to solve $\mathbf{Ax = b}$ where $\mathbf{A}$ is a $\mathbf{d \times d}$ matrix and $\mathbf{b}$ is a $\mathbf{d \times 1}$ matrix.

Simple Algorithm in General Case

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 $\mathbb{R}^{\mathbf{d}}$? Using Gaussian elimination to solve $\mathbf{Ax = b}$ where $\mathbf{A}$ is a $\mathbf{d} \times \mathbf{d}$ matrix and $\mathbf{b}$ is a $\mathbf{d} \times \mathbf{1}$ matrix.

Simple Algorithm in General Case

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 $\mathbb{R}^{\mathbf{d}}$? Using Gaussian elimination to solve $\mathbf{Ax = b}$ where $\mathbf{A}$ is a $\mathbf{d} \times \mathbf{d}$ matrix and $\mathbf{b}$ is a $\mathbf{d} \times \mathbf{1}$ matrix.

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 "neighboring" vertices of polygon
3. If neighboring vertex improves objective function, move to this vertex, and repeat step 2
4. If current vertex is local optimum, then stop.

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. Optimal value attained at a vertex of the polyhedron.
4. Every local optimum is a global optimum.

Simplex in Higher Dimensions

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 local optimum, then stop

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

Simplex in Higher Dimensions

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 local optimum, 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
    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?

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.

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.

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.

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.

Infeasibility: Example

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

Infeasibility has to do only with constraints.

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.

Feasible Solutions and Lower Bounds

Consider the program

$$\begin{array}{lrrrl} \text{maximize} & 4x_1 + & x_2 + & 3x_3 & \\ \text{subject to} & x_1 + & 4x_2 & & \leq 1 \\ & 3x_1 - & x_2 + & x_3 & \leq 3 \\ & & & x_1, x_2, x_3 & \geq 0 \end{array}$$

1. $(1, 0, 0)$ satisfies all the constraints and gives value 4 for the objective function.
2. Thus, optimal value σ^* is at least 4.
3. $(0, 0, 3)$ also feasible, and gives a better bound of 9.
4. How good is 9 when compared with σ^*?

Feasible Solutions and Lower Bounds

Consider the program

$$\begin{array}{llllll} \text{maximize} & 4x_1 + & x_2 + & 3x_3 & & \\ \text{subject to} & x_1 + & 4x_2 & & \leq & 1 \\ & 3x_1 - & x_2 + & x_3 & \leq & 3 \\ & & & x_1, x_2, x_3 & \geq & 0 \end{array}$$

1. $(1, 0, 0)$ satisfies all the constraints and gives value 4 for the objective function.
2. Thus, optimal value σ^* is at least 4.
3. $(0, 0, 3)$ also feasible, and gives a better bound of 9.
4. How good is 9 when compared with σ^*?

Feasible Solutions and Lower Bounds

Consider the program

$$\begin{array}{lrrrl} \text{maximize} & 4x_1 + & x_2 + & 3x_3 & \\ \text{subject to} & x_1 + & 4x_2 & & \leq 1 \\ & 3x_1 - & x_2 + & x_3 & \leq 3 \\ & & & x_1, x_2, x_3 & \geq 0 \end{array}$$

1. $(1, 0, 0)$ satisfies all the constraints and gives value 4 for the objective function.
2. Thus, optimal value σ^* is at least 4.
3. $(0, 0, 3)$ also feasible, and gives a better bound of 9.
4. How good is 9 when compared with σ^*?

Feasible Solutions and Lower Bounds

Consider the program

$$\begin{array}{lrrrl} \text{maximize} & 4x_1 + & x_2 + & 3x_3 & \\ \text{subject to} & x_1 + & 4x_2 & & \leq 1 \\ & 3x_1 - & x_2 + & x_3 & \leq 3 \\ & & & x_1, x_2, x_3 & \geq 0 \end{array}$$

1. $(1, 0, 0)$ satisfies all the constraints and gives value 4 for the objective function.
2. Thus, optimal value σ^* is at least 4.
3. $(0, 0, 3)$ also feasible, and gives a better bound of 9.
4. How good is 9 when compared with σ^*?

Feasible Solutions and Lower Bounds

Consider the program

$$
\begin{array}{lrrrl}
\text{maximize} & 4x_1 + & x_2 + & 3x_3 & \\
\text{subject to} & x_1 + & 4x_2 & & \leq 1 \\
 & 3x_1 - & x_2 + & x_3 & \leq 3 \\
 & & & x_1, x_2, x_3 & \geq 0
\end{array}
$$

1. $(1, 0, 0)$ satisfies all the constraints and gives value 4 for the objective function.
2. Thus, optimal value σ^* is at least 4.
3. $(0, 0, 3)$ also feasible, and gives a better bound of 9.
4. How good is 9 when compared with σ^*?

Obtaining Upper Bounds

1. Let us multiply the first constraint by $\mathbf{2}$ and the second by $\mathbf{3}$ and add the result

$$\begin{array}{rrrrl} 2(& x_1+ & 4x_2 & &) \leq 2(1) \\ +3(& 3x_1- & x_2+ & x_3 &) \leq 3(3) \\ \hline & 11x_1+ & 5x_2+ & 3x_3 & \leq 11 \end{array}$$

2. Since $\mathbf{x_i}$s are positive, compared to objective function $\mathbf{4x_1 + x_2 + 3x_3}$, we have

$$4x_1 + x_2 + 3x_3 \leq 11x_1 + 5x_2 + 3x_3 \leq 11$$

3. Thus, 11 is an upper bound on the optimum value!

Generalizing . . .

1. Multiply first equation by $\mathbf{y_1}$ and second by $\mathbf{y_2}$ (both $\mathbf{y_1, y_2}$ being positive) and add

$$\begin{array}{rrrrl} \mathbf{y_1}(& \mathbf{x_1}+ & \mathbf{4x_2} & &) \leq \mathbf{y_1(1)} \\ +\mathbf{y_2}(& \mathbf{3x_1}- & \mathbf{x_2}+ & \mathbf{x_3} &) \leq \mathbf{y_2(3)} \\ \hline & \mathbf{(y_1+3y_2)x_1}+ & \mathbf{(4y_1-y_2)x_2}+ & \mathbf{(y_2)x_3} & \leq \mathbf{y_1+3y_2} \end{array}$$

2. $\mathbf{y_1 + 3y_2}$ is an upper bound, provided coefficients of $\mathbf{x_i}$ are as large as in the objective function, i.e.,

$$\mathbf{y_1 + 3y_2 \geq 4 \quad 4y_1 - y_2 \geq 1 \quad y_2 \geq 3}$$

3. The best upper bound is when $\mathbf{y_1 + 3y_2}$ is minimized!

Dual LP: Example

Thus, the optimum value of program

$$\begin{aligned} \text{maximize} \quad & 4x_1 + x_2 + 3x_3 \\ \text{subject to} \quad & x_1 + 4x_2 \le 1 \\ & 3x_1 - x_2 + x_3 \le 3 \\ & x_1, x_2, x_3 \ge 0 \end{aligned}$$

is upper bounded by the optimal value of the program

$$\begin{aligned} \text{minimize} \quad & y_1 + 3y_2 \\ \text{subject to} \quad & y_1 + 3y_2 \ge 4 \\ & 4y_1 - y_2 \ge 1 \\ & y_2 \ge 3 \\ & y_1, y_2 \ge 0 \end{aligned}$$

Dual Linear Program

Given a linear program Π in canonical form

$$\begin{array}{lll} \text{maximize} & \sum_{j=1}^{d} c_j x_j & \\ \text{subject to} & \sum_{j=1}^{d} a_{ij} x_j \leq b_i & i = 1, 2, \ldots n \\ & x_j \geq 0 & j = 1, 2, \ldots d \end{array}$$

the dual $\mathrm{Dual}(\Pi)$ is given by

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{n} b_i y_i & \\ \text{subject to} & \sum_{i=1}^{n} y_i a_{ij} \geq c_j & j = 1, 2, \ldots d \\ & y_i \geq 0 & i = 1, 2, \ldots n \end{array}$$

Proposition

$\mathrm{Dual}(\mathrm{Dual}(\Pi))$ *is equivalent to* Π

Dual Linear Program

Given a linear program Π in canonical form

$$\begin{array}{lll} \text{maximize} & \sum_{j=1}^{d} c_j x_j & \\ \text{subject to} & \sum_{j=1}^{d} a_{ij} x_j \leq b_i & i = 1, 2, \ldots n \\ & x_j \geq 0 & j = 1, 2, \ldots d \end{array}$$

the dual $\mathrm{Dual}(\Pi)$ is given by

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{n} b_i y_i & \\ \text{subject to} & \sum_{i=1}^{n} y_i a_{ij} \geq c_j & j = 1, 2, \ldots d \\ & y_i \geq 0 & i = 1, 2, \ldots n \end{array}$$

Proposition

$\mathrm{Dual}(\mathrm{Dual}(\Pi))$ *is equivalent to* Π

Duality Theorem

Theorem (Weak Duality)

If $\mathbf{x}$ *is a feasible solution to* Π *and* $\mathbf{y}$ *is a feasible solution to* $\mathbf{Dual}(\Pi)$ *then* $\mathbf{c} \cdot \mathbf{x} \leq \mathbf{y} \cdot \mathbf{b}$.

Theorem (Strong Duality)

If $\mathbf{x}^*$ *is an optimal solution to* Π *and* $\mathbf{y}^*$ *is an optimal solution to* $\mathbf{Dual}(\Pi)$ *then* $\mathbf{c} \cdot \mathbf{x}^* = \mathbf{y}^* \cdot \mathbf{b}$.

Many applications! Maxflow-Mincut theorem can be deduced from duality.

Duality Theorem

Theorem (Weak Duality)

If $\mathbf{x}$ *is a feasible solution to* $\mathbf{\Pi}$ *and* $\mathbf{y}$ *is a feasible solution to* $\mathbf{Dual(\Pi)}$ *then* $\mathbf{c} \cdot \mathbf{x} \leq \mathbf{y} \cdot \mathbf{b}$.

Theorem (Strong Duality)

If $\mathbf{x}^*$ *is an optimal solution to* $\mathbf{\Pi}$ *and* $\mathbf{y}^*$ *is an optimal solution to* $\mathbf{Dual(\Pi)}$ *then* $\mathbf{c} \cdot \mathbf{x}^* = \mathbf{y}^* \cdot \mathbf{b}$.

Many applications! Maxflow-Mincut theorem can be deduced from duality.

Maximum Flow Revisited

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

Maximize $\sum_{\mathbf{e} \text{ out of } \mathbf{s}} \mathbf{f_e}$	subject to
$\mathbf{f_e \le c_e}$	for each $\mathbf{e \in E}$
$\sum_{\mathbf{e} \text{ out of } \mathbf{v}} \mathbf{f_e} - \sum_{\mathbf{e} \text{ into } \mathbf{v}} \mathbf{f_e} = \mathbf{0}$	for each $\mathbf{v \in V - \{s, t\}}$
$\mathbf{f_e \ge 0}$	for each $\mathbf{e \in E}$

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

Maximum flow can be reduced to Linear Programming.

Integer Linear Programming

Problem

Find a vector $\mathbf{x} \in \mathbf{Z}^{\mathbf{d}}$ (integer values) that

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

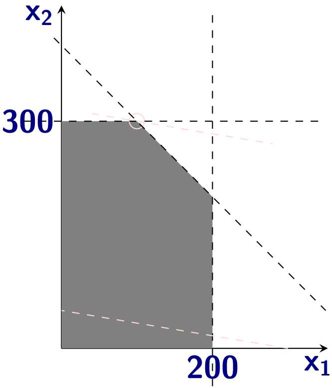
Input is matrix $\mathbf{A} = (\mathbf{a_{ij}}) \in \mathbb{R}^{\mathbf{n} \times \mathbf{d}}$, column vector $\mathbf{b} = (\mathbf{b_i}) \in \mathbb{R}^{\mathbf{n}}$, and row vector $\mathbf{c} = (\mathbf{c_j}) \in \mathbb{R}^{\mathbf{d}}$

Factory Example

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

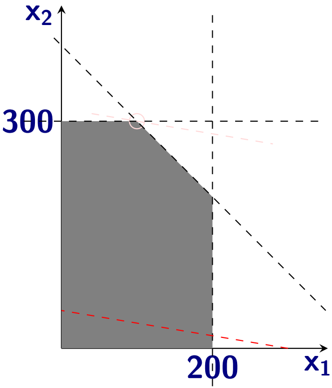
Suppose we want x_1, x_2 to be integer valued.

Factory Example Figure



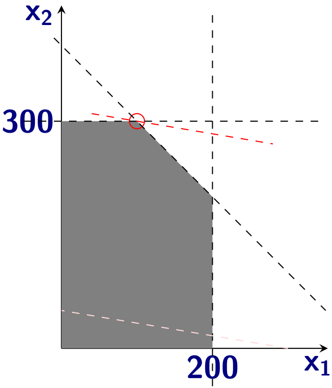
1. Feasible values of x_1 and x_2 are integer points in shaded region
2. Optimization function is a line; moving the line until it just leaves the final integer point in feasible region, gives optimal values

Factory Example Figure



1. Feasible values of $\mathbf{x_1}$ and $\mathbf{x_2}$ are integer points in shaded region
2. Optimization function is a line; moving the line until it just leaves the final integer point in feasible region, gives optimal values

Factory Example Figure



1. Feasible values of x_1 and x_2 are integer points in shaded region
2. Optimization function is a line; moving the line until it just leaves the final integer point in feasible region, gives optimal values

Integer Programming

Can model many difficult discrete optimization problems as integer programs!

Therefore integer programming is a hard problem. NP-hard.

Can relax integer program to linear program and *approximate*.

Practice: integer programs are solved by a variety of methods

1. branch and bound
2. branch and cut
3. adding cutting planes
4. linear programming plays a fundamental role

Linear Programs with Integer Vertices

Suppose we know that for a linear program *all* vertices have integer coordinates.

Then solving linear program is same as solving integer program. We know how to solve linear programs efficiently (polynomial time) and hence we get an integer solution for free!

Luck or *Structure*:

1. Linear program for flows with integer capacities have integer vertices
2. Linear program for matchings in bipartite graphs have integer vertices
3. A complicated linear program for matchings in general graphs have integer vertices.

All of above problems can hence be solved efficiently.

Linear Programs with Integer Vertices

Suppose we know that for a linear program *all* vertices have integer coordinates.
Then solving linear program is same as solving integer program. We know how to solve linear programs efficiently (polynomial time) and hence we get an integer solution for free!

Luck or *Structure*:

1. Linear program for flows with integer capacities have integer vertices
2. Linear program for matchings in bipartite graphs have integer vertices
3. A complicated linear program for matchings in general graphs have integer vertices.

All of above problems can hence be solved efficiently.

Linear Programs with Integer Vertices

Suppose we know that for a linear program *all* vertices have integer coordinates.
Then solving linear program is same as solving integer program. We know how to solve linear programs efficiently (polynomial time) and hence we get an integer solution for free!

Luck or *Structure*:

1. Linear program for flows with integer capacities have integer vertices
2. Linear program for matchings in bipartite graphs have integer vertices
3. A complicated linear program for matchings in general graphs have integer vertices.

All of above problems can hence be solved efficiently.

Linear Programs with Integer Vertices

Meta Theorem: A combinatorial optimization problem can be solved efficiently if and only if there is a linear program for problem with integer vertices.

Consequence of the Ellipsoid method for solving linear programming.

In a sense linear programming and other geometric generalizations such as convex programming are the most general problems that we can solve efficiently.

Summary

1. Linear Programming is a useful and powerful (modeling) problem.
2. Can be solved in polynomial time. Practical solvers available commercially as well as in open source. Whether there is a strongly polynomial time algorithm is a major open problem.
3. Geometry and linear algebra are important to understand the structure of LP and in algorithm design. Vertex solutions imply that LPs have poly-sized optimum solutions. This implies that LP is in **NP**.
4. Duality is a critical tool in the theory of linear programming. Duality implies the Linear Programming is in **co-NP**. Do you see why?
5. Integer Programming in **NP-Complete**. LP-based techniques critical in heuristically solving integer programs.

Notes

Notes

Notes

Notes