BBM402-Lecture 19: The primal and dual simplex algorithms

Lecturer: Lale Özkahya

Resources for the presentation:
https://courses.engr.illinois.edu/cs473/fa2016/lectures.html
Simplex: Intuition and Implementation Details

- Computing starting vertex: equivalent to solving an LP!

Infeasibility, Unboundedness, and Degeneracy.

Duality: Bounding the objective value through \textit{weak-duality}

Strong Duality, Cone view.
Part I

Recall
Feasible Region and Convexity

Canonical Form

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

\[
\begin{align*}
\text{max :} & \quad c \cdot x \\
\text{s.t.} & \quad Ax \leq b
\end{align*}
\]
Linear Inequalities Define a Polyhedron

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

Optimizing linear objective over a polyhedron ⇒ Vertex solution
Vertex Solution

Optimizing linear objective over a polyhedron ⇒ Vertex solution

*Basic Feasible Solution*: feasible, and $d$ linearly independent tight constraints.
Each linear constraint defines a **halfspace**.

Feasible region, which is an intersection of halfspaces, is a convex **polyhedron**.

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?
Suppose we are at a non-optimal vertex $\hat{x}$ and optimal is $x^*$, then $c \cdot x^* > c \cdot \hat{x}$.

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

$d = x^* - \hat{x}$ is the direction from $\hat{x}$ to $x^*$.

$(c \cdot d) = (c \cdot x^*) - (c \cdot \hat{x}) > 0$.

In $x = \hat{x} + \delta d$, as $\delta$ goes from 0 to 1, we move from $\hat{x}$ to $x^*$.

$c \cdot x = c \cdot \hat{x} + \delta (c \cdot d)$.

Strictly increasing with $\delta$!

Due to convexity, all of these are feasible points.
Suppose we are at a non-optimal vertex $\hat{x}$ and optimal is $x^*$, then $c \cdot x^* > c \cdot \hat{x}$.

How does $(c \cdot x)$ change as we move from $\hat{x}$ to $x^*$ on the line joining the two?
Suppose we are at a non-optimal vertex $\hat{x}$ and optimal is $x^*$, then $c \cdot x^* > c \cdot \hat{x}$.

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

Strictly increases!
Suppose we are at a non-optimal vertex $\hat{x}$ and optimal is $x^*$, then $c \cdot x^* > c \cdot \hat{x}$.

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

Strictly increases!

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

Definition

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

\[
\text{cone}(D) = \{d \mid d = \sum_{i=1}^{k} \lambda_i d_i, \text{ where } \lambda_i \geq 0, \forall i\}
\]
Cone at a Vertex

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

Lemma

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

If \( \mathbf{d} \in \text{cone}(\{\mathbf{d}_1, \ldots, \mathbf{d}_k\}) \) and \( (\mathbf{c} \cdot \mathbf{d}) > 0 \), then there exists \( \mathbf{d}_i \) such that \( (\mathbf{c} \cdot \mathbf{d}_i) > 0 \).
Lemma

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

Proof.

To the contrary suppose \((c \cdot d_i) \leq 0\), \( \forall i \leq k \).

Since \( d \) is a positive linear combination of \( d_i \)'s,

\[
(c \cdot d) = (c \cdot \sum_{i=1}^{k} \lambda_i d_i)
= \sum_{i=1}^{k} \lambda_i (c \cdot d_i)
\leq 0 \quad \text{A contradiction!}
\]
Lemma

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

Proof.

To the contrary suppose \( (c \cdot d_i) \leq 0, \forall i \leq k \).

Since \( d \) is a positive linear combination of \( d_i \)'s,

\[
(c \cdot d) = (c \cdot \sum_{i=1}^{k} \lambda_i d_i) \\
= \sum_{i=1}^{k} \lambda_i(c \cdot d_i) \\
\leq 0 \quad \text{A contradiction!}
\]

Theorem

If vertex \( \hat{x} \) is not optimal then it has a neighbor where cost improves.
How Many Neighbors a Vertex Has?

Geometric view…

\[ A \in \mathbb{R}^{n \times d} \quad (n > d), \quad b \in \mathbb{R}^n, \quad \text{the} \]
\[ \text{constraints are: } A x \leq b \]

Faces

- Vertex: 0-dimensional face.
- Edge: 1D face. …
- Hyperplane: \((d - 1)\)D face.
How Many Neighbors a Vertex Has?

Geometric view...

\[ A \in \mathbb{R}^{n \times d} \quad (n > d), \quad b \in \mathbb{R}^n, \quad \text{the}\] constraints are: \( Ax \leq b \)

**Faces**

- Vertex: 0-dimensional face.
- Edge: 1D face. . . .
- Hyperplane: \((d - 1)D\) face.
- \( r \) linearly independent tight hyperplanes forms \( d - r \) dimensional face.
A ∈ \( \mathbb{R}^{n \times d} \) (n > d), \( b \in \mathbb{R}^n \), the constraints are: \( Ax \leq b \)

**Faces**

- Vertex: 0-dimensional face.
- Edge: 1D face.
- Hyperplane: \((d - 1)\)D face.
- \( r \) linearly independent tight hyperplanes forms \( d - r \) dimensional face.
- Vertices being of 0D, \( d \) L.I. tight hyperplanes.
How Many Neighbors a Vertex Has?

Geometric view...

\[ A \in \mathbb{R}^{n \times d} \quad (n > d) \quad b \in \mathbb{R}^n, \quad \text{the} \quad \text{constraints are:} \quad Ax \leq b \]

**Faces**

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

In 2-dimension \((d = 2)\)
How Many Neighbors a Vertex Has?

Geometric view...

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

**Faces**

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

In 3-dimension \((d = 3)\)

(image source: webpage of Prof. Forbes W. Lewis)
How Many Neighbors a Vertex Has?

Geometry view...

One neighbor per tight hyperplane. Therefore typically \( d \).

- Suppose \( x' \) is a neighbor of \( \hat{x} \), then on the edge joining the two \( d - 1 \) constraints are tight.

- These \( d - 1 \) are also tight at both \( \hat{x} \) and \( x' \).

- One more constraints, say \( i \), is tight at \( \hat{x} \). “Relaxing” \( i \) at \( \hat{x} \) leads to \( x' \).
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.

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

Chandra & Ruta (UIUC) CS473 19 Fall 2016
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.
Part III

Implementation of the Pivoting Step (Moving to an Improving Neighbor)
Fix a vertex $\hat{x}$. Let the $d$ hyperplanes/constraints tight at $\hat{x}$ be,

$$
\sum_{j=1}^{d} a_{ij}x_j = b_i, \quad 1 \leq i \leq d
$$

Equivalently, $\hat{A}x = \hat{b}$

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

$d - 1$ hyperplanes tight on this edge.

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

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

Proof.

For a small $\epsilon > 0$, let $y = \hat{x} + \epsilon \mathbf{d}_i$, then

$$\hat{A}y = \hat{A}(\hat{x} + \epsilon \mathbf{d}_i) = \hat{A}\hat{x} + \epsilon \hat{A}(-\hat{A}^{-1})_{(.,i)}$$
Moving in direction $d_i$ from $\hat{x}$, all except constraint $i$ remain tight.

**Proof.**

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

\[
\hat{A}y = \hat{A}(\hat{x} + \epsilon d_i) = \hat{A}\hat{x} + \epsilon \hat{A}(\hat{A}^{-1})_{(.,i)} = \hat{b} + \epsilon [0, \ldots, -1, \ldots, 0]^T
\]

Clearly, $\sum_j a_{kj}y_j = b_k, \forall k \neq i$, and $\sum_j a_{ij}y_j = b_i - \epsilon < b_i$. \hfill \square
Move in $d_i$ direction from $\hat{x}$, i.e., $\hat{x} + \epsilon d_i$, and STOP when hit a new hyperplane!

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

$$A_k \cdot (\hat{x} + \epsilon d_i) \leq b_k \implies (A_k \cdot \hat{x}) + \epsilon (A_k \cdot d_i) \leq b_k$$
$$\implies \epsilon (A_k \cdot d_i) \leq b_k - (A_k \cdot \hat{x})$$
Computing the Neighbor Move in \(d_i\) direction from \(\hat{x}\), i.e., \(\hat{x} + \epsilon d_i\), and STOP when hit a new hyperplane!

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

\[
A_k \cdot (\hat{x} + \epsilon d_i) \leq b_k \implies (A_k \cdot \hat{x}) + \epsilon (A_k \cdot d_i) \leq b_k \\
\implies \epsilon (A_k \cdot d_i) \leq b_k - (A_k \cdot \hat{x})
\]

(If \((A_k \cdot d_i) > 0\)) \implies \(\epsilon \leq \frac{b_k - (A_k \cdot \hat{x})}{A_k \cdot d_i}\)
Computing the Neighbor Move in $d_i$ direction from $\hat{x}$, i.e., $\hat{x} + \epsilon d_i$, and STOP when hit a new hyperplane!

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

$$A_k \cdot (\hat{x} + \epsilon d_i) \leq b_k \Rightarrow (A_k \cdot \hat{x}) + \epsilon (A_k \cdot d_i) \leq b_k \Rightarrow \epsilon (A_k \cdot d_i) \leq b_k - (A_k \cdot \hat{x})$$

(If $(A_k \cdot d_i) > 0$) $\Rightarrow \epsilon \leq \frac{b_k - (A_k \cdot \hat{x})}{A_k \cdot d_i}$ (positive)

If moving towards hyperplane $k$
Computing the Neighbor

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

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

$$A_k \cdot (\hat{x} + \epsilon d_i) \leq b_k \Rightarrow (A_k \cdot \hat{x}) + \epsilon (A_k \cdot d_i) \leq b_k \Rightarrow \epsilon (A_k \cdot d_i) \leq b_k - (A_k \cdot \hat{x})$$

(If $(A_k \cdot d_i) > 0$) \[\Rightarrow \epsilon \leq \frac{b_k - (A_k \cdot \hat{x})}{A_k \cdot d_i} \text{ (positive)}\]

If moving towards hyperplane $k$

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

If moving away from hyperplane $k$.

No upper bound, and -ve lower bound!
Computing the Neighbor

Algorithm

NextVertex(\(\hat{x}, d_i\))
Set \(\epsilon \leftarrow \infty\).
For \(k = d + 1 \ldots n\)
\[\epsilon' \leftarrow \frac{b_k - (A_k \cdot \hat{x})}{A_k \cdot d_i}\]
If \(\epsilon' > 0\) and \(\epsilon' < \epsilon\) then
set \(\epsilon \leftarrow \epsilon'\)
If \(\epsilon < \infty\) then return \(\hat{x} + \epsilon d_i\).
Else return null.

If \((c \cdot d_i) > 0\) then the algorithm returns an improving neighbor.
max : $x_1 + 6x_2$

s.t.

$0 \leq x_1 \leq 200$

$0 \leq x_2 \leq 300$

$x_1 + x_2 \leq 400$

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

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

\[ \hat{x} = (0, 0) \]
\[ \hat{A} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \]

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

\[
\begin{align*}
\text{max : } & \quad x_1 + 6x_2 \\
\text{s.t. : } & \quad 0 \leq x_1 \leq 200 \\
& \quad 0 \leq x_2 \leq 300 \\
& \quad x_1 + x_2 \leq 400
\end{align*}
\]

\[
\hat{x} = (0, 0)
\]

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

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

Moving in direction \(d_1\) gives \((200, 0)\).
Factory Example

\[
\begin{align*}
\text{max : } & \quad x_1 + 6x_2 \\
\text{s.t. : } & \quad 0 \leq x_1 \leq 200 \\
& \quad 0 \leq x_2 \leq 300 \\
& \quad x_1 + x_2 \leq 400
\end{align*}
\]

\[
\hat{x} = (0, 0)
\]

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

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

Moving in direction \(d_1\) gives \((200, 0)\)

Moving in direction \(d_2\) gives \((0, 300)\).
Find an $x$ such that $Ax \leq b$. If $b \geq 0$ then trivial!
Computing Starting Vertex  
Equivalent to solving another LP!

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

\[
\begin{align*}
\text{min} & : s \\
\text{subject to} & : \sum_j a_{ij} x_j - s \leq b_i, \quad \forall i \\
& : s \geq 0
\end{align*}
\]

Trivial feasible solution: $x = 0$, $s = |\min_i b_i|$. If $Ax \leq b$ feasible then optimal value of the above LP is $s = 0$. 

---

Chandra & Ruta (UIUC)  
CS473  
Fall 2016
Find an $x$ such that $Ax \leq b$.
If $b \geq 0$ then trivial! $x = 0$. Otherwise.

$$\min : \quad s$$
$$\text{s.t.} : \quad \sum_j a_{ij}x_j - s \leq b_i, \quad \forall i$$
$$s \geq 0$$

Trivial feasible solution:
Computing Starting Vertex

Equivalent to solving another LP!

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

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

Trivial feasible solution: $x = 0$, $s = |\min_i b_i|$.
Find an $x$ such that $Ax \leq b$.
If $b \geq 0$ then trivial! $x = 0$. Otherwise.

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

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

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

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

Choosing which neighbor to move to can significantly affect running time

Very efficient Simplex-based algorithms exist

Simplex algorithm takes exponential time in the worst case but works extremely well in practice with many improvements over the years

Non Simplex based methods like interior point methods work well for large problems.
Major open problem for many years: is there a 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.
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.

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

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

- very practical for some large problems and beats simplex
- 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

maximize \( x_1 + 6x_2 \)
subject to \( x_1 \leq 2 \) \( x_2 \leq 1 \) \( x_1 + x_2 \geq 4 \) \( x_1, x_2 \geq 0 \)

Infeasibility has to do only with constraints.
Infeasibility: Example

maximize $x_1 + 6x_2$
subject to $x_1 \leq 2$ $x_2 \leq 1$ $x_1 + x_2 \geq 4$
$x_1, x_2 \geq 0$

Infeasibility has to do only with constraints.

No starting vertex for Simplex. How to detect this?
Infeasibility: Example

maximize \( x_1 + 6x_2 \)
subject to
\[
\begin{align*}
x_1 & \leq 2 \\
x_2 & \leq 1 \\
x_1 + x_2 & \geq 4 \\
x_1, x_2 & \geq 0
\end{align*}
\]

Infeasibility has to do only with constraints.

No starting vertex for Simplex. How to detect this?

\[
\begin{align*}
\text{min : } & s \\
\text{LP s.t. } & \sum_j a_{ij} x_j - s \leq b_i, \forall i \\
& s \geq 0
\end{align*}
\]

to find a feasible point will have positive optimal.
Unboundedness: Example

maximize $x_2$

$x_1 + x_2 \geq 2$

$x_1, x_2 \geq 0$

Unboundedness depends on both constraints and the objective function.
Unboundedness: Example

\[
\text{maximize } x_2 \\
\ x_1 + x_2 \geq 2 \\
\ x_1, x_2 \geq 0
\]

Unboundedness depends on both constraints and the objective function.

If unbounded in the direction of objective function, then NextVertex will eventually return \textit{null}.
More than $d$ constraints are tight at vertex $\hat{x}$. Say $d + 1$.

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

This can be avoided by adding small random perturbation to $b_i$s.
Degeneracy and Cycling

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

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

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

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

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

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

Same phenomenon will repeat!
Degeneracy and Cycling

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

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

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

Same phenomenon will repeat!

This can be avoided by adding small random perturbation to $b_i$s.
Consider the program

\[
\begin{align*}
\text{maximize} & \quad 4x_1 + 2x_2 \\
\text{subject to} & \quad x_1 + 3x_2 \leq 5 \\
& \quad 2x_1 - 4x_2 \leq 10 \\
& \quad x_1 + x_2 \leq 7 \\
& \quad x_1 \leq 5
\end{align*}
\]

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

\((2, 0)\) is also feasible, and gives a better bound of 8.

How good is 8 when compared with \(\sigma^*\)?
Consider the program

\[
\begin{align*}
\text{maximize} & \quad 4x_1 + 2x_2 \\
\text{subject to} & \quad x_1 + 3x_2 \leq 5 \\
& \quad 2x_1 - 4x_2 \leq 10 \\
& \quad x_1 + x_2 \leq 7 \\
& \quad x_1 \leq 5
\end{align*}
\]

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

\[
\begin{align*}
\text{maximize} & \quad 4x_1 + 2x_2 \\
\text{subject to} & \quad x_1 + 3x_2 \leq 5 \\
& \quad 2x_1 - 4x_2 \leq 10 \\
& \quad x_1 + x_2 \leq 7 \\
& \quad x_1 \leq 5
\end{align*}
\]

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

2. Thus, optimal value \(\sigma^*\) is at least 4.
Consider the program

\[
\begin{align*}
\text{maximize} & \quad 4x_1 + 2x_2 \\
\text{subject to} & \quad x_1 + 3x_2 \leq 5 \\
& \quad 2x_1 - 4x_2 \leq 10 \\
& \quad x_1 + x_2 \leq 7 \\
& \quad x_1 \leq 5
\end{align*}
\]

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

2. Thus, optimal value \(\sigma^*\) is at least 4.

3. \((2, 0)\) also feasible, and gives a better bound of 8.
Consider the program

\[
\begin{align*}
\text{maximize} & \quad 4x_1 + 2x_2 \\
\text{subject to} & \quad x_1 + 3x_2 \leq 5 \\
& \quad 2x_1 - 4x_2 \leq 10 \\
& \quad x_1 + x_2 \leq 7 \\
& \quad x_1 \leq 5
\end{align*}
\]

1. \((0, 1)\) satisfies all the constraints and gives value 2 for the objective function.
2. Thus, optimal value \(\sigma^*\) is at least 4.
3. \((2, 0)\) also feasible, and gives a better bound of 8.
4. How good is 8 when compared with \(\sigma^*\)?
Let us multiply the first constraint by 2 and add it to the second constraint:

\[
2(x_1 + 3x_2) + 1(2x_1 - 4x_2) \leq 2(5) + 1(10) \\
4x_1 + 2x_2 \leq 20
\]

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

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

$$
\begin{align*}
&y_1(x_1 + 3x_2) \leq y_1(5) \\
&+y_2(2x_1 - 4x_2) \leq y_2(10) \\
&+y_3(x_1 + x_2) \leq y_3(7) \\
&+y_4(x_1) \leq y_4(5)
\end{align*}
$$

$$(y_1 + 2y_2 + y_3 + y_4)x_1 + (3y_1 - 4y_2 + y_3)x_2 \leq \ldots$$

2. $5y_1 + 10y_2 + 7y_3 + 5y_4$ is an upper bound, provided coefficients of $x_i$ 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 $5y_1 + 10y_2 + 7y_3 + 5y_4$ is minimized!
Thus, the optimum value of program

\[
\begin{align*}
\text{maximize} & \quad 4x_1 + 2x_2 \\
\text{subject to} & \quad x_1 + 3x_2 \leq 5 \\
& \quad 2x_1 - 4x_2 \leq 10 \\
& \quad x_1 + x_2 \leq 7 \\
& \quad x_1 \leq 5
\end{align*}
\]

is upper bounded by the optimal value of the program

\[
\begin{align*}
\text{minimize} & \quad 5y_1 + 10y_2 + 7y_3 + 5y_4 \\
\text{subject to} & \quad y_1 + 2y_2 + y_3 + y_4 = 4 \\
& \quad 3y_1 - 4y_2 + y_3 = 2 \\
& \quad y_1, y_2 \geq 0
\end{align*}
\]
Dual Linear Program

Given a linear program $\Pi$ in canonical form

maximize $\sum_{j=1}^{d} c_j x_j$
subject to $\sum_{j=1}^{d} a_{ij} x_j \leq b_i \quad i = 1, 2, \ldots n$

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

minimize $\sum_{i=1}^{n} b_i y_i$
subject to $\sum_{i=1}^{n} y_i a_{ij} = c_j \quad j = 1, 2, \ldots d$
yi $\geq 0 \quad i = 1, 2, \ldots n$
Given a linear program Π in canonical form

\[
\text{maximize } \sum_{j=1}^{d} c_j x_j \\
\text{subject to } \sum_{j=1}^{d} a_{ij} x_j \leq b_i \quad i = 1, 2, \ldots, n
\]

the dual \textbf{Dual}(Π) is given by

\[
\text{minimize } \sum_{i=1}^{n} b_i y_i \\
\text{subject to } \sum_{i=1}^{n} y_i a_{ij} = c_j \quad j = 1, 2, \ldots, d \\
y_i \geq 0 \quad i = 1, 2, \ldots, n
\]

**Proposition**

\textbf{Dual(Dual(Π))} is equivalent to Π
Duality Theorem

Theorem (Weak Duality)

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

Theorem (Strong Duality)

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

Many applications! Maxflow-Mincut theorem can be deduced from duality.
**Theorem (Weak Duality)**

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

**Theorem (Strong Duality)**

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

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