CS 473: Algorithms

Chandra Chekuri Ruta Mehta

University of Illinois, Urbana-Champaign

Fall 2016

Simplex and LP Duality

Lecture 19

October 28, 2016

Outline

Simplex: Intuition and Implementation Details

- Computing starting vertex: equivalent to solving an LP!

Infeasibility, Unboundedness, and Degeneracy.

Duality: Bounding the objective value through *weak-duality*

Strong Duality, Cone view.

Part I

Recall

Feasible Region and Convexity

Canonical Form

Given $\mathbf{A} \in \mathbf{R}^{n \times d}$, $\mathbf{b} \in \mathbf{R}^{n \times 1}$ and $\mathbf{c} \in \mathbf{R}^{1 \times d}$, find $\mathbf{x} \in \mathbf{R}^{d \times 1}$

$$\begin{aligned} \mathbf{max}: \quad & \mathbf{c} \cdot \mathbf{x} \\ \mathbf{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \end{aligned}$$

Linear Inequalities Define a Polyhedron

If $\sum_j a_{ij}x_j \leq b_i$ hold we equality, we say the constraint/hyperplane $\mathbf{i}$ is *tight*

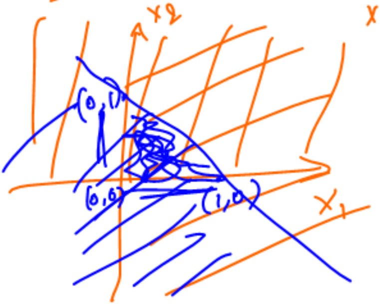
=

$x_1 \geq 0$

$x_2 \geq 0$

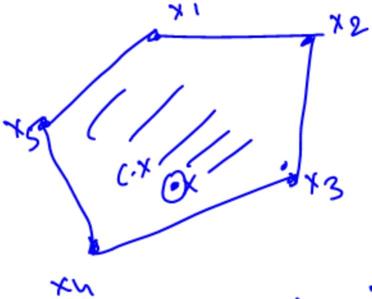
$x_1 + x_2 \leq 1$

$x_1 + x_2 = 1$



$x_1 \leq 1$

Optimizing linear objective over a polyhedron $\Rightarrow$ Vertex solution



$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 + \lambda_4 x_4 + \lambda_5 x_5$$

$$\forall \lambda_i \geq 0$$

$$\sum_i \lambda_i = 1$$

$$a = \sum_i \lambda_i a_i$$

$$\min_i a_i \leq a \leq \max_i a_i$$

$$(c \cdot x) = \sum_i \lambda_i (c \cdot x_i)$$

$$(c \cdot x) \leq \max_i (c \cdot x_i)$$

Vertex Solution

Optimizing linear objective over a polyhedron $\Rightarrow$ Vertex solution

Basic Feasible Solution: feasible, and $\mathbf{d}$ linearly independent tight constraints.

Summary

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.

Part II

Simplex

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 $\hat{\mathbf{x}}$ and optimal is $\mathbf{x}^*$, then $\mathbf{c} \cdot \mathbf{x}^* > \mathbf{c} \cdot \hat{\mathbf{x}}$.

Observations

For Simplex

Suppose we are at a non-optimal vertex $\hat{\mathbf{x}}$ and optimal is $\mathbf{x}^*$, 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 $\hat{\mathbf{x}}$ and optimal is $\mathbf{x}^*$, 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 $\hat{\mathbf{x}}$ and optimal is $\mathbf{x}^*$, 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!

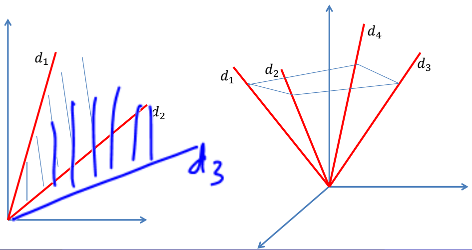
- $\mathbf{d} = \mathbf{x}^* - \hat{\mathbf{x}}$ is the direction from $\hat{\mathbf{x}}$ to $\mathbf{x}^*$.
- $(\mathbf{c} \cdot \mathbf{d}) = (\mathbf{c} \cdot \mathbf{x}^*) - (\mathbf{c} \cdot \hat{\mathbf{x}}) > \mathbf{0}$.
- In $\mathbf{x} = \hat{\mathbf{x}} + \delta\mathbf{d}$, as δ goes from $\mathbf{0}$ 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.,

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

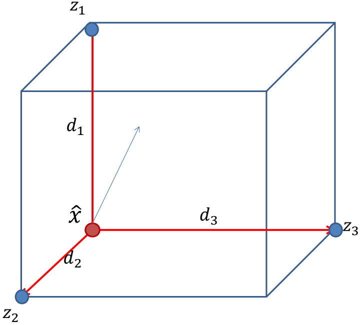


Cone at a Vertex

Let $\mathbf{z_1, \ldots, z_k}$ be the neighboring vertices of $\hat{\mathbf{x}}$. And let $\mathbf{d_i = z_i - \hat{x}}$ 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* $\mathbf{cone(\{d_1, \ldots, d_k\})}$.



Improving Direction Implies Improving Neighbor

Lemma

If $\mathbf{d} \in \mathbf{cone}(\{\mathbf{d_1}, \ldots, \mathbf{d_k}\})$ *and* $(\mathbf{c} \cdot \mathbf{d}) > \mathbf{0}$*, then there exists* $\mathbf{d_i}$ *such that* $(\mathbf{c} \cdot \mathbf{d_i}) > \mathbf{0}$*.*

Improving Direction Implies Improving Neighbor

Lemma

If $\mathbf{d} \in \mathbf{cone}(\{\mathbf{d_1}, \ldots, \mathbf{d_k}\})$ *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,

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

□

Improving Direction Implies Improving Neighbor

Lemma

If $\mathbf{d} \in \mathbf{cone}(\{\mathbf{d_1}, \ldots, \mathbf{d_k}\})$ *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,

$$\begin{aligned} (\mathbf{c} \cdot \mathbf{d}) &= (\mathbf{c} \cdot \textstyle\sum_{\mathbf{i=1}}^{\mathbf{k}} \boldsymbol{\lambda}_\mathbf{i} \mathbf{d_i}) \\ &= \textstyle\sum_{\mathbf{i=1}}^{\mathbf{k}} \boldsymbol{\lambda}_\mathbf{i} (\mathbf{c} \cdot \mathbf{d_i}) \\ &\leq \mathbf{0} \quad \text{A contradiction!} \end{aligned}$$

□

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: $\mathbf{Ax} \leq \mathbf{b}$

Faces

- 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

- Vertex: 0-dimensional face.
  Edge: 1D face. $\ldots$
  Hyperplane: $(\mathbf{d}-\mathbf{1})$D face.
- $\mathbf{r}$ linearly independent tight 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

- Vertex: 0-dimensional face. Edge: 1D face. $\ldots$ Hyperplane: $(\mathbf{d} - \mathbf{1})$D face.
- $\mathbf{r}$ linearly independent tight hyperplanes forms $\mathbf{d} - \mathbf{r}$ dimensional face.
- Vertices being of $\mathbf{0}$D, $\mathbf{d}$ L.I. tight hyperplanes.

How Many Neighbors a Vertex Has?

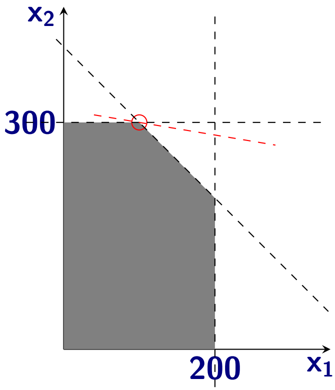
Geometric view...

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

Faces

- Vertex: 0-dimensional face. Edge: 1D face. $\ldots$ Hyperplane: $(\mathbf{d}-\mathbf{1})$D face.
- $\mathbf{r}$ linearly independent tight hyperplanes forms $\mathbf{d}-\mathbf{r}$ dimensional face.
- Vertices being of $\mathbf{0}$D, $\mathbf{d}$ L.I. tight hyperplanes.

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

- Vertex: 0-dimensional face. Edge: 1D face. **. . .** Hyperplane: $(\mathbf{d}-\mathbf{1})$D face.
- $\mathbf{r}$ linearly independent tight constraints forms $\mathbf{d}-\mathbf{r}$ dimensional face.
- Vertices (Basic feasible solution) has $\mathbf{d}$ L.I. tight constraints.

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

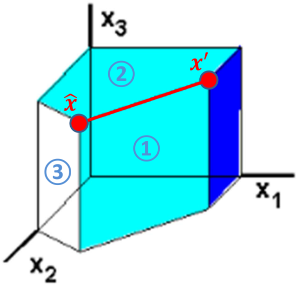


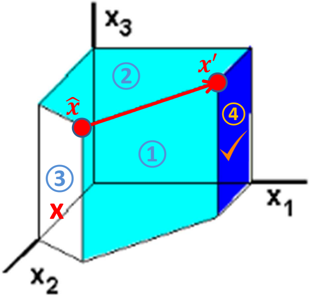
image source: webpage of Prof. Forbes W. Lewis

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 $\mathbf{d}-\mathbf{1}$ constraints are tight.
- These $\mathbf{d}-\mathbf{1}$ are also tight at both $\hat{\mathbf{x}}$ and $\mathbf{x}'$.
- One more constraints, say $\mathbf{i}$, is tight at $\hat{\mathbf{x}}$. "Relaxing" $\mathbf{i}$ 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 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 $\mathbf{d}$ "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.

Part III

Implementation of the Pivoting Step (Moving to an improving neighbor)

Moving to a Neighbor

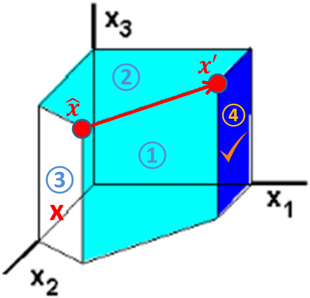
Fix a vertex $\hat{\mathbf{x}}$. Let the $\mathbf{d}$ hyperplanes/constraints tight at $\hat{\mathbf{x}}$ be,

$$\sum_{j=1}^{d} a_{ij} x_j = b_i, \quad 1 \leq i \leq d \quad \text{Equivalently, } \hat{A}x = \hat{b}$$

A neighbor vertex $\mathbf{x}'$ is connected to $\hat{\mathbf{x}}$ by an edge.

$\mathbf{d-1}$ hyperplanes tight on this edge.

To reach $\mathbf{x}'$, one hyperplane has to be relaxed, while maintaining other $\mathbf{d-1}$ tight.



Moving to a Neighbor (Contd.)

$$-\hat{\mathbf{A}}^{-1} = \begin{bmatrix} \vdots & & \vdots \\ \mathbf{d_1} & \dots & \mathbf{d_d} \\ \vdots & & \vdots \end{bmatrix}$$

Lemma

Moving in direction $\mathbf{d_i}$ from $\hat{\mathbf{x}}$, all except constraint $\mathbf{i}$ remain tight.

Proof.

For a small $\epsilon > \mathbf{0}$, let $\mathbf{y} = \hat{\mathbf{x}} + \epsilon(\mathbf{d_i})$, then

$$\hat{\mathbf{A}}\mathbf{y} = \hat{\mathbf{A}}(\hat{\mathbf{x}} + \epsilon\mathbf{d_i}) \;=\; \hat{\mathbf{A}}\hat{\mathbf{x}} + \epsilon\hat{\mathbf{A}}(-\hat{\mathbf{A}}^{-1})_{(.,\mathbf{i})}$$

Moving to a Neighbor (Contd.)

$$-\hat{\mathbf{A}}^{-1} = \begin{bmatrix} \vdots & & \vdots \\ \mathbf{d_1} & \dots & \mathbf{d_d} \\ \vdots & & \vdots \end{bmatrix}$$

Lemma

Moving in direction $\mathbf{d_i}$ *from* $\hat{\mathbf{x}}$*, all except constraint* $\mathbf{i}$ *remain tight.*

Proof.

For a small $\epsilon > 0$, let $\mathbf{y} = \hat{\mathbf{x}} + \epsilon(\mathbf{d_i})$, then

$$\begin{aligned} \hat{\mathbf{A}}\mathbf{y} = \hat{\mathbf{A}}(\hat{\mathbf{x}} + \epsilon\mathbf{d_i}) &= \hat{\mathbf{A}}\hat{\mathbf{x}} + \epsilon\hat{\mathbf{A}}(-\hat{\mathbf{A}}^{-1})_{(.,i)} \\ &= \hat{\mathbf{b}} + \epsilon[0, \dots, -1, \dots, 0]^T \end{aligned}$$

Clearly, $\sum_j \mathbf{a_{kj}y_j} = \mathbf{b_k}, \forall \mathbf{k} \neq \mathbf{i}$, and $\sum_j \mathbf{a_{ij}y_j} = \mathbf{b_i} - \epsilon < \mathbf{b_i}$. $\square$

Computing the Neighbor

Move in $\mathbf{d_i}$ direction from $\hat{\mathbf{x}}$, i.e., $\hat{\mathbf{x}} + \epsilon \mathbf{d_i}$, and STOP when hit a new hyperplane!

Need to ensure feasibility. Above lemma implies inequalities $\mathbf{1}$ through $\mathbf{d}$ will be satisfied. For any $\mathbf{k} > \mathbf{d}$, where $\mathbf{A_k}$ is $\mathbf{k^{th}}$ row of $\mathbf{A}$,

$$\begin{aligned} \mathbf{A_k} \cdot (\hat{\mathbf{x}} + \epsilon \mathbf{d_i}) \leq \mathbf{b_k} &\Rightarrow (\mathbf{A_k} \cdot \hat{\mathbf{x}}) + \epsilon (\mathbf{A_k} \cdot \mathbf{d_i}) \leq \mathbf{b_k} \\ &\Rightarrow \epsilon (\mathbf{A_k} \cdot \mathbf{d_i}) \leq \mathbf{b_k} - (\mathbf{A_k} \cdot \hat{\mathbf{x}}) \end{aligned}$$

Computing the Neighbor

Move in $\mathbf{d_i}$ direction from $\hat{\mathbf{x}}$, i.e., $\hat{\mathbf{x}} + \epsilon \mathbf{d_i}$, and STOP when hit a new hyperplane!

Need to ensure feasibility. Above lemma implies inequalities $\mathbf{1}$ through $\mathbf{d}$ will be satisfied. For any $\mathbf{k} > \mathbf{d}$, where $\mathbf{A_k}$ is $\mathbf{k^{th}}$ row of $\mathbf{A}$,

$$\begin{array}{lcl} \mathbf{A_k} \cdot (\hat{\mathbf{x}} + \epsilon \mathbf{d_i}) \leq \mathbf{b_k} & \Rightarrow & (\mathbf{A_k} \cdot \hat{\mathbf{x}}) + \epsilon(\mathbf{A_k} \cdot \mathbf{d_i}) \leq \mathbf{b_k} \\ & \Rightarrow & \epsilon(\mathbf{A_k} \cdot \mathbf{d_i}) \leq \mathbf{b_k} - (\mathbf{A_k} \cdot \hat{\mathbf{x}}) \\ (\text{If } (\mathbf{A_k} \cdot \mathbf{d_i}) > \mathbf{0}) & \Rightarrow & \epsilon \leq \frac{\mathbf{b_k} - (\mathbf{A_k} \cdot \hat{\mathbf{X}})}{\mathbf{A_k} \cdot \mathbf{d_i}} \end{array}$$

Computing the Neighbor

Move in $\mathbf{d_i}$ direction from $\hat{\mathbf{x}}$, i.e., $\hat{\mathbf{x}} + \epsilon \mathbf{d_i}$, and STOP when hit a new hyperplane!

Need to ensure feasibility. Above lemma implies inequalities $\mathbf{1}$ through $\mathbf{d}$ will be satisfied. For any $\mathbf{k} > \mathbf{d}$, where $\mathbf{A_k}$ is $\mathbf{k^{th}}$ row of $\mathbf{A}$,

$$\begin{array}{lcl} \mathbf{A_k} \cdot (\hat{\mathbf{x}} + \epsilon \mathbf{d_i}) \leq \mathbf{b_k} & \Rightarrow & (\mathbf{A_k} \cdot \hat{\mathbf{x}}) + \epsilon(\mathbf{A_k} \cdot \mathbf{d_i}) \leq \mathbf{b_k} \\ & \Rightarrow & \epsilon(\mathbf{A_k} \cdot \mathbf{d_i}) \leq \mathbf{b_k} - (\mathbf{A_k} \cdot \hat{\mathbf{x}}) \\ (\text{If } (\mathbf{A_k} \cdot \mathbf{d_i}) > \mathbf{0}) & \Rightarrow & \epsilon \leq \frac{\mathbf{b_k} - (\mathbf{A_k} \cdot \hat{\mathbf{x}})}{\mathbf{A_k} \cdot \mathbf{d_i}} \quad (\text{positive}) \end{array}$$

If moving towards hyperplane $\mathbf{k}$

Computing the Neighbor

Move in $\mathbf{d_i}$ direction from $\hat{\mathbf{x}}$, i.e., $\hat{\mathbf{x}} + \epsilon\mathbf{d_i}$, and STOP when hit a new hyperplane!

Need to ensure feasibility. Above lemma implies inequalities $\mathbf{1}$ through $\mathbf{d}$ will be satisfied. For any $\mathbf{k} > \mathbf{d}$, where $\mathbf{A_k}$ is $\mathbf{k^{th}}$ row of $\mathbf{A}$,

$$\begin{aligned}
\mathbf{A_k} \cdot (\hat{\mathbf{x}} + \epsilon\mathbf{d_i}) \leq \mathbf{b_k} &\Rightarrow (\mathbf{A_k} \cdot \hat{\mathbf{x}}) + \epsilon(\mathbf{A_k} \cdot \mathbf{d_i}) \leq \mathbf{b_k} \\
&\Rightarrow \epsilon(\mathbf{A_k} \cdot \mathbf{d_i}) \leq \mathbf{b_k} - (\mathbf{A_k} \cdot \hat{\mathbf{x}}) \\
(\text{If } (\mathbf{A_k} \cdot \mathbf{d_i}) > \mathbf{0}) &\Rightarrow \epsilon \leq \frac{\mathbf{b_k} - (\mathbf{A_k} \cdot \hat{\mathbf{x}})}{\mathbf{A_k} \cdot \mathbf{d_i}} \quad (\text{positive})
\end{aligned}$$

If moving towards hyperplane $\mathbf{k}$

$$(\text{If } (\mathbf{A_k} \cdot \mathbf{d_i}) < \mathbf{0}) \quad \Rightarrow \quad \epsilon \geq \frac{\mathbf{b_k} - (\mathbf{A_k} \cdot \hat{\mathbf{x}})}{\mathbf{A_k} \cdot \mathbf{d_i}} \quad (\text{negative})$$

If moving away from hyperplane $\mathbf{k}$.

No upper bound, and -ve lower bound!

Computing the Neighbor

Algorithm

NextVertex($\hat{\mathbf{x}}, \mathbf{d_i}$)
Set $\epsilon \leftarrow \infty$. μ^H
~~For $\mathbf{k = d+1 \ldots n}$~~ $\forall$ k s.t. k^{th} constraint is 'not' tight at $\hat{x}$
  $\epsilon' \leftarrow \frac{\mathbf{b_k - (A_k \cdot \hat{x})}}{\mathbf{A_k \cdot d_i}}$
  If $\epsilon' > \mathbf{0}$ and $\epsilon' < \epsilon$ then
    set $\epsilon \leftarrow \epsilon'$
$\epsilon > \mu^H$
If $\epsilon < \infty$ then return $\hat{\mathbf{x}} + \epsilon \mathbf{d_i}$.
Else return *null*.

If $(\mathbf{c} \cdot \mathbf{d_i}) > \mathbf{0}$ then the algorithm returns an *improving* neighbor.

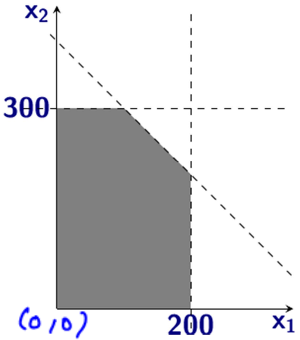
$c \cdot (\hat{x} + \epsilon d_i) = (c \cdot \hat{x}) + \epsilon (c \cdot d_i) > (c \cdot \hat{x})$

Factory Example

$$\hat{x} = (0, 0)$$

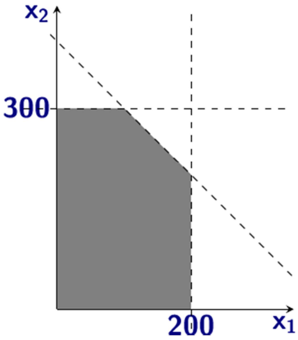
$-x_1 \leq 0$
$-x_2 \leq 0$

$$\begin{aligned} \max: \quad & x_1 + 6x_2 \\ \text{s.t.} \quad & 0 \leq x_1 \leq 200 \\ & 0 \leq x_2 \leq 300 \\ & x_1 + x_2 \leq 400 \end{aligned}$$



Factory Example

$$\max: \quad x_1 + 6x_2$$
$$\text{s.t.} \quad 0 \leq x_1 \leq 200$$
$$0 \leq x_2 \leq 300$$
$$x_1 + x_2 \leq 400$$



$$\hat{x} = (0, 0)$$

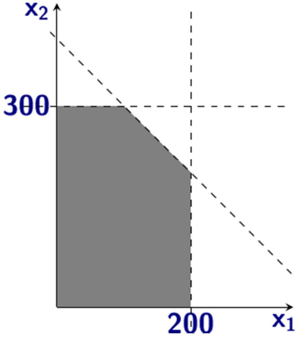
$$\hat{A} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$-\hat{A}^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = [d_1 \; d_2]$$

$$\begin{array}{cc} d_1 & d_2 \end{array}$$

Factory Example

$$\begin{aligned} \textbf{max}: &\quad x_1 + 6x_2 \\ \textbf{s.t.} &\quad 0 \le x_1 \le 200 \\ &\quad 0 \le x_2 \le 300 \\ &\quad x_1 + x_2 \le 400 \end{aligned}$$



$$\hat{x} = (0, 0)$$

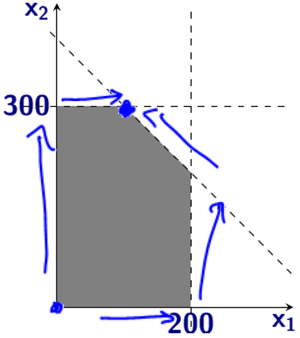
$$\hat{A} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$-\hat{A}^{-1} = \begin{array}{c} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \;\; d_1 \;\; d_2 \end{array} = [d_1 \; d_2]$$

Moving in direction $\mathbf{d_1}$ gives $(200, 0)$

Factory Example

$$\max: \quad x_1 + 6x_2$$
$$\text{s.t.} \quad 0 \le x_1 \le 200$$
$$0 \le x_2 \le 300$$
$$x_1 + x_2 \le 400$$



$$\hat{x} = (0, 0)$$

$$\hat{A} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$-\hat{A}^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = [d_1 \; d_2]$$

Moving in direction $\mathbf{d}_1$ gives $(200, 0)$

Moving in direction $\mathbf{d}_2$ gives $(0, 300)$.

Computing Starting Vertex

Equivalent to solving another LP!

Find an $\mathbf{x}$ such that $\mathbf{A}\mathbf{x} \leq \mathbf{b}$.
If $\mathbf{b} \geq \mathbf{0}$ then trivial!

Computing Starting Vertex

Equivalent to solving another LP!

Find an $\mathbf{x}$ such that $\mathbf{A}\mathbf{x} \leq \mathbf{b}$.
If $\mathbf{b} \geq \mathbf{0}$ then trivial! $\mathbf{x} = \mathbf{0}$. Otherwise.

Computing Starting Vertex

Equivalent to solving another LP!

$\hat{x}$ s.t.

$A\hat{x} \leq b$

$x = \hat{x}$

Find an $\mathbf{x}$ such that $\mathbf{Ax} \leq \mathbf{b}$.
If $\mathbf{b} \geq \mathbf{0}$ then trivial! $\mathbf{x} = \mathbf{0}$. Otherwise.

$$
\begin{aligned}
\textbf{min :} \quad & \mathbf{s} \quad \neq 0 \\
\textbf{s.t.} \quad & \textstyle\sum_j \mathbf{a_{ij}x_j} - \underset{=0}{\mathbf{s}} \leq \mathbf{b_i}, \quad \forall \mathbf{i} \\
& \mathbf{s} \geq \mathbf{0}
\end{aligned}
$$

Trivial feasible solution:

$x = 0 \Rightarrow -s \leq b_i \quad \forall i$

$\Rightarrow s \geq -b_i \quad \forall i$

$s = \max_i -b_i$

Computing Starting Vertex

Equivalent to solving another LP!

Find an $\mathbf{x}$ such that $\mathbf{Ax} \leq \mathbf{b}$.
If $\mathbf{b} \geq \mathbf{0}$ then trivial! $\mathbf{x} = \mathbf{0}$. Otherwise.

$$\begin{aligned} \mathbf{min:} \quad & \mathbf{s} \\ \mathbf{s.t.} \quad & \textstyle\sum_j a_{ij} x_j - s \leq b_i, \quad \forall i \\ & s \geq 0 \end{aligned}$$

Trivial feasible solution: $\mathbf{x} = \mathbf{0}$, $\mathbf{s} = |\min_i b_i|$.

Computing Starting Vertex

Equivalent to solving another LP!

Find an $\mathbf{x}$ such that $\mathbf{Ax} \leq \mathbf{b}$.
If $\mathbf{b} \geq \mathbf{0}$ then trivial! $\mathbf{x} = \mathbf{0}$. Otherwise.

$$\begin{array}{ll} \mathbf{min:} & \mathbf{s} \\ \mathbf{s.t.} & \sum_j a_{ij}x_j - s \leq b_i, \quad \forall i \\ & s \geq 0 \end{array}$$

Trivial feasible solution: $\mathbf{x} = \mathbf{0}$, $\mathbf{s} = |\min_i b_i|$.

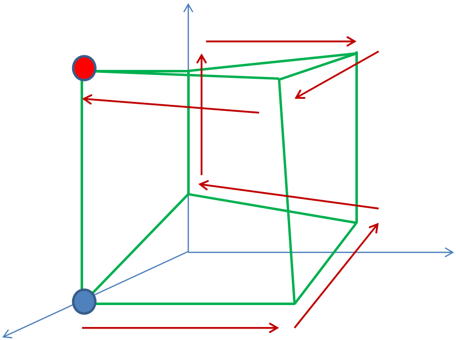
If $\mathbf{Ax} \leq \mathbf{b}$ feasible then optimal value of the above LP is $\mathbf{s} = \mathbf{0}$.

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

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

Infeasibility has to do only with constraints.

No starting vertex for Simplex. How to detect this?

Infeasibility: Example

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

Infeasibility has to do only with constraints.

No starting vertex for Simplex. How to detect this?

$$\text{LP} \quad \begin{array}{ll} \mathbf{min:} & \mathbf{s} \\ \mathbf{s.t.} & \mathbf{\sum_j a_{ij}x_j - s \le b_i, \quad \forall i} \\ & \mathbf{s \ge 0} \end{array}$$

to find a feasible point will have positive optimal.

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

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

Unboundedness depends on both constraints and the objective function.

If unbounded in the direction of objective function, then NextVertex will eventually return *null*

Degeneracy and Cycling

More than $\mathbf{d}$ constraints are tight at vertex $\hat{\mathbf{x}}$. Say $\mathbf{d+1}$.

Suppose, we pick first $\mathbf{d}$ to form $\hat{\mathbf{A}}$, and compute directions $\mathbf{d_1, \ldots, d_d}$.

Degeneracy and Cycling

More than $\mathbf{d}$ constraints are tight at vertex $\hat{\mathbf{x}}$. Say $\mathbf{d+1}$.

Suppose, we pick first $\mathbf{d}$ to form $\hat{\mathbf{A}}$, and compute directions $\mathbf{d_1, \ldots, d_d}$.

Then NextVertex($\hat{\mathbf{x}}, \mathbf{d_i}$) will encounter $\mathbf{(d+1)^{th}}$ constraint with $\epsilon = 0$ as an upper bound. Hence it will return $\hat{\mathbf{x}}$ again.

Degeneracy and Cycling

More than $\mathbf{d}$ constraints are tight at vertex $\hat{\mathbf{x}}$. Say $\mathbf{d+1}$.

Suppose, we pick first $\mathbf{d}$ to form $\hat{\mathbf{A}}$, and compute directions $\mathbf{d_1, \ldots, d_d}$.

Then NextVertex($\hat{\mathbf{x}}, \mathbf{d_i}$) will encounter $(\mathbf{d+1})^{\mathbf{th}}$ constraint with $\epsilon = \mathbf{0}$ as an upper bound. Hence it will return $\hat{\mathbf{x}}$ again.

Same phenomenon will repeat!

Degeneracy and Cycling

More than $\mathbf{d}$ constraints are tight at vertex $\hat{\mathbf{x}}$. Say $\mathbf{d+1}$.

Suppose, we pick first $\mathbf{d}$ to form $\hat{\mathbf{A}}$, and compute directions $\mathbf{d_1, \ldots, d_d}$.

Then NextVertex($\hat{\mathbf{x}}, \mathbf{d_i}$) will encounter $\mathbf{(d+1)^{th}}$ constraint with $\epsilon = \mathbf{0}$ as an upper bound. Hence it will return $\hat{\mathbf{x}}$ again.

Same phenomenon will repeat!

This can be avoided by adding small random perturbation to $\mathbf{b_i}$s.

Feasible Solutions and Lower Bounds

Consider the program

$$\begin{array}{lrrl} \text{maximize} & 4x_1 + & 2x_2 & \\ \text{subject to} & x_1 + & 3x_2 & \leq 5 \\ & 2x_1 - & 4x_2 & \leq 10 \\ & x_1 + & x_2 & \leq 7 \\ & x_1 & & \leq 5 \end{array}$$

Feasible Solutions and Lower Bounds

Consider the program

$$\begin{array}{lrrl} \text{maximize} & 4x_1 + & 2x_2 & \\ \text{subject to} & x_1 + & 3x_2 & \leq 5 \\ & 2x_1 - & 4x_2 & \leq 10 \\ & x_1 + & x_2 & \leq 7 \\ & x_1 & & \leq 5 \end{array}$$

1. $(0,1)$ satisfies all the constraints and gives value 2 for the objective function.

Feasible Solutions and Lower Bounds

Consider the program

$$\begin{array}{lrrl} \text{maximize} & 4x_1 + & 2x_2 & \\ \text{subject to} & x_1 + & 3x_2 & \leq 5 \\ & 2x_1 - & 4x_2 & \leq 10 \\ & x_1 + & x_2 & \leq 7 \\ & x_1 & & \leq 5 \end{array}$$

1. $(0, 1)$ satisfies all the constraints and gives value 2 for the objective function.
2. Thus, optimal value σ^* is at least 4.

Feasible Solutions and Lower Bounds

Consider the program

$$
\begin{array}{lrrl}
\text{maximize} & 4x_1 + & 2x_2 & \\
\text{subject to} & x_1 + & 3x_2 & \leq 5 \\
 & 2x_1 - & 4x_2 & \leq 10 \\
 & x_1 + & x_2 & \leq 7 \\
 & x_1 & & \leq 5
\end{array}
$$

1. $(0, 1)$ satisfies all the constraints and gives value 2 for the objective function.
2. Thus, optimal value σ^* is at least 4.
3. $(2, 0)$ also feasible, and gives a better bound of 8.

Feasible Solutions and Lower Bounds

Consider the program

$$
\begin{array}{lrrl}
\text{maximize} & 4x_1 + & 2x_2 & \\
\text{subject to} & x_1 + & 3x_2 & \leq 5 \\
& 2x_1 - & 4x_2 & \leq 10 \\
& x_1 + & x_2 & \leq 7 \\
& x_1 & & \leq 5
\end{array}
$$

1. $(0, 1)$ satisfies all the constraints and gives value 2 for the objective function.
2. Thus, optimal value σ^* is at least 4.
3. $(2, 0)$ also feasible, and gives a better bound of 8.
4. How good is 8 when compared with σ^*?

Obtaining Upper Bounds

1. Let us multiply the first constraint by **2** and the and add it to second constraint

$$\begin{array}{rrrl} 2(& x_1 + & 3x_2 &) \le 2(5) \\ +1(& 2x_1 - & 4x_2 &) \le 1(10) \\ \hline & 4x_1 + & 2x_2 & \le 20 \end{array}$$

2. Thus, 20 is an upper bound on the optimum value!

Generalizing ...

1. Multiply first equation by $\mathbf{y_1}$, second by $\mathbf{y_2}$, third by $\mathbf{y_3}$ and fourth by $\mathbf{y_4}$ (both $\mathbf{y_1, y_2, y_3, y_4}$ being positive) and add

$$\begin{array}{rrrl}
y_1(& x_1 + & 3x_2 &) \leq y_1(5) \\
+y_2(& 2x_1 - & 4x_2 &) \leq y_2(10) \\
+y_3(& x_1 + & x_2 &) \leq y_3(7) \\
+y_4(& x_1 & &) \leq y_4(5) \\
\hline
\multicolumn{4}{l}{(y_1 + 2y_2 + y_3 + y_4)x_1 + (3y_1 - 4y_2 + y_3)x_2 \leq \ldots}
\end{array}$$

2. $\mathbf{5y_1 + 10y_2 + 7y_3 + 5y_4}$ is an upper bound, provided coefficients of $\mathbf{x_i}$ are same as in the objective function, i.e.,

$$y_1 + 2y_2 + y_3 + y_4 = 4 \quad 3y_1 - 4y_2 + y_3 = 2$$

3. The best upper bound is when $\mathbf{5y_1 + 10y_2 + 7y_3 + 5y_4}$ is minimized!

Dual LP: Example

Thus, the optimum value of program

$$\begin{aligned} \text{maximize} \quad & 4x_1 + 2x_2 \\ \text{subject to} \quad & x_1 + 3x_2 \leq 5 \\ & 2x_1 - 4x_2 \leq 10 \\ & x_1 + x_2 \leq 7 \\ & x_1 \leq 5 \end{aligned}$$

is upper bounded by the optimal value of the program

$$\begin{aligned} \text{minimize} \quad & 5y_1 + 10y_2 + 7y_3 + 5y_4 \\ \text{subject to} \quad & y_1 + 2y_2 + y_3 + y_4 = 4 \\ & 3y_1 - 4y_2 + y_3 = 2 \\ & y_1, y_2 \geq 0 \end{aligned}$$

Dual Linear Program

Given a linear program Π in canonical form

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

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

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

Dual Linear Program

Given a linear program Π in canonical form

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

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

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

Proposition

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

Duality Theorem

Theorem (Weak Duality)

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

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.