Advanced Algorithms (Fall 2023)

Linear Programming Rounding

Lecturers: 尹一通，刘景铖，栗师

Nanjing University
Outline

1. Linear Programming and Rounding

2. Exact Algorithms Using LP: Integral Polytopes
   - Bipartite Matching Polytope
   - $s-t$ Flow Polytope
   - Weighted Interval Scheduling Problem

3. Approximation Algorithms Using LP: LP Rounding
   - 2-Approximation Algorithm for Weighted Vertex Cover
   - 2-Approximation Algorithm for Unrelated Machine Scheduling
Opti. Problem $X \Longleftrightarrow$ Integer Program (IP) $\xrightarrow{\text{relax}}$ LP

- Integer programming is NP-hard; linear programming is in P
- For some problems, $\text{LP} \equiv \text{IP} \Rightarrow \text{exact algorithms}$
- For some problems, $\text{LP} \not\equiv \text{IP}$ solve LP to obtain a fractional solution, round it to an integral solution $\Rightarrow \text{approximation algorithms}$
Algorithm Design Based on Linear Programming (LP)

- Opti. Problem $X \iff$ Integer Program (IP) $\xrightarrow{\text{relax}}$ LP

- Integer programming is NP-hard; linear programming is in P
Algorithm Design Based on Linear Programming (LP)

- Opti. Problem $X \iff$ Integer Program (IP) $\xrightarrow{\text{relax}}$ LP
- Integer programming is NP-hard; linear programming is in P
- For some problems LP $\equiv$ IP $\implies$ exact algorithms
Opti. Problem $X \iff \text{Integer Program (IP)} \overset{\text{relax}}{\Rightarrow} \text{LP}$

Integer programming is NP-hard; linear programming is in P

For some problems $\text{LP} \equiv \text{IP} \implies$ exact algorithms

For some problems, $\text{LP} \not\equiv \text{IP}$

- solve LP to obtain a fractional solution,
- round it to an integral solution

$\implies$ approximation algorithms
min \[ 7x_1 + 4x_2 \]
\[ x_1 + x_2 \geq 5 \]
\[ x_1 + 2x_2 \geq 6 \]
\[ 4x_1 + x_2 \geq 8 \]
\[ x_1, x_2 \geq 0 \]
Linear Programming (LP), Linear Program (LP)

\[
\begin{align*}
\text{min} & \quad 7x_1 + 4x_2 \\
& x_1 + x_2 \geq 5 \\
& x_1 + 2x_2 \geq 6 \\
& 4x_1 + x_2 \geq 8 \\
& x_1, x_2 \geq 0
\end{align*}
\]
min $7x_1 + 4x_2$

$x_1 + x_2 \geq 5$

$x_1 + 2x_2 \geq 6$

$4x_1 + x_2 \geq 8$

$x_1, x_2 \geq 0$

optimum solution: $x_1 = 1$, $x_2 = 4$

optimum value = $7 \times 1 + 4 \times 4 = 23$

general case: many variables and constraints, but objective and constraints are linear
\begin{align*}
\text{min} & \quad 7x_1 + 4x_2 \\
& \quad x_1 + x_2 \geq 5 \\
& \quad x_1 + 2x_2 \geq 6 \\
& \quad 4x_1 + x_2 \geq 8 \\
& \quad x_1, x_2 \geq 0
\end{align*}

Optimum solution: 
\begin{align*}
x_1 &= 1 \\
x_2 &= 4
\end{align*}

Optimum value = $7 \times 1 + 4 \times 4 = 23$
min \( 7x_1 + 4x_2 \)

- \( x_1 + x_2 \geq 5 \)
- \( x_1 + 2x_2 \geq 6 \)
- \( 4x_1 + x_2 \geq 8 \)
- \( x_1, x_2 \geq 0 \)

Optimum solution:
\( x_1 = 1 \), \( x_2 = 4 \)

Optimum value = \( 7 \times 1 + 4 \times 4 = 23 \)

General case: many variables and constraints, but objective and constraints are linear.
min \quad 7x_1 + 4x_2 \\
\quad x_1 + x_2 \geq 5 \\
\quad x_1 + 2x_2 \geq 6 \\
4x_1 + x_2 \geq 8 \\
\quad x_1, x_2 \geq 0

- optimum solution: 
  \quad x_1 = 1, x_2 = 4 \\
- optimum value = 
  \quad 7 \times 1 + 4 \times 4 = 23
Linear Programming (LP), Linear Program (LP)

\[
\begin{align*}
\text{min} & \quad 7x_1 + 4x_2 \\
& x_1 + x_2 \geq 5 \\
& x_1 + 2x_2 \geq 6 \\
& 4x_1 + x_2 \geq 8 \\
& x_1, x_2 \geq 0
\end{align*}
\]

- optimum solution: 
  \[x_1 = 1, \quad x_2 = 4\]
- optimum value = 
  \[7 \times 1 + 4 \times 4 = 23\]
- general case: many variables and constraints, but objective and constraints are linear
Standard Form of Linear Programs

\[
\begin{align*}
\text{min} & \quad c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \\
a_{1,1} x_1 + a_{1,2} x_2 + \cdots + a_{1,n} x_n & \geq b_1 \\
a_{2,1} x_1 + a_{2,2} x_2 + \cdots + a_{2,n} x_n & \geq b_2 \\
\vdots & \quad \vdots & \quad \vdots & \quad \vdots \\
a_{m,1} x_1 + a_{m,2} x_2 + \cdots + a_{m,n} x_n & \geq b_m \\
x_1, x_2, \cdots, x_n & \geq 0
\end{align*}
\]

- \( n \): number of variables
- \( m \): number of constraints
Standard Form of Linear Programs

\[ \begin{align*}
\text{min} & \quad c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \\
& a_{1,1} x_1 + a_{1,2} x_2 + \cdots + a_{1,n} x_n \geq b_1 \\
& a_{2,1} x_1 + a_{2,2} x_2 + \cdots + a_{2,n} x_n \geq b_2 \\
& \quad \vdots \quad \vdots \quad \vdots \\
& a_{m,1} x_1 + a_{m,2} x_2 + \cdots + a_{m,n} x_n \geq b_m
\end{align*} \]

\[ \begin{align*}
x_1, x_2, \ldots, x_n \geq 0
\end{align*} \]

- \( n \): number of variables
- \( m \): number of constraints

- \( \leq \) constraints? equilities?
Standard Form of Linear Programs

\[
\begin{align*}
\text{min} \quad & c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \\
& a_{1,1} x_1 + a_{1,2} x_2 + \cdots + a_{1,n} x_n \geq b_1 \\
& a_{2,1} x_1 + a_{2,2} x_2 + \cdots + a_{2,n} x_n \geq b_2 \\
& \quad \vdots \quad \vdots \quad \vdots \\
& a_{m,1} x_1 + a_{m,2} x_2 + \cdots + a_{m,n} x_n \geq b_m \\
& x_1, x_2, \cdots, x_n \geq 0
\end{align*}
\]

- \( n \): number of variables
- \( m \): number of constraints
- \( \leq \): constraints? equalities?
- Variables can be negative? maximization problem?
Standard Form of Linear Programs

\[ x := \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n, \]

\[ A := \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} \in \mathbb{R}^{n \times m}, \]

\[ b := \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \in \mathbb{R}^m. \]

\[ c := \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} \in \mathbb{R}^n. \]
\[ \begin{align*}
& \text{min } c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \\
& a_{1,1} x_1 + a_{1,2} x_2 + \cdots + a_{1,n} x_n \geq b_1 \\
& a_{2,1} x_1 + a_{2,2} x_2 + \cdots + a_{2,n} x_n \geq b_2 \\
& \quad \vdots \\
& a_{m,1} x_1 + a_{m,2} x_2 + \cdots + a_{m,n} x_n \geq b_m
\end{align*} \]

\[ x_1, x_2, \cdots, x_n \geq 0 \]

\[ \geq: \text{ coordinate-wise less than or equal to} \]

**Standard Form of Linear Program**

\[ \begin{align*}
& \text{min } c^T x \\
& Ax \geq b \\
& x \geq 0
\end{align*} \]
[Fourier, 1827]: Fourier-Motzkin elimination method

[Kantorovich, Koopmans 1939]: formulated the general linear programming problem
History

- [Fourier, 1827]: Fourier-Motzkin elimination method
- [Kantorovich, Koopmans 1939]: formulated the general linear programming problem
- [Dantzig 1946]: simplex method
- [Khachiyan 1979]: ellipsoid method, polynomial time, proved linear programming is in $P$
- [Karmarkar, 1984]: interior-point method, polynomial time, algorithm is practical
- feasible region: the set of $x$’s satisfying $Ax \geq b, x \geq 0$
**Preliminaries**

- **feasible region**: the set of $x$’s satisfying $Ax \geq b, x \geq 0$
- feasible region is a **polyhedron**

![Polyhedron](image-url)
• feasible region: the set of $x$’s satisfying $Ax \geq b, x \geq 0$

• feasible region is a polyhedron

• if every coordinate has an upper and lower bound in the polyhedron, then the polyhedron is a polytope
x is a convex combination of $x^{(1)}, x^{(2)}, \cdots, x^{(t)}$ if the following condition holds: there exist $\lambda_1, \lambda_2, \cdots, \lambda_t \in [0, 1]$ such that

$$\lambda_1 + \lambda_2 + \cdots + \lambda_t = 1, \quad \lambda_1 x^{(1)} + \lambda_2 x^{(2)} + \cdots + \lambda_t x^{(t)} = x$$
**Preliminaries**

- $x$ is a **convex combination** of $x^{(1)}, x^{(2)}, \cdots, x^{(t)}$ if the following condition holds: there exist $\lambda_1, \lambda_2, \cdots, \lambda_t \in [0, 1]$ such that

\[
\lambda_1 + \lambda_2 + \cdots + \lambda_t = 1, \quad \lambda_1 x^{(1)} + \lambda_2 x^{(2)} + \cdots + \lambda_t x^{(t)} = x
\]
The set of convex combinations of \( x^{(1)}, x^{(2)}, \ldots, x^{(t)} \) is called the convex hull of these points.
$x$ is a convex combination of $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$ if the following condition holds: there exist $\lambda_1, \lambda_2, \ldots, \lambda_t \in [0, 1]$ such that

$$\lambda_1 + \lambda_2 + \cdots + \lambda_t = 1, \quad \lambda_1 x^{(1)} + \lambda_2 x^{(2)} + \cdots + \lambda_t x^{(t)} = x$$
**Preliminaries**

- $x$ is a **convex combination** of $x^{(1)}, x^{(2)}, \cdots, x^{(t)}$ if the following condition holds: there exist $\lambda_1, \lambda_2, \cdots, \lambda_t \in [0, 1]$ such that

$$\lambda_1 + \lambda_2 + \cdots + \lambda_t = 1, \quad \lambda_1 x^{(1)} + \lambda_2 x^{(2)} + \cdots + \lambda_t x^{(t)} = x$$
**Preliminaries**

- *x* is a **convex combination** of \( x^{(1)}, x^{(2)}, \ldots, x^{(t)} \) if the following condition holds: there exist \( \lambda_1, \lambda_2, \ldots, \lambda_t \in [0, 1] \) such that
  \[
  \lambda_1 + \lambda_2 + \cdots + \lambda_t = 1, \quad \lambda_1 x^{(1)} + \lambda_2 x^{(2)} + \cdots + \lambda_t x^{(t)} = x
  \]

- the set of convex combinations of \( x^{(1)}, x^{(2)}, \ldots, x^{(t)} \) is called the **convex hull** of these points

\[
\begin{align*}
\frac{2}{3} x^1 + \frac{1}{3} x^2 &= x^1 \\
0.3 x^1 + 0.6 x^2 + 0.1 x^3 &= x^3
\end{align*}
\]
$x$ is a **convex combination** of $x^{(1)}, x^{(2)}, \cdots, x^{(t)}$ if the following condition holds: there exist $\lambda_1, \lambda_2, \cdots, \lambda_t \in [0, 1]$ such that

$$\lambda_1 + \lambda_2 + \cdots + \lambda_t = 1, \quad \lambda_1 x^{(1)} + \lambda_2 x^{(2)} + \cdots + \lambda_t x^{(t)} = x$$

the set of convex combinations of $x^{(1)}, x^{(2)}, \cdots, x^{(t)}$ is called the **convex hull** of these points

\[
\text{convex-hull} \left( \{ x^1, x^2 \} \right)
\]

\[
\text{convex-hull} \left( \{ x^1, x^2, x^3 \} \right)
\]
**Preliminaries**

- $x$ is a **convex combination** of $x^{(1)}, x^{(2)}, \cdots, x^{(t)}$ if the following condition holds: there exist $\lambda_1, \lambda_2, \cdots, \lambda_t \in [0, 1]$ such that

$$\lambda_1 + \lambda_2 + \cdots + \lambda_t = 1, \quad \lambda_1 x^{(1)} + \lambda_2 x^{(2)} + \cdots + \lambda_t x^{(t)} = x$$

- the set of convex combinations of $x^{(1)}, x^{(2)}, \cdots, x^{(t)}$ is called the **convex hull** of these points
Preliminaries

- $x$ is a **convex combination** of $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$ if the following condition holds: there exist $\lambda_1, \lambda_2, \ldots, \lambda_t \in [0, 1]$ such that

$$\lambda_1 + \lambda_2 + \cdots + \lambda_t = 1, \quad \lambda_1 x^{(1)} + \lambda_2 x^{(2)} + \cdots + \lambda_t x^{(t)} = x$$

- the set of convex combinations of $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$ is called the **convex hull** of these points

\[
\text{convex-hull} \left( \{x^1, x^2\} \right)
\]

\[
\text{convex-hull} \left( \{x^1, x^2, x^3\} \right)
\]
let $P$ be polytope, $x \in P$. If there are no other points $x', x'' \in P$ such that $x$ is a convex combination of $x'$ and $x''$, then $x$ is called a vertex/extreme point of $P$. 

Lemma: A polytope has finite number of vertices, and it is the convex hull of the vertices.
let $P$ be polytope, $x \in P$. If there are no other points $x', x'' \in P$ such that $x$ is a convex combination of $x'$ and $x''$, then $x$ is called a vertex/extreme point of $P$.
let $P$ be polytope, $x \in P$. If there are no other points $x', x'' \in P$ such that $x$ is a convex combination of $x'$ and $x''$, then $x$ is called a vertex/extreme point of $P$. 
let \( P \) be polytope, \( x \in P \). If there are no other points \( x', x'' \in P \) such that \( x \) is a convex combination of \( x' \) and \( x'' \), then \( x \) is called a vertex/extreme point of \( P \).
let $P$ be polytope, $x \in P$. If there are no other points $x', x'' \in P$ such that $x$ is a convex combination of $x'$ and $x''$, then $x$ is called a vertex/extreme point of $P$. 

A polytope has finite number of vertices, and it is the convex hull of the vertices.
Preliminaries

- let $P$ be polytope, $x \in P$. If there are no other points $x', x'' \in P$ such that $x$ is a convex combination of $x'$ and $x''$, then $x$ is called a vertex/extreme point of $P$. 

![Diagram of a polytope with vertices indicated.](attachment)
Let $P$ be polytope, $x \in P$. If there are no other points $x', x'' \in P$ such that $x$ is a convex combination of $x'$ and $x''$, then $x$ is called a vertex/extreme point of $P$. 

**Lemma**

A polytope has finite number of vertices, and it is the convex hull of the vertices.
Let $P$ be polytope, $x \in P$. If there are no other points $x', x'' \in P$ such that $x$ is a convex combination of $x'$ and $x''$, then $x$ is called a vertex/extreme point of $P$. 

A polytope has a finite number of vertices, and it is the convex hull of the vertices.
let $P$ be polytope, $x \in P$. If there are no other points $x', x'' \in P$ such that $x$ is a convex combination of $x'$ and $x''$, then $x$ is called a vertex/extreme point of $P$.

**Lemma** A polytope has finite number of vertices, and it is the convex hull of the vertices.
let $P$ be polytope, $x \in P$. If there are no other points $x', x'' \in P$ such that $x$ is a convex combination of $x'$ and $x''$, then $x$ is called a **vertex/extreme point of $P$**.

**Lemma**  A polytope has finite number of vertices, and it is the convex hull of the vertices.

$$P = \text{convex-hull}(\{x^1, x^2, x^3, x^4, x^5\})$$
**Lemma** Let $x \in \mathbb{R}^n$ be an extreme point in a $n$-dimensional polytope. Then, there are $n$ constraints in the definition of the polytope, such that $x$ is the unique solution to the linear system obtained from the $n$ constraints by replacing inequalities to equalities.
Lemma  Let $x \in \mathbb{R}^n$ be an extreme point in a $n$-dimensional polytope. Then, there are $n$ constraints in the definition of the polytope, such that $x$ is the unique solution to the linear system obtained from the $n$ constraints by replacing inequalities to equalities.
Lemma  Let $x \in \mathbb{R}^n$ be an extreme point in a $n$-dimensional polytope. Then, there are $n$ constraints in the definition of the polytope, such that $x$ is the unique solution to the linear system obtained from the $n$ constraints by replacing inequalities to equalities.
Lemma Let $x \in \mathbb{R}^n$ be an extreme point in a $n$-dimensional polytope. Then, there are $n$ constraints in the definition of the polytope, such that $x$ is the unique solution to the linear system obtained from the $n$ constraints by replacing inequalities to equalities.

Lemma If the feasible region of a linear program is a polytope, then the optimum value can be attained at some vertex of the polytope.
Lemma  Let $x \in \mathbb{R}^n$ be an extreme point in a $n$-dimensional polytope. Then, there are $n$ constraints in the definition of the polytope, such that $x$ is the unique solution to the linear system obtained from the $n$ constraints by replacing inequalities to equalities.

Lemma  If the feasible region of a linear program is a polytope, then the optimum value can be attained at some vertex of the polytope.

Special cases (for minimization linear programs):
- if feasible region is empty, then its value is $\infty$
- if the feasible region is unbounded, then its value can be $-\infty$
## Algorithms for Linear Programming

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Running Time</th>
<th>Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplex Method</td>
<td>exponential time</td>
<td>fast</td>
</tr>
<tr>
<td>Ellipsoid Method</td>
<td>polynomial time</td>
<td>slow</td>
</tr>
<tr>
<td>Interior Point Method</td>
<td>polynomial time</td>
<td>fast</td>
</tr>
</tbody>
</table>
Simplex Method

- [Dantzig, 1946]

- move from one vertex to another, so as to improve the objective
- repeat until we reach an optimum vertex

[ Spielman-Teng, 2002]: smoothed analysis
Simplex Method

- [Dantzig, 1946]

- move from one vertex to another, so as to improve the objective

- repeat until we reach an optimum vertex

[Algorithm runs fast in practice]

[Spielman-Teng, 2002]: smoothed analysis

The number of iterations might be exponentially large, but the algorithm runs fast in practice.
Simplex Method

[Dantzig, 1946]

- move from one vertex to another, so as to improve the objective
- repeat until we reach an optimum vertex

The number of iterations might be exponentially large; but the algorithm runs fast in practice [Spielman-Teng, 2002]: smoothed analysis.
Simplex Method

- [Dantzig, 1946]

- move from one vertex to another, so as to improve the objective
- repeat until we reach an optimum vertex

[Algorithm runs fast in practice; Spielman-Teng, 2002]: smoothed analysis
Simplex Method

- [Dantzig, 1946]
  - move from one vertex to another, so as to improve the objective
  - repeat until we reach an optimum vertex

objective improves
Simplex Method

- [Dantzig, 1946]
  - move from one vertex to another, so as to improve the objective
  - repeat until we reach an optimum vertex

The number of iterations might be exponentially large; but the algorithm runs fast in practice [Spielman-Teng, 2002]: smoothed analysis
[Dantzig, 1946]

- move from one vertex to another, so as to improve the objective
- repeat until we reach an optimum vertex
Simplex Method

- [Dantzig, 1946]

- move from one vertex to another, so as to improve the objective
- repeat until we reach an optimum vertex

- the number of iterations might be exponentially large; but algorithm runs fast in practice
Simplex Method

- [Dantzig, 1946]

- move from one vertex to another, so as to improve the objective
- repeat until we reach an optimum vertex

- the number of iterations might be exponentially large; but algorithm runs fast in practice
- [Spielman-Teng, 2002]: smoothed analysis
Interior Point Method

- [Karmarkar, 1984]

  - keep the solution inside the polytope
  - design penalty function so that the solution is not too close to the boundary
  - the final solution will be arbitrarily close to the optimum solution
**Interior Point Method**

- [Karmarkar, 1984]

- Keep the solution inside the polytope
- Design penalty function so that the solution is not too close to the boundary
- The final solution will be arbitrarily close to the optimum solution
Interior Point Method

- [Karmarkar, 1984]
  - keep the solution inside the polytope
  - design penalty function so that the solution is not too close to the boundary
  - the final solution will be arbitrarily close to the optimum solution
Interior Point Method

- [Karmarkar, 1984]

- keep the solution inside the polytope
- design penalty function so that the solution is not too close to the boundary
- the final solution will be arbitrarily close to the optimum solution
Interior Point Method

- [Karmarkar, 1984]

- keep the solution inside the polytope
- design penalty function so that the solution is not too close to the boundary
- the final solution will be arbitrarily close to the optimum solution
[Karmarkar, 1984]

- keep the solution inside the polytope
- design penalty function so that the solution is not too close to the boundary
- the final solution will be arbitrarily close to the optimum solution
- [Karmarkar, 1984]

- keep the solution inside the polytope

- design penalty function so that the solution is not too close to the boundary

- the final solution will be arbitrarily close to the optimum solution
Interior Point Method

- [Karmarkar, 1984]

- keep the solution inside the polytope
- design penalty function so that the solution is not too close to the boundary
- the final solution will be arbitrarily close to the optimum solution
[Karmarkar, 1984]

- keep the solution inside the polytope
- design penalty function so that the solution is not too close to the boundary
- the final solution will be arbitrarily close to the optimum solution

- polynomial time
Ellipsoid Method

- [Khachiyan, 1979]

The Ellipsoid Method is used to decide if the feasible region is empty or not. It maintains an ellipsoid that contains the feasible region. If the center of the ellipsoid is in the feasible region:
- Yes: then the feasible region is not empty.
- No: cut the ellipsoid in half, find a smaller ellipsoid to enclose the half-ellipsoid, and repeat.

This method runs in polynomial time, but it is impractical.
Ellipsoid Method

- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not
Ellipsoid Method

- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not

  - maintain an ellipsoid that contains the feasible region

  query a separation oracle if the center of ellipsoid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat

polynomial time, but impractical
Ellipsoid Method

- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not

- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsoid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat
Ellipsoid Method

- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not

- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsoid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat
Ellipsoid Method

- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not

- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsoid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat

\[
P
\]
Ellipsoid Method

- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not

- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsoid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat
Ellipsoid Method

- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not
- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsoid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat
Ellipsoid Method

- [Khachiyan, 1979]
- Used to decide if the feasible region is empty or not

- Maintain an ellipsoid that contains the feasible region
- Query a separation oracle if the center of ellipsoid is in the feasible region:
  - Yes: then the feasible region is not empty
  - No: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat
Ellipsoid Method

- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not

- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsoid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat
Ellipsoid Method

- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not

- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsoid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat
Ellipsoid Method

- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not

- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsoid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat
Ellipsoid Method

- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not

- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsoid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat
Ellipsoid Method

- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not

- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsoid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat
Ellipsoid Method

- [Khachiyan, 1979]
  used to decide if the feasible region is empty or not

- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsoid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the ellipsoid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat

- polynomial time, but impractical
Q: The exact running time of these algorithms?
Q: The exact running time of these algorithms?

- it depends on many parameters: #variables, #constraints, #(non-zero coefficients), magnitude of integers
- precision issue
Q: The exact running time of these algorithms?

- it depends on many parameters: \#variables, \#constraints, \#(non-zero coefficients), magnitude of integers
- precision issue

Open Problem
Can linear programming be solved in strongly polynomial time algorithm?
Applications of Linear Programming

- domain: computer science, mathematics, operations research, economics
- types of problems: transportation, scheduling, clustering, network routing, resource allocation, facility location
Applications of Linear Programming

- domain: computer science, mathematics, operations research, economics
- types of problems: transportation, scheduling, clustering, network routing, resource allocation, facility location

Research Directions

- polynomial time exact algorithm
- polynomial time approximation algorithm
- sub-routines for the branch-and-bound method for integer programming
- other algorithmic models: online algorithm, distributed algorithms, dynamic algorithms, fast algorithms
Simple Example: Brewery Problem

- Small brewery produces ale and beer.
- Production limited by scarce resources: corn, hops, barley malt.
- Recipes for ale and beer require different proportions of resources.
Simple Example: Brewery Problem

- Small brewery produces ale and beer.
- Production limited by scarce resources: corn, hops, barley malt.
- Recipes for ale and beer require different proportions of resources.

<table>
<thead>
<tr>
<th>Beverage</th>
<th>Corn (pounds)</th>
<th>Hops (pounds)</th>
<th>Malt (pounds)</th>
<th>Profit ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ale (barrel)</td>
<td>5</td>
<td>4</td>
<td>35</td>
<td>13</td>
</tr>
<tr>
<td>Beer (barrel)</td>
<td>15</td>
<td>4</td>
<td>20</td>
<td>23</td>
</tr>
<tr>
<td>Constraint</td>
<td>480</td>
<td>160</td>
<td>1190</td>
<td></td>
</tr>
</tbody>
</table>
**Simple Example: Brewery Problem**

- Small brewery produces ale and beer.
- Production limited by scarce resources: corn, hops, barley malt.
- Recipes for ale and beer require different proportions of resources.

<table>
<thead>
<tr>
<th>Beverage</th>
<th>Corn (pounds)</th>
<th>Hops (pounds)</th>
<th>Malt (pounds)</th>
<th>Profit ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ale (barrel)</td>
<td>5</td>
<td>4</td>
<td>35</td>
<td>13</td>
</tr>
<tr>
<td>Beer (barrel)</td>
<td>15</td>
<td>4</td>
<td>20</td>
<td>23</td>
</tr>
<tr>
<td>Constraint</td>
<td>480</td>
<td>160</td>
<td>1190</td>
<td></td>
</tr>
</tbody>
</table>

- How can brewer maximize profits?

## Brewery Problem

<table>
<thead>
<tr>
<th>Beverage</th>
<th>Corn (pounds)</th>
<th>Hops (pounds)</th>
<th>Malt (pounds)</th>
<th>Profit ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ale (barrel)</td>
<td>5</td>
<td>4</td>
<td>35</td>
<td>13</td>
</tr>
<tr>
<td>Beer (barrel)</td>
<td>15</td>
<td>4</td>
<td>20</td>
<td>23</td>
</tr>
<tr>
<td>Constraint</td>
<td>480</td>
<td>160</td>
<td>1190</td>
<td></td>
</tr>
</tbody>
</table>

Constraint:

\[ 5x + 15y \leq 480 \]
\[ 4x + 4y \leq 160 \]
\[ 35x + 20y \leq 1190 \]
\[ x, y \geq 0 \]

## Brewery Problem

<table>
<thead>
<tr>
<th>Beverage</th>
<th>Corn (pounds)</th>
<th>Hops (pounds)</th>
<th>Malt (pounds)</th>
<th>Profit ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ale (barrel)</td>
<td>5</td>
<td>4</td>
<td>35</td>
<td>13</td>
</tr>
<tr>
<td>Beer (barrel)</td>
<td>15</td>
<td>4</td>
<td>20</td>
<td>23</td>
</tr>
<tr>
<td>Constraint</td>
<td>480</td>
<td>160</td>
<td>1190</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{max} & \quad 13x + 23y \\
5x + 15y & \leq 480 \\
4x + 4y & \leq 160 \\
35x + 20y & \leq 1190 \\
x, y & \geq 0
\end{align*}
\]

Outline

1. Linear Programming and Rounding

2. Exact Algorithms Using LP: Integral Polytopes
   - Bipartite Matching Polytope
   - $s$-$t$ Flow Polytope
   - Weighted Interval Scheduling Problem

3. Approximation Algorithms Using LP: LP Rounding
   - 2-Approximation Algorithm for Weighted Vertex Cover
   - 2-Approximation Algorithm for Unrelated Machine Scheduling
Def. A polytope $P \subseteq \mathbb{R}^n$ is said to be integral, if all vertices of $P$ are in $\mathbb{Z}^n$. 
Def. A polytope $P \subseteq \mathbb{R}^n$ is said to be integral, if all vertices of $P$ are in $\mathbb{Z}^n$.

For some combinatorial optimization problems, a polynomial-sized LP $Ax \leq b$ already defines an integral polytope, whose vertices correspond to valid integral solutions.
**Def.** A polytope $P \subseteq \mathbb{R}^n$ is said to be **integral**, if all vertices of $P$ are in $\mathbb{Z}^n$.

- For some combinatorial optimization problems, a polynomial-sized LP $Ax \leq b$ already defines an integral polytope, whose vertices correspond to valid integral solutions.
- Such a problem can be solved directly using the LP:

$$\max \ / \ \min \ c^T x \quad Ax \leq b.$$
Outline

1. Linear Programming and Rounding

2. Exact Algorithms Using LP: Integral Polytopes
   - Bipartite Matching Polytope
   - s-t Flow Polytope
   - Weighted Interval Scheduling Problem

3. Approximation Algorithms Using LP: LP Rounding
   - 2-Approximation Algorithm for Weighted Vertex Cover
   - 2-Approximation Algorithm for Unrelated Machine Scheduling
Example: Bipartite Matching Polytope

Maximum Weight Bipartite Matching

**Input:** bipartite graph $G = (L \cup R, E)$
edge weights $w \in \mathbb{Z}^E_{>0}$

**Output:** a matching $M \subseteq E$ so as to maximize $\sum_{e \in M} w_e$
**Example: Bipartite Matching Polytope**

### Maximum Weight Bipartite Matching

**Input:** bipartite graph $G = (L \cup R, E)$

- edge weights $w \in \mathbb{Z}^E_{>0}$

**Output:** a matching $M \subseteq E$ so as to maximize $\sum_{e \in M} w_e$

### LP Relaxation

\[
\begin{align*}
\max & \quad \sum_{e \in E} w_e x_e \\
\sum_{e \in \delta(v)} x_e & \leq 1 \quad \forall v \in L \cup R \\
x_e & \geq 0 \quad \forall e \in E
\end{align*}
\]
Example: Bipartite Matching Polytope

**Maximum Weight Bipartite Matching**

**Input:** bipartite graph $G = (L \cup R, E)$
edge weights $w \in \mathbb{Z}^E_{>0}$

**Output:** a matching $M \subseteq E$ so as to maximize $\sum_{e \in M} w_e$

**LP Relaxation**

$$\max \sum_{e \in E} w_e x_e$$

$$\sum_{e \in \delta(v)} x_e \leq 1 \quad \forall v \in L \cup R$$

$$x_e \geq 0 \quad \forall e \in E$$

- In IP: $x_e \in \{0, 1\}: e \in M$?
**Example: Bipartite Matching Polytope**

### Maximum Weight Bipartite Matching

**Input:** bipartite graph $G = (L \cup R, E)$
- edge weights $w \in \mathbb{Z}^E_{>0}$

**Output:** a matching $M \subseteq E$ so as to maximize $\sum_{e \in M} w_e$

---

### LP Relaxation

$$\begin{align*}
\text{max} & \quad \sum_{e \in E} w_e x_e \\
\sum_{e \in \delta(v)} x_e & \leq 1 \quad \forall v \in L \cup R \\
x_e & \geq 0 \quad \forall e \in E
\end{align*}$$

- In IP: $x_e \in \{0, 1\}: e \in M$?
Example: Bipartite Matching Polytope

Maximum Weight Bipartite Matching

**Input:** bipartite graph $G = (L \cup R, E)$
edge weights $w \in \mathbb{Z}^E_{>0}$

**Output:** a matching $M \subseteq E$ so as to maximize $\sum_{e \in M} w_e$

\[ \sum_j x_{ij} \leq 1 \quad \sum_i x_{ij} \leq 1 \]

LP Relaxation

$\max \sum_{e \in E} w_e x_e$
$\sum_{e \in \delta(v)} x_e \leq 1 \quad \forall v \in L \cup R$
$x_e \geq 0 \quad \forall e \in E$

\[ \text{In IP: } x_e \in \{0, 1\}: e \in M? \]
Example: Bipartite Matching Polytope

**Maximum Weight Bipartite Matching**

**Input:** bipartite graph \( G = (L \cup R, E) \)
edge weights \( w \in \mathbb{Z}_E^{>0} \)

**Output:** a matching \( M \subseteq E \) so as to maximize \( \sum_{e \in M} w_e \)

\[
\sum_j x_{ij} \leq 1 \quad \sum_i x_{ij} \leq 1
\]

**LP Relaxation**

\[
\max \sum_{e \in E} w_e x_e
\]

\[
\sum_{e \in \delta(v)} x_e \leq 1 \quad \forall v \in L \cup R
\]

\[
x_e \geq 0 \quad \forall e \in E
\]

- In LP: \( x_e \in \{0, 1\} \): \( e \in M \)?
- \( \chi^M \in \{0, 1\}^E \): \( \chi^M_e = 1 \) iff \( e \in M \)

**Theorem** The LP polytope is integral: It is the convex hull of \( \{\chi^M : M \text{ is a matching}\} \).
Theorem  The LP polytope is integral: It is the convex hull of \( \{ \chi^M : M \text{ is a matching} \} \).

Proof.

Take \( x \) in the polytope \( P \) prove: \( x \) non-integral \( \Rightarrow \) \( x \) non-vertex find \( x' \), \( x'' \) \( \in P \): \( x' \neq x'' \), \( x = \frac{1}{2}(x' + x'') \)

Case 1: fractional edges contain a cycle
- color edges in cycle blue and red
- \( x' \): +\( \epsilon \) for blue edges, -\( \epsilon \) for red edges
- \( x'' \): -\( \epsilon \) for blue edges, +\( \epsilon \) for red edges

Case 2: fractional edges form a forest
- color edges in a leaf-leaf path blue and red
- \( x' \): +\( \epsilon \) for blue edges, -\( \epsilon \) for red edges
- \( x'' \): -\( \epsilon \) for blue edges, +\( \epsilon \) for red edges
**Theorem**  The LP polytope is integral: It is the convex hull of \( \{\chi^M : M \text{ is a matching}\} \).

**Proof.**

- take \( x \) in the polytope \( P \)
Theorem  The LP polytope is integral: It is the convex hull of \( \{ \chi^M : M \text{ is a matching} \} \).

Proof.

- take \( x \) in the polytope \( P \)
- prove: \( x \) non-integral \( \implies \) \( x \) non-vertex
**Theorem**  The LP polytope is integral: It is the convex hull of \( \{\chi^M : M \text{ is a matching}\} \).

**Proof.**
- take \( x \) in the polytope \( P \)
- prove: \( x \) non integral \( \implies \) \( x \) non-vertex
- find \( x', x'' \in P : x' \neq x'', x = \frac{1}{2}(x' + x'') \)
**Theorem**  The LP polytope is integral: It is the convex hull of \{\chi^M : M \text{ is a matching}\}.

**Proof.**

- take \( x \) in the polytope \( P \)
- prove: \( x \) non integral \( \implies x \) non-vertex
- find \( x', x'' \in P: x' \neq x'', x = \frac{1}{2}(x' + x'') \)
- case 1: fractional edges contain a cycle
Theorem  The LP polytope is integral: It is the convex hull of \( \{ \chi^M : M \text{ is a matching} \} \).

Proof.
- take \( x \) in the polytope \( P \)
- prove: \( x \) non integral \( \implies \) \( x \) non-vertex
- find \( x', x'' \in P: x' \neq x'', x = \frac{1}{2}(x' + x'') \)
- case 1: fractional edges contain a cycle
  - color edges in cycle blue and red

[Diagram showing edge coloring]
**Theorem**  The LP polytope is integral: It is the convex hull of \( \{ \chi^M : M \text{ is a matching} \} \).

**Proof.**
- take \( x \) in the polytope \( P \)
- prove: \( x \) non integral \( \implies x \) non-vertex
- find \( x', x'' \in P: x' \neq x'', x = \frac{1}{2}(x' + x'') \)
- case 1: fractional edges contain a cycle
  - color edges in cycle blue and red
  - \( x' \): +\( \epsilon \) for blue edges, −\( \epsilon \) for red edges
  - \( x'' \): −\( \epsilon \) for blue edges, +\( \epsilon \) for red edges

![Diagram showing edges colored blue and red with annotations for each edge.]
**Theorem** The LP polytope is integral: It is the convex hull of \( \{\chi^M : M \text{ is a matching}\} \).

**Proof.**
- take \( x \) in the polytope \( P \)
- prove: \( x \) non integral \( \Rightarrow \) \( x \) non-vertex
- find \( x', x'' \in P: x' \neq x'', x = \frac{1}{2}(x' + x'') \)
- case 1: fractional edges contain a cycle
  - color edges in cycle blue and red
  - \( x' \): \(+\epsilon\) for blue edges, \(-\epsilon\) for red edges
  - \( x'' \): \(-\epsilon\) for blue edges, \(+\epsilon\) for red edges
- case 2: fractional edges form a forest
Theorem. The LP polytope is integral: It is the convex hull of \( \{ \chi^M : M \text{ is a matching} \} \).

Proof.
- take \( x \) in the polytope \( P \)
- prove: \( x \) non integral \( \implies \) \( x \) non-vertex
- find \( x', x'' \in P: x' \neq x'', x = \frac{1}{2}(x' + x'') \)
- case 1: fractional edges contain a cycle
  - color edges in cycle blue and red
  - \( x' \): \( +\epsilon \) for blue edges, \( -\epsilon \) for red edges
  - \( x'' \): \( -\epsilon \) for blue edges, \( +\epsilon \) for red edges
- case 2: fractional edges form a forest
  - color edges in a leaf-leaf path blue and red
**Theorem**  The LP polytope is integral: It is the convex hull of \( \{ \chi^M : M \text{ is a matching} \} \).

**Proof.**

- take \( x \) in the polytope \( P \)
- prove: \( x \) non integral \( \implies x \) non-vertex
- find \( x', x'' \in P: x' \neq x'', x = \frac{1}{2}(x' + x'') \)
- case 1: fractional edges contain a cycle
  - color edges in cycle blue and red
  - \( x' \): +\( \epsilon \) for blue edges, −\( \epsilon \) for red edges
  - \( x'' \): −\( \epsilon \) for blue edges, +\( \epsilon \) for red edges
- case 2: fractional edges form a forest
  - color edges in a leaf-leaf path blue and red
  - \( x' \): +\( \epsilon \) for blue edges, −\( \epsilon \) for red edges
  - \( x'' \): −\( \epsilon \) for blue edges, +\( \epsilon \) for red edges

\[\begin{align*}
1 \\
\pm \epsilon \\
\pm \epsilon \\
\end{align*}\]
Outline

1. Linear Programming and Rounding

2. Exact Algorithms Using LP: Integral Polytopes
   - Bipartite Matching Polytope
   - $s-t$ Flow Polytope
   - Weighted Interval Scheduling Problem

3. Approximation Algorithms Using LP: LP Rounding
   - 2-Approximation Algorithm for Weighted Vertex Cover
   - 2-Approximation Algorithm for Unrelated Machine Scheduling
Example: $s$-$t$ Flow Polytope

Flow Network

- directed graph $G = (V, E)$, source $s \in V$, sink $t \in V$, edge capacities $c_e \in \mathbb{Z}_{>0}$, $\forall e \in E$
- $s$ has no incoming edges, $t$ has no outgoing edges
Def. A \textit{s-t flow} is a vector $f \in \mathbb{R}^E_{\geq 0}$ satisfying the following conditions:

- $\forall e \in E, 0 \leq f_e \leq c_e$ \ (capacity constraints)
- $\forall v \in V \setminus \{s, t\}$,
  $$\sum_{e \in \delta^{\text{in}}(v)} f_e = \sum_{e \in \delta^{\text{out}}(v)} f_e$$ \ (flow conservation)

The value of flow $f$ is defined as:

$$\text{val}(f) := \sum_{e \in \delta^{\text{out}}(s)} f_e = \sum_{e \in \delta^{\text{in}}(t)} f_e$$
Maximum Flow Problem

**Input:** flow network \((G = (V, E), c, s, t)\)

**Output:** maximum value of a \(s-t\) flow \(f\)

---

**Ford-Fulkerson method**

**Maximum-Flow Min-Cut Theorem:** value of the maximum flow is equal to the value of the minimum \(s-t\) cut

[Chen-Kyng-Liu-Peng-Gutenberg-Sachdeva, 2022]: nearly linear-time algorithm
Maximum Flow Problem

**Input:** flow network \((G = (V, E), c, s, t)\)

**Output:** maximum value of a \(s-t\) flow \(f\)
Maximum Flow Problem

**Input:** flow network \((G = (V, E), c, s, t)\)

**Output:** maximum value of a \(s-t\) flow \(f\)

- Ford-Fulkerson method
Maximum Flow Problem

**Input:** flow network \((G = (V, E), c, s, t)\)

**Output:** maximum value of a \(s-t\) flow \(f\)

- **Ford-Fulkerson method**
- **Maximum-Flow Min-Cut Theorem:** value of the maximum flow is equal to the value of the minimum \(s-t\) cut
**Maximum Flow Problem**

**Input:** flow network \((G = (V, E), c, s, t)\)

**Output:** maximum value of a \(s-t\) flow \(f\)

- Ford-Fulkerson method
- **Maximum-Flow Min-Cut Theorem:** value of the maximum flow is equal to the value of the minimum \(s-t\) cut
- [Chen-Kyng-Liu-Peng-Gutenberg-Sachdeva, 2022]: nearly linear-time algorithm
LP for Maximum Flow

\[
\begin{align*}
\text{max} & \quad \sum_{e \in \delta^{\text{in}}(t)} x_e \\
\sum_{e \in \delta^{\text{out}}(v)} x_e - & \sum_{e \in \delta^{\text{in}}(v)} x_e = 0 \quad \forall v \in V \setminus \{s, t\} \\
\sum_{e \in E} x_e & \geq 0 \quad \forall e \in E \\
x_e & \leq c_e \quad \forall e \in E
\end{align*}
\]
LP for Maximum Flow

\[
\begin{align*}
\text{max} & \quad \sum_{e \in \delta^{\text{in}}(t)} x_e \\
\sum_{e \in \delta^{\text{out}}(v)} x_e - \sum_{e \in \delta^{\text{in}}(v)} x_e & = 0 \quad \forall v \in V \setminus \{s, t\} \\
\sum_{e \in \delta^{\text{out}}(v)} x_e - \sum_{e \in \delta^{\text{in}}(v)} x_e & = 0 \quad \forall v \in V \setminus \{s, t\} \\
x_e & \geq 0 \quad \forall e \in E \\
x_e & \leq c_e \quad \forall e \in E
\end{align*}
\]

**Theorem** The LP polytope is integral.
LP for Maximum Flow

\[
\max \sum_{e \in \delta^{\text{in}}(t)} x_e \\
\sum_{e \in \delta^{\text{out}}(v)} x_e - \sum_{e \in \delta^{\text{in}}(v)} x_e = 0 \quad \forall v \in V \setminus \{s, t\} \\
x_e \geq 0 \quad \forall e \in E
\]

Theorem  The LP polytope is integral.

Sketch of Proof.

- Take any \(s-t\) flow \(x\); consider fractional edges \(E'\)
- Every \(v \notin \{s, t\}\) must be incident to 0 or \(\geq 2\) edges in \(E'\)
- Ignoring the directions of \(E'\), it contains a cycle, or a \(s-t\) path
- We can increase/decrease flow values along cycle/path
Outline

1. Linear Programming and Rounding

2. Exact Algorithms Using LP: Integral Polytopes
   - Bipartite Matching Polytope
   - $s$-$t$ Flow Polytope
   - Weighted Interval Scheduling Problem

3. Approximation Algorithms Using LP: LP Rounding
   - 2-Approximation Algorithm for Weighted Vertex Cover
   - 2-Approximation Algorithm for Unrelated Machine Scheduling
Weighted Interval Scheduling Problem

**Input:** \( n \) activities, activity \( i \) starts at time \( s_i \), finishes at time \( f_i \), and has weight \( w_i > 0 \)

\( i \) and \( j \) can be scheduled together iff \([s_i, f_i)\) and \([s_j, f_j)\) are disjoint

**Output:** maximum weight subset of jobs that can be scheduled

\[ \begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
100 & & & & 50 & & & & 30 & \\
25 & & & & & & & & 50 & \\
90 & & & & & & & & & \\
80 & & & & & & & & & \\
80 & & & & & & & & & \\
\end{array} \]

- optimum value = 220
Weighted Interval Scheduling Problem

**Input:** $n$ activities, activity $i$ starts at time $s_i$, finishes at time $f_i$, and has weight $w_i > 0$

$i$ and $j$ can be scheduled together iff $[s_i, f_i)$ and $[s_j, f_j)$ are disjoint

**Output:** maximum weight subset of jobs that can be scheduled

- optimum value $= 220$
- Classic Problem for Dynamic Programming
**Weighted Interval Scheduling Problem**

**Linear Program**

\[
\begin{align*}
\text{max} & \quad \sum_{j \in [n]} x_j w_j \\
\sum_{j \in [n]: t \in [s_j, f_j]} x_j & \leq 1 \quad \forall t \in [T] \\
x_j & \geq 0 \quad \forall j \in [n]
\end{align*}
\]

Theorem

The LP polytope is integral.

Def.

A matrix \( A \in \mathbb{R}^{m \times n} \) is said to be totally unimodular (TUM), if every sub-square of \( A \) has determinant in \( \{-1, 0, 1\} \).

Theorem

If a polytope \( P \) is defined by \( Ax \geq b, x \geq 0 \) with a totally unimodular matrix \( A \) and integral \( b \), then \( P \) is integral.

Lemma

A matrix \( A \in \{0, 1\}^{m \times n} \) where the 1’s on every column form an interval is TUM.

So, the matrix for the LP is TUM, and the polytope is integral.
Weighted Interval Scheduling Problem

**Theorem** The LP polytope is integral.

**Linear Program**

\[
\begin{align*}
\text{max} & \quad \sum_{j \in [n]} x_j w_j \\
\sum_{j \in [n]: t \in [s_j, f_j]} x_j & \leq 1 \quad \forall t \in [T] \\
x_j & \geq 0 \quad \forall j \in [n]
\end{align*}
\]
Weighted Interval Scheduling Problem

**Linear Program**

\[
\begin{align*}
\text{max} & \quad \sum_{j \in [n]} x_j w_j \\
\sum_{j \in [n]: t \in [s_j, f_j]} x_j & \leq 1 \quad \forall t \in [T] \\
x_j & \geq 0 \quad \forall j \in [n]
\end{align*}
\]

**Theorem** The LP polytope is integral.

**Def.** A matrix \( A \in \mathbb{R}^{m \times n} \) is said to be totally unimodular (TUM), if every sub-square of \( A \) has determinant in \( \{-1, 0, 1\} \).
Weighted Interval Scheduling Problem

**Linear Program**

\[
\begin{align*}
\text{max} & \quad \sum_{j \in [n]} x_j w_j \\
\sum_{j \in [n]: t \in [s_j, f_j]} x_j & \leq 1 \quad \forall t \in [T] \\
x_j & \geq 0 \quad \forall j \in [n]
\end{align*}
\]

**Theorem** The LP polytope is integral.

**Def.** A matrix \( A \in \mathbb{R}^{m \times n} \) is said to be totally unimodular (TUM), if every sub-square of \( A \) has determinant in \( \{-1, 0, 1\} \).

**Theorem** If a polytope \( P \) is defined by \( Ax \geq b, x \geq 0 \) with a totally unimodular matrix \( A \) and integral \( b \), then \( P \) is integral.
Linear Program

\[
\text{max } \sum_{j \in [n]} x_j w_j \\
\sum_{j \in [n]: t \in [s_j, f_j]} x_j \leq 1 \quad \forall t \in [T] \\
x_j \geq 0 \quad \forall j \in [n]
\]

**Theorem** The LP polytope is integral.

**Def.** A matrix \( A \in \mathbb{R}^{m \times n} \) is said to be totally unimodular (TUM), if every sub-square of \( A \) has determinant in \( \{-1, 0, 1\} \).

**Theorem** If a polytope \( P \) is defined by \( Ax \geq b, x \geq 0 \) with a totally unimodular matrix \( A \) and integral \( b \), then \( P \) is integral.

**Lemma** A matrix \( A \in \{0, 1\}^{m \times n} \) where the 1’s on every column form an interval is TUM.

- So, the matrix for the LP is TUM, and the polytope is integral.
**Theorem**  If a polytope $P$ is defined by $Ax \geq b, x \geq 0$ with a totally unimodular matrix $A$ and integral $b$, then $P$ is integral.
Theorem  If a polytope \( P \) is defined by \( Ax \geq b, x \geq 0 \) with a totally unimodular matrix \( A \) and integral \( b \), then \( P \) is integral.

Proof.

- Every vertex \( x \in P \) is the unique solution to the linear system (after permuting coordinates): 
  \[
  \begin{pmatrix}
  A' & 0 \\
  0 & I
  \end{pmatrix}
  \begin{pmatrix}
  x_1 \\
  x_2
  \end{pmatrix}
  =
  \begin{pmatrix}
  b' \\
  0
  \end{pmatrix},
  \]
  where
- \( A' \) is a square submatrix of \( A \) with \( \det(A') = \pm 1 \), \( b' \) is a sub-vector of \( b \),
- and the rows for \( b' \) are the same as the rows for \( A' \).
Theorem  If a polytope $P$ is defined by $Ax \geq b$, $x \geq 0$ with a totally unimodular matrix $A$ and integral $b$, then $P$ is integral.

Proof.

- Every vertex $x \in P$ is the unique solution to the linear system (after permuting coordinates): $$\begin{pmatrix} A' & 0 \\ 0 & I \end{pmatrix} x = \begin{pmatrix} b' \\ 0 \end{pmatrix},$$ where
  - $A'$ is a square submatrix of $A$ with $\det(A') = \pm 1$, $b'$ is a sub-vector of $b$,
  - and the rows for $b'$ are the same as the rows for $A'$.
- Let $x = \begin{pmatrix} x^1 \\ x^2 \end{pmatrix}$, so that $A'x^1 = b'$ and $x^2 = 0$. 
**Theorem**  If a polytope $P$ is defined by $Ax \geq b, x \geq 0$ with a totally unimodular matrix $A$ and integral $b$, then $P$ is integral.

**Proof.**

- Every vertex $x \in P$ is the unique solution to the linear system (after permuting coordinates): \[
\begin{pmatrix}
A' & 0 \\
0 & I
\end{pmatrix} x = \begin{pmatrix} b' \\ 0 \end{pmatrix},
\]
where $A'$ is a square submatrix of $A$ with $\det(A') = \pm 1$, $b'$ is a sub-vector of $b$, and the rows for $b'$ are the same as the rows for $A'$.

- Let $x = \begin{pmatrix} x^1 \\ x^2 \end{pmatrix}$, so that $A'x^1 = b'$ and $x^2 = 0$.

- Cramer's rule: $x^1_i = \frac{\det(A'_i|b)}{\det(A')} \quad \text{for every } i \implies x^1_i \text{ is integer}$

$A'_i|b$: the matrix of $A'$ with the $i$-th column replaced by $b$
Example for the Proof

\[
\begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\
a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\
a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5}
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5
\end{pmatrix}
\geq
\begin{pmatrix}
b_1 \\
b_2 \\
b_3
\end{pmatrix}
\]

\[x_1, x_2, x_3, x_4, x_5 \geq 0\]
Example for the Proof

\[
\begin{pmatrix}
  a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\
  a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\
  a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5}
\end{pmatrix}
\begin{pmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5
\end{pmatrix}
\geq
\begin{pmatrix}
  b_1 \\
  b_2 \\
  b_3
\end{pmatrix}
\]

\[x_1, x_2, x_3, x_4, x_5 \geq 0\]

The following equation system may give a vertex:

\[
\begin{pmatrix}
  a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\
  a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\
  1 & 0 & 0 & 0 & 0 \\
  0 & 0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5
\end{pmatrix}
= \begin{pmatrix}
  b_1 \\
  b_3 \\
  0 \\
  0 \\
  0
\end{pmatrix}
\]
Example for the Proof

\[
\begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\
a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
b_3 \\
0 \\
0 \\
0 \\
\end{pmatrix}
\]
Example for the Proof

\[
\begin{pmatrix}
 a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\
 a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\
 1 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
 x_1 \\
 x_2 \\
 x_3 \\
 x_4 \\
 x_5
\end{pmatrix}
=
\begin{pmatrix}
 b_1 \\
 b_3 \\
 0 \\
 0 \\
 0
\end{pmatrix}
\]

Equivalently, the vertex satisfies

\[
\begin{pmatrix}
 a_{1,2} & a_{1,3} & 0 & 0 & 0 \\
 a_{3,2} & a_{3,3} & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
 x_2 \\
 x_3 \\
 x_1 \\
 x_4 \\
 x_5
\end{pmatrix}
=
\begin{pmatrix}
 b_1 \\
 b_3 \\
 0 \\
 0 \\
 0
\end{pmatrix}
\]
Lemma  Let $A' \in \{0, \pm 1\}^{n \times n}$ such that every row of $A'$ contains at most one 1 and one $-1$. Then $\det(A') \in \{0, \pm 1\}$.

Proof.
Lemma  Let $A' \in \{0, \pm 1\}^{n \times n}$ such that every row of $A'$ contains at most one 1 and one $-1$. Then $\det(A') \in \{0, \pm 1\}$.

Proof.

- wlog assume every row of $A'$ contains one 1 and one $-1$
Lemma Let $A' \in \{0, \pm 1\}^{n \times n}$ such that every row of $A'$ contains at most one 1 and one $-1$. Then $\det(A') \in \{0, \pm 1\}$.

Proof.

- wlog assume every row of $A'$ contains one 1 and one $-1$
- otherwise, we can reduce the matrix
**Lemma**  Let $A' \in \{0, \pm 1\}^{n \times n}$ such that every row of $A'$ contains at most one $1$ and one $-1$. Then $\det(A') \in \{0, \pm 1\}$.

**Proof.**
- wlog assume every row of $A'$ contains one $1$ and one $-1$
- otherwise, we can reduce the matrix
- treat $A'$ as a directed graph: columns $\equiv$ vertices, rows $\equiv$ arcs
**Lemma**  Let $A' \in \{0, \pm 1\}^{n \times n}$ such that every row of $A'$ contains at most one 1 and one $-1$. Then $\det(A') \in \{0, \pm 1\}$.

**Proof.**

- wlog assume every row of $A'$ contains one 1 and one $-1$
- otherwise, we can reduce the matrix
- treat $A'$ as a directed graph: columns $\equiv$ vertices, rows $\equiv$ arcs
- $\#\text{edges} = \#\text{vertices} \implies$ underlying undirected graph contains a cycle $\implies \det(A') = 0$
**Lemma** Let $A' \in \{0, \pm 1\}^{n \times n}$ such that every row of $A'$ contains at most one $1$ and one $-1$. Then $\det(A') \in \{0, \pm 1\}$.

**Proof.**
- wlog assume every row of $A'$ contains one $1$ and one $-1$
- otherwise, we can reduce the matrix
- treat $A'$ as a directed graph: columns $\equiv$ vertices, rows $\equiv$ arcs
- $\#\text{edges} = \#\text{vertices} \implies$ underlying undirected graph contains a cycle $\implies \det(A') = 0$

**Lemma** Let $A \in \{0, \pm 1\}^{m \times n}$ such that every row of $A$ contains at most one $1$ and one $-1$. Then $A$ is TUM.
**Lemma** Let $A' \in \{0, \pm 1\}^{n \times n}$ such that every row of $A'$ contains at most one 1 and one $-1$. Then $\det(A') \in \{0, \pm 1\}$.

**Proof.**

- wlog assume every row of $A'$ contains one 1 and one $-1$
- otherwise, we can reduce the matrix
- treat $A'$ as a directed graph: columns $\equiv$ vertices, rows $\equiv$ arcs
- $\#\text{edges} = \#\text{vertices} \implies$ underlying undirected graph contains a cycle $\implies \det(A') = 0$

**Lemma** Let $A \in \{0, \pm 1\}^{m \times n}$ such that every row of $A$ contains at most one 1 and one $-1$. Then $A$ is TUM.

**Coro.** The matrix for $s$-$t$ flow polytope is TUM; thus, the polytope is integral.
Example for the Proof

\[
\begin{pmatrix}
1 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 1 \\
1 & 0 & 0 & 0 & -1 & 0 & 0 \\
\end{pmatrix}
\]
Example for the Proof

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

$$(1 - 1 0 0 0 0) - (0 - 1 1 0 0) + (0 0 1 - 1 0) - (1 0 0 - 1 0) = (0 0 0 0 0)$$
Example for the Proof

\[
\begin{pmatrix}
1 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & -1 & 0 & 1 & 0 \\
1 & 0 & 0 & -1 & 0 & 0 & 0
\end{pmatrix}
\]
Example for the Proof

\[
\begin{pmatrix}
1 & -1 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & -1 & 0 & 1 \\
1 & 0 & 0 & -1 & 0 & 0 \\
\end{pmatrix}
\]
Example for the Proof

\[
\begin{pmatrix}
1 & -1 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & -1 & 1 \\
1 & 0 & 0 & -1 & 0
\end{pmatrix}
\]

\[\pmatrix{1 \ -1 \ 0 \ 0 \ 0 \\ 0 \ -1 \ 1 \ 0 \ 0 \\ 0 \ 0 \ 1 \ -1 \ 0 \\ 0 \ 0 \ 0 \ -1 \ 1 \\ 1 \ 0 \ 0 \ -1 \ 0} \]

= \begin{pmatrix}
0 & 0 & 0 & 0 & 0
\end{pmatrix}
Example for the Proof

\[
\begin{pmatrix}
1 & -1 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & -1 & 1 \\
1 & 0 & 0 & -1 & 0
\end{pmatrix}
\]

\[
\begin{align*}
(1 - 1 0 0 0) \\
(0 - 1 1 0 0) \\
(0 0 1 -1 0) \\
(0 0 0 -1 1) \\
(1 0 0 -1 0)
\end{align*}
\]
Example for the Proof

\[
\begin{pmatrix}
1 & -1 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & -1 & 1 \\
1 & 0 & 0 & -1 & 0
\end{pmatrix}
\]

\[
\begin{array}{c}
1 \\
2 \\
3 \\
4 \\
5
\end{array}
\]

\[
(1 - (1 - 1 0 0 0)) - (0 - (0 - 1 1 0 0)) + (0 0 1 - (1 0)) - (1 0 0 - (1 0)) = (0 0 0 0 0)
\]
Example for the Proof

\[
\begin{pmatrix}
1 & -1 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & -1 & 1 \\
1 & 0 & 0 & -1 & 0
\end{pmatrix}
\]

\[
+ (1 \ -1 \ 0 \ 0 \ 0) \\
- (0 \ -1 \ 1 \ 0 \ 0) \\
+ (0 \ 0 \ 1 \ -1 \ 0) \\
- (1 \ 0 \ 0 \ -1 \ 0)
\]

\[
= (0 \ 0 \ 0 \ 0 \ 0 \ 0)
\]
**Lemma** A matrix \( A \in \{0, 1\}^{m \times n} \) where the 1’s on every row form an interval is TUM.

**Proof.**
Lemma  A matrix $A \in \{0, 1\}^{m \times n}$ where the 1’s on every row form an interval is TUM.

Proof.

- take any square submatrix $A'$ of $A$, 

\[
\begin{pmatrix}
1 & -1 & 0 & \cdots & 0 \\
0 & 1 & -1 & \cdots & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & \cdots & 1 & -1 \\
0 & 0 & \cdots & 0 & 1
\end{pmatrix}
\]

$\det(M) = 1$.

$\det(A'M) \in \{0, \pm 1\} \Rightarrow \det(A') \in \{0, \pm 1\}$.
Lemma  A matrix $A \in \{0, 1\}^{m \times n}$ where the 1’s on every row form an interval is TUM.

Proof.

- take any square submatrix $A'$ of $A$,
- the 1’s on every row of $A'$ form an interval.
Lemma: A matrix $A \in \{0, 1\}^{m \times n}$ where the 1’s on every row form an interval is TUM.

Proof.

- take any square submatrix $A'$ of $A$,
- the 1’s on every row of $A'$ form an interval.
- $A'M$ is a matrix satisfying condition of first lemma, where
  $$M = \begin{pmatrix}
    1 & -1 & 0 & \cdots & 0 \\
    0 & 1 & -1 & \cdots & 0 \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    0 & 0 & \cdots & 1 & -1 \\
    0 & 0 & \cdots & 0 & 1
  \end{pmatrix}. \quad \det(M) = 1.$$
**Lemma** A matrix $A \in \{0, 1\}^{m \times n}$ where the 1’s on every row form an interval is TUM.

**Proof.**

- take any square submatrix $A'$ of $A$,
- the 1’s on every row of $A'$ form an interval.
- $A'M$ is a matrix satisfying condition of first lemma, where

$$M = \begin{pmatrix}
1 & -1 & 0 & \ldots & 0 \\
0 & 1 & -1 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 1 & -1 \\
0 & 0 & \ldots & 0 & 1
\end{pmatrix}.$$  

$$\det(M) = 1.$$  

- $\det(A'M) \in \{0, \pm 1\} \implies \det(A') \in \{0, \pm 1\}.$
Example for the Proof

\[
\begin{pmatrix}
0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
0 & 1 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}
\]
every row has at most one 1, at most one −1.
Example for the Proof

\[
\begin{pmatrix}
0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1
\end{pmatrix}
\]

every row has at most one 1, at most one -1
Example for the Proof

$$
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 \\
\end{bmatrix}
$$

every row has at most one 1, at most one $-1$
Example for the Proof

\[
\begin{pmatrix}
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 & 1 & 0 & 0 \\
−1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0
\end{pmatrix}
\]

- (col 1, col 2 − col 1, col 3 − col 2, col 4 − col 3, col 5 − col 4)
Example for the Proof

\[
\begin{pmatrix}
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 \\
\end{pmatrix}
\quad \Rightarrow 
\begin{pmatrix}
0 & 1 & 0 & 0 & -1 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 \\
\end{pmatrix}
\]

- \((\text{col 1, col 2} - \text{col 1, col 3} - \text{col 2, col 4} - \text{col 3, col 5} - \text{col 4})\)
Example for the Proof

\[
\begin{pmatrix}
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
0 & 1 & 0 & 0 & -1 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0
\end{pmatrix}
\]

- \((\text{col } 1, \text{col } 2 - \text{col } 1, \text{col } 3 - \text{col } 2, \text{col } 4 - \text{col } 3, \text{col } 5 - \text{col } 4)\)
- every row has at most one 1, at most one \(-1\)
Lemma  The edge-vertex incidence matrix $A$ of a bipartite graph is totally-unimodular.

Proof.

Example
**Lemma**  The edge-vertex incidence matrix $A$ of a bipartite graph is totally-unimodular.

**Proof.**

- $G = (L \cup R, E)$: the bipartite graph

**Example**

$$
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 \\
\end{bmatrix}
$$
Lemma The edge-vertex incidence matrix $A$ of a bipartite graph is totally-unimodular.

Proof.

- $G = (L \cup R, E)$: the bipartite graph
- $A'$: obtained from $A$ by negating columns correspondent to $R$

Example
Lemma  The edge-vertex incidence matrix $A$ of a bipartite graph is totally-unimodular.

Proof.

- $G = (L \cup R, E)$: the bipartite graph
- $A'$: obtained from $A$ by negating columns correspondent to $R$
- each row of $A'$ has exactly one $+1$, and exactly one $-1$

Example
Lemma  The edge-vertex incidence matrix $A$ of a bipartite graph is totally-unimodular.

Proof.

- $G = (L \cup R, E)$: the bipartite graph
- $A'$: obtained from $A$ by negating columns correspondent to $R$
- each row of $A'$ has exactly one $+1$, and exactly one $-1$
- $\Rightarrow A'$ is TUM $\iff A$ is TUM

Example
Lemma  The edge-vertex incidence matrix $A$ of a bipartite graph is totally-unimodular.

Proof.

- $G = (L \cup R, E)$: the bipartite graph
- $A'$: obtained from $A$ by negating columns correspondent to $R$
- each row of $A'$ has exactly one $+1$, and exactly one $-1$
- $\implies A'$ is TUM $\iff A$ is TUM

Example

```
1 4
2 5
3 6
```
Lemma  The edge-vertex incidence matrix $A$ of a bipartite graph is totally-unimodular.

Proof.

- $G = (L \oplus R, E)$: the bipartite graph
- $A'$: obtained from $A$ by negating columns correspondent to $R$
- each row of $A'$ has exactly one $+1$, and exactly one $-1$
- $\Rightarrow A'$ is TUM $\iff A$ is TUM

Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\begin{pmatrix}
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 \\
\end{pmatrix}
\]
Lemma  The edge-vertex incidence matrix $A$ of a bipartite graph is totally-unimodular.

Proof.

- $G = (L \cup R, E)$: the bipartite graph
- $A'$: obtained from $A$ by negating columns correspondent to $R$
- each row of $A'$ has exactly one $+1$, and exactly one $-1$
- $\Rightarrow A'$ is TUM $\iff A$ is TUM

Example

- Graph:
  - Vertices: 1, 2, 3, 4, 5, 6
  - Edges: 1-4, 2-5, 3-6
- Incidence matrix:
  $\begin{pmatrix}
  1 & 0 & 0 & -1 & 0 & 0 \\
  1 & 0 & 0 & 0 & -1 & 0 \\
  1 & 0 & 0 & 0 & 0 & -1 \\
  1 & 0 & 0 & -1 & 0 & 0 \\
  1 & 0 & 0 & 0 & 0 & -1 \\
  1 & 0 & 0 & 0 & -1 & 0 \\
  \end{pmatrix}$
remark: bipartiteness is needed. The edge-vertex incidence matrix of a triangle has determinant 2.

\[
\begin{pmatrix}
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0
\end{pmatrix}
\]
remark: bipartiteness is needed. The edge-vertex incidence matrix

\[
\begin{pmatrix}
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0
\end{pmatrix}
\]

of a triangle has determinant 2.

**Coro.** Bipartite matching polytope is integral.
1. Linear Programming and Rounding

2. Exact Algorithms Using LP: Integral Polytopes
   - Bipartite Matching Polytope
   - s-t Flow Polytope
   - Weighted Interval Scheduling Problem

3. Approximation Algorithms Using LP: LP Rounding
   - 2-Approximation Algorithm for Weighted Vertex Cover
   - 2-Approximation Algorithm for Unrelated Machine Scheduling
Approximation Algorithm based on LP Rounding

Opti. Problem $X \iff 0/1$ Integer Program (IP) $\xrightarrow{\text{relax}}$ LP

- Integer programming is NP-hard, linear programming is in P
- Solve LP to obtain a fractional $x \in [0,1]^n$
- Round it to an integral $\tilde{x} \in \{0,1\}^n \iff \text{solution for } X$
- Prove $c^T \tilde{x} \leq \alpha \cdot c^T x$, then $c^T \tilde{x} \leq \alpha \cdot \text{LP} \leq \alpha \cdot \text{IP} = \alpha \cdot \text{opt} \Rightarrow \alpha$-approximation
Approximation Algorithm based on LP Rounding

- Opti. Problem $X \iff 0/1$ Integer Program (IP) $\xrightarrow{\text{relax}}$ LP

<table>
<thead>
<tr>
<th>0/1 Integer Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\min \ c^T x$</td>
</tr>
<tr>
<td>$Ax \geq b$</td>
</tr>
<tr>
<td>$x \in {0, 1}^n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linear Program Relaxation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\min \ c^T x$</td>
</tr>
<tr>
<td>$Ax \geq b$</td>
</tr>
<tr>
<td>$x \in [0, 1]^n$</td>
</tr>
</tbody>
</table>
Approximation Algorithm based on LP Rounding

- Opti. Problem $X \iff$ 0/1 Integer Program (IP) $\overset{\text{relax}}{\Rightarrow}$ LP

<table>
<thead>
<tr>
<th>0/1 Integer Program</th>
<th>Linear Program Relaxation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\min \ c^T x$</td>
<td>$\min \ c^T x$</td>
</tr>
<tr>
<td>$Ax \geq b$</td>
<td>$Ax \geq b$</td>
</tr>
<tr>
<td>$x \in {0, 1}^n$</td>
<td>$x \in [0, 1]^n$</td>
</tr>
</tbody>
</table>

- LP $\leq$ IP

Integer programming is NP-hard, linear programming is in P

Solve LP to obtain a fractional $x \in [0, 1]^n$.
Round it to an integral $\tilde{x} \in \{0, 1\}^n \iff$ solution for $X$

Prove $c^T \tilde{x} \leq \alpha \cdot c^T x$, then $c^T \cdot \tilde{x} \leq \alpha \cdot LP \leq \alpha \cdot IP = \alpha \cdot \text{opt} \Rightarrow \alpha$-approximation
Approximation Algorithm based on LP Rounding

- Opti. Problem $X \iff 0/1$ Integer Program (IP) $\xrightarrow{\text{relax}}$ LP

0/1 Integer Program

$$\min c^T x$$
$$Ax \geq b$$
$$x \in \{0, 1\}^n$$

Linear Program Relaxation

$$\min c^T x$$
$$Ax \geq b$$
$$x \in [0, 1]^n$$

- $\text{LP} \leq \text{IP}$
- Integer programming is NP-hard, linear programming is in P
Approximation Algorithm based on LP Rounding

- Opti. Problem $X \iff 0/1$ Integer Program (IP) $\xrightarrow{\text{relax}}$ LP

0/1 Integer Program
\[
\begin{align*}
\min & \quad c^T x \\
Ax & \geq b \\
x & \in \{0, 1\}^n
\end{align*}
\]

Linear Program Relaxation
\[
\begin{align*}
\min & \quad c^T x \\
Ax & \geq b \\
x & \in [0, 1]^n
\end{align*}
\]

- LP $\leq$ IP
- Integer programming is NP-hard, linear programming is in P
- Solve LP to obtain a fractional $x \in [0, 1]^n$. 

\[
LP \leq IP
\]

\[
\text{Integer programming is NP-hard, linear programming is in P}
\]

\[
\text{Solve LP to obtain a fractional } x \in [0, 1]^n.
\]
Approximation Algorithm based on LP Rounding

- Opti. Problem $X \iff 0/1$ Integer Program (IP) $\xrightarrow{\text{relax}}$ LP

0/1 Integer Program

\[
\begin{align*}
\text{min} & \quad c^T x \\
Ax & \geq b \\
x & \in \{0, 1\}^n
\end{align*}
\]

Linear Program Relaxation

\[
\begin{align*}
\text{min} & \quad c^T x \\
Ax & \geq b \\
x & \in [0, 1]^n
\end{align*}
\]

- LP $\leq$ IP
- Integer programming is NP-hard, linear programming is in P
- Solve LP to obtain a fractional $x \in [0, 1]^n$.
- Round it to an integral $\tilde{x} \in \{0, 1\}^n \iff$ solution for $X$
Approximation Algorithm based on LP Rounding

- Opti. Problem $X \iff 0/1$ Integer Program (IP) $\overset{\text{relax}}{\implies} \text{LP}$

0/1 Integer Program

\[
\begin{align*}
\min & \quad c^T x \\
Ax & \geq b \\
x & \in \{0, 1\}^n
\end{align*}
\]

Linear Program Relaxation

\[
\begin{align*}
\min & \quad c^T x \\
Ax & \geq b \\
x & \in [0, 1]^n
\end{align*}
\]

- LP $\leq$ IP
- Integer programming is NP-hard, linear programming is in P
- Solve LP to obtain a fractional $x \in [0, 1]^n$.
- Round it to an integral $\tilde{x} \in \{0, 1\}^n \iff$ solution for $X$
- Prove $c^T \tilde{x} \leq \alpha \cdot c^T x$, then $c^T \cdot \tilde{x} \leq \alpha \cdot \text{LP} \leq \alpha \cdot \text{IP} = \alpha \cdot \text{opt}$
Approximation Algorithm based on LP Rounding

- **Opti. Problem** $X \iff 0/1$ **Integer Program (IP)** $\xrightarrow{\text{relax}}$ **LP**

<table>
<thead>
<tr>
<th>0/1 Integer Program</th>
<th>Linear Program Relaxation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\min c^T x$</td>
<td>$\min c^T x$</td>
</tr>
<tr>
<td>$Ax \geq b$</td>
<td>$Ax \geq b$</td>
</tr>
<tr>
<td>$x \in {0, 1}^n$</td>
<td>$x \in [0, 1]^n$</td>
</tr>
</tbody>
</table>

- **LP \leq IP**
- Integer programming is NP-hard, linear programming is in P
- Solve LP to obtain a fractional $x \in [0, 1]^n$.
- **Round** it to an integral $\tilde{x} \in \{0, 1\}^n \iff$ solution for $X$
- **Prove** $c^T \tilde{x} \leq \alpha \cdot c^T x$, then $c^T \cdot \tilde{x} \leq \alpha \cdot \text{LP} \leq \alpha \cdot \text{IP} = \alpha \cdot \text{opt}$
- $\implies \alpha$-approximation
**IP**
\[
\begin{align*}
\text{min} & \quad c^T x \\
Ax & \geq b \\
x & \in \{0, 1\}^n
\end{align*}
\]

**LP Relaxation**
\[
\begin{align*}
\text{min} & \quad c^T x \\
Ax & \geq b \\
x & \in [0, 1]^n
\end{align*}
\]

The ratio between IP = opt and LP is called the integrality gap of the LP relaxation. The approximation ratio based on this analysis cannot be better than the worst integrality gap.
**IP**

\[
\begin{align*}
\min & \quad c^T x \\
Ax & \geq b \\
x & \in \{0, 1\}^n
\end{align*}
\]

**LP Relaxation**

\[
\begin{align*}
\min & \quad c^T x \\
Ax & \geq b \\
x & \in [0, 1]^n
\end{align*}
\]

**Def.** The ratio between \( IP = \text{opt} \) and LP is called the integrality gap of the LP relaxation.
**Def.** The ratio between $\text{IP} = \text{opt}$ and $\text{LP}$ is called the \textit{integrality gap} of the LP relaxation.
**IP**

\[
\begin{align*}
\min & \quad c^T x \\
Ax & \geq b \\
x & \in \{0, 1\}^n
\end{align*}
\]

**LP Relaxation**

\[
\begin{align*}
\min & \quad c^T x \\
Ax & \geq b \\
x & \in [0, 1]^n
\end{align*}
\]

**Def.** The ratio between \( IP = \text{opt} \) and LP is called the **integrality gap** of the LP relaxation.

- The approximation ratio based on this analysis can not be better than the worst integrality gap.
The ratio between $\text{IP} = \text{opt}$ and $\text{LP}$ is called the integrality gap of the LP relaxation.

The approximation ratio based on this analysis cannot be better than the worst integrality gap.
**IP**

\[
\begin{align*}
\text{min } & \quad c^T x \\
Ax & \geq b \\
x & \in \{0, 1\}^n
\end{align*}
\]

**LP Relaxation**

\[
\begin{align*}
\text{min } & \quad c^T x \\
Ax & \geq b \\
x & \in [0, 1]^n
\end{align*}
\]

**Def.** The ratio between \( IP = \text{opt} \) and LP is called the **integrality gap** of the LP relaxation.

- The approximation ratio based on this analysis cannot be better than the worst integrality gap.
Def. The ratio between $\text{IP} = \text{opt}$ and $\text{LP}$ is called the integrality gap of the LP relaxation.

- The approximation ratio based on this analysis cannot be better than the worst integrality gap.
Outline

1 Linear Programming and Rounding

2 Exact Algorithms Using LP: Integral Polytopes
   - Bipartite Matching Polytope
   - s-t Flow Polytope
   - Weighted Interval Scheduling Problem

3 Approximation Algorithms Using LP: LP Rounding
   - 2-Approximation Algorithm for Weighted Vertex Cover
   - 2-Approximation Algorithm for Unrelated Machine Scheduling
Weighted Vertex Cover Problem

**Input:** graph $G = (V, E)$, vertex weights $w \in \mathbb{Z}^V_{>0}$

**Output:** vertex cover $S$ of $G$, to minimize $\sum_{v \in S} w_v$
- $x_v \in \{0, 1\}, \forall v \in V$: indicate if we include $v$ in the vertex cover

<table>
<thead>
<tr>
<th>Integer Program</th>
<th>LP Relaxation</th>
</tr>
</thead>
<tbody>
<tr>
<td>min $\sum_{v \in V} w_v x_v$</td>
<td>min $\sum_{v \in V} w_v x_v$</td>
</tr>
<tr>
<td>$x_u + x_v \geq 1 \ \ \ \forall (u, v) \in E$</td>
<td>$x_u + x_v \geq 1 \ \ \ \forall (u, v) \in E$</td>
</tr>
<tr>
<td>$x_v \in {0, 1} \ \ \ \forall v \in V$</td>
<td>$x_v \in [0, 1] \ \ \ \forall v \in V$</td>
</tr>
</tbody>
</table>

$\text{IP} := \text{value of integer program}$

$\text{LP} := \text{value of linear program}$

$\text{LP} \leq \text{IP} = \text{opt}$
- $x_v \in \{0, 1\}, \forall v \in V$: indicate if we include $v$ in the vertex cover

### Integer Program

$$\min \sum_{v \in V} w_v x_v$$

$$x_u + x_v \geq 1 \quad \forall (u, v) \in E$$

$$x_v \in \{0, 1\} \quad \forall v \in V$$

### LP Relaxation

$$\min \sum_{v \in V} w_v x_v$$

$$x_u + x_v \geq 1 \quad \forall (u, v) \in E$$

$$x_v \in [0, 1] \quad \forall v \in V$$

- IP := value of integer program, LP := value of linear program

- LP $\leq$ IP $=$ opt
Rounding Algorithm

1: Solve LP to obtain solution $\{x_u^*\}_{u \in V}$
   ▷ So, LP = $\sum_{u \in V} w_u x_u^* \leq IP$
Rounding Algorithm

1: Solve LP to obtain solution \( \{x_u^*\}_{u \in V} \)

\[ \triangleright \text{So, LP} = \sum_{u \in V} w_u x_u^* \leq \text{IP} \]

2: return \( S := \{u \in V : x_u \geq 1/2\} \)
Rounding Algorithm

1: Solve LP to obtain solution $\{x_u^*\}_{u \in V}$
   $\triangleright$ So, LP $= \sum_{u \in V} w_u x_u^* \leq \text{IP}$

2: return $S := \{u \in V : x_u \geq 1/2\}$

Lemma $S$ is a vertex cover of $G$.

Proof.
Rounding Algorithm

1. Solve LP to obtain solution \( \{x_u^*\}_{u \in V} \)
   \[ \text{So, LP} = \sum_{u \in V} w_u x_u^* \leq \text{IP} \]

2. return \( S := \{ u \in V : x_u \geq 1/2 \} \)

Lemma \( S \) is a vertex cover of \( G \).

Proof.

- Consider any \((u, v) \in E\): we have \( x_u^* + x_v^* \geq 1 \)
Rounding Algorithm

1: Solve LP to obtain solution \( \{x_u^*\}_{u \in V} \)

\[ \text{\triangleright\ So, LP } = \sum_{u \in V} w_u x_u^* \leq \text{IP} \]

2: return \( S := \{u \in V : x_u \geq 1/2\} \)

Lemma \( S \) is a vertex cover of \( G \).

Proof.

- Consider any \((u, v) \in E\): we have \( x_u^* + x_v^* \geq 1 \)
- So, \( x_u^* \geq 1/2 \) or \( x_v^* \geq 1/2 \) \( \implies \) \( u \in S \) or \( v \in S \). \( \square \)
Rounding Algorithm

1: Solve LP to obtain solution \( \{x_u^*\}_{u \in V} \)
\[ \triangleright \text{So, LP} = \sum_{u \in V} w_u x_u^* \leq \text{IP} \]

2: return \( S := \{u \in V : x_u \geq 1/2\} \)

**Lemma** \( S \) is a vertex cover of \( G \).
Rounding Algorithm

1: Solve LP to obtain solution \( \{x_u^*\}_{u \in V} \)
\[ \triangleright \text{So, } \text{LP} = \sum_{u \in V} w_u x_u^* \leq \text{IP} \]

2: return \( S := \{ u \in V : x_u \geq 1/2 \} \)

**Lemma** \( S \) is a vertex cover of \( G \).

**Lemma** \( \text{cost}(S) := \sum_{u \in S} w_u \leq 2 \cdot \text{LP} \).
Rounding Algorithm

1: Solve LP to obtain solution \( \{x_u^*\}_{u \in V} \)
   \[ \triangleright \text{So, LP} = \sum_{u \in V} w_u x_u^* \leq \text{IP} \]
2: \textbf{return} \( S := \{u \in V : x_u \geq 1/2\} \)

\textbf{Lemma} \( S \) is a vertex cover of \( G \).

\textbf{Lemma} \( \text{cost}(S) := \sum_{u \in S} w_u \leq 2 \cdot \text{LP}. \)

\textbf{Proof}.

\[
\text{cost}(S) = \sum_{u \in S} w_u \leq \sum_{u \in S} w_u \cdot 2x_u^* = 2 \sum_{u \in S} w_u \cdot x_u^* \\
\leq 2 \sum_{u \in V} w_u \cdot x_u^* = 2 \cdot \text{LP}. \]

\[\square\]
Rounding Algorithm

1: Solve LP to obtain solution \( \{x_u^*\} \forall u \in V \)
   \[ \triangleright \text{So, LP} = \sum_{u \in V} w_u x_u^* \leq \text{IP} \]

2: return \( S := \{u \in V : x_u \geq 1/2\} \)

**Lemma** \( S \) is a vertex cover of \( G \).

**Lemma** \( \text{cost}(S) := \sum_{u \in S} w_u \leq 2 \cdot \text{LP} \).

**Theorem** The algorithm is a 2-approximation algorithm for weighted vertex cover.
Rounding Algorithm

1: Solve LP to obtain solution \( \{x_u^*\}_{u \in V} \)

\[ \triangleright \text{So, } \text{LP} = \sum_{u \in V} w_u x_u^* \leq \text{IP} \]

2: return \( S := \{u \in V : x_u \geq 1/2\} \)

Lemma \( S \) is a vertex cover of \( G \).

Lemma \( \text{cost}(S) := \sum_{u \in S} w_u \leq 2 \cdot \text{LP} \).

Theorem The algorithm is a 2-approximation algorithm for weighted vertex cover.

Proof.

\[ \text{cost}(S) \leq 2 \cdot \text{LP} \leq 2 \cdot \text{IP} = 2 \cdot (\text{optimum value}) \]
Outline

1. Linear Programming and Rounding

2. Exact Algorithms Using LP: Integral Polytopes
   - Bipartite Matching Polytope
   - $s$-$t$ Flow Polytope
   - Weighted Interval Scheduling Problem

3. Approximation Algorithms Using LP: LP Rounding
   - 2-Approximation Algorithm for Weighted Vertex Cover
   - 2-Approximation Algorithm for Unrelated Machine Scheduling
Unrelated Machine Scheduling

**Input:** $J, |J| = n$: jobs  
$M, |M| = m$: machines  
$p_{ij}$: processing time of job $j$ on machine $i$

**Output:** assignment $\sigma : J \mapsto M$, so as to minimize makespan:

$$\max_{i \in M} \sum_{j \in \sigma^{-1}(i)} p_{ij}$$
Unrelated Machine Scheduling

**Input:** \( J, |J| = n \): jobs
\( M, |M| = m \): machines
\( p_{ij} \): processing time of job \( j \) on machine \( i \)

**Output:** assignment \( \sigma : J \leftrightarrow M \), so as to minimize makespan:

\[
\max_{i \in M} \sum_{j \in \sigma^{-1}(i)} p_{ij}
\]
Unrelated Machine Scheduling

**Input:** $J, |J| = n$: jobs

$M, |M| = m$: machines

$p_{ij}$: processing time of job $j$ on machine $i$

**Output:** assignment $\sigma: J \mapsto M$, so as to minimize makespan:

$$\max_{i \in M} \sum_{j \in \sigma^{-1}(i)} p_{ij}$$
**Unrelated Machine Scheduling**

**Input:** $J, |J| = n$: jobs

$M, |M| = m$: machines

$p_{ij}$: processing time of job $j$ on machine $i$

**Output:** assignment $\sigma: J \mapsto M$, so as to minimize makespan:

$$\max_{i \in M} \sum_{j \in \sigma^{-1}(i)} p_{ij}$$
Assumption: we are given a target makespan $T$, and $p_{ij} \in [0, T] \cup \{\infty\}$
Assumption: we are given a target makespan $T$, and $p_{ij} \in [0, T] \cup \{\infty\}$

$x_{ij}$: fraction of $j$ assigned to $i$

\[
\sum_{i} x_{ij} = 1 \quad \forall j \in J
\]

\[
\sum_{j} p_{ij} x_{ij} \leq T \quad \forall i \in M
\]

$x_{ij} \geq 0 \quad \forall ij$
Assumption: we are given a target makespan $T$, and $p_{ij} \in [0, T] \cup \{\infty\}$

$x_{ij}$: fraction of $j$ assigned to $i$

$$\sum_{i} x_{ij} = 1 \quad \forall j \in J$$

$$\sum_{j} p_{ij} x_{ij} \leq T \quad \forall i \in M$$

$$x_{ij} \geq 0 \quad \forall i, j$$
2-Approximate Rounding Algorithm of Shmoys-Tardos

\[ x_{ij} \]

The algorithm involves matching jobs in set \( J \) with machines in set \( M \) such that the resulting configuration is a point in the bipartite-matching polytope, where all jobs in \( J \) are matched.
2-Approximate Rounding Algorithm of Shmoys-Tardos

\[ \sum_i x_{ij} = 1 \]

A bipartite graph with nodes labeled \( J \) and \( M \) connected by edges labeled \( x_{ij} \) represents the matching polytope, where all jobs in \( J \) are matched.
2-Approximate Rounding Algorithm of Shmoys-Tardos

\[ \sum_{i} x_{ij} = 1 \quad x_{ij} \quad \sum_{i} p_{ij} x_{ij} \leq T \]
2-Approximate Rounding Algorithm of Shmoys-Tardos

\[ p_{ij_1} \geq p_{ij_2} \geq \ldots \geq p_{ij_5} \]
2-Approximate Rounding Algorithm of Shmoys-Tardos

\[ p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_5} \]
2-Approximate Rounding Algorithm of Shmoys-Tardos

\[
p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_5}
\]
2-Approximate Rounding Algorithm of Shmoys-Tardos

\[ p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_5} \]

Segment of length 1

Obs. between \( J \) and sub-machines is a point in the bipartite-matching polytope, where all jobs in \( J \) are matched.
2-Approximate Rounding Algorithm of Shmoys-Tardos

\[ p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_5} \]
2-Approximate Rounding Algorithm of Shmoys-Tardos

\[ p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_5} \]
2-Approximate Rounding Algorithm of Shmoys-Tardos

\[ p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_5} \]
2-Approximate Rounding Algorithm of Shmoys-Tardos

\[
x_{ij}
\]

\(x_{ij}\) between \(J\) and sub-machines is a point in the bipartite-matching polytope, where all jobs in \(J\) are matched.

\(J\)  

\(M\)  

sub-machines
2-Approximate Rounding Algorithm of Shmoys-Tardos

Obs. $x_{ij}$ between $J$ and sub-machines is a point in the bipartite-matching polytope, where all jobs in $J$ are matched.

$$\sum_g x_{gj} = 1$$

$$\sum_j x_{gj} \leq 1$$
2-Approximate Rounding Algorithm of Shmoys-Tardos

\[ \sum_{g} x_{gj} = 1 \]
\[ \sum_{j} x_{gj} \leq 1 \]

\( x_{ij} \) between \( J \) and sub-machines is a point in the bipartite-matching polytope, where all jobs in \( J \) are matched.
Recall bipartite matching polytope is integral.
Recall bipartite matching polytope is integral.

$x$ is a convex combination of matchings.
Recall bipartite matching polytope is integral.

$x$ is a convex combination of matchings.

Any matching in the combination covers all jobs $J$. 
Recall bipartite matching polytope is integral.

$x$ is a convex combination of matchings.

Any matching in the combination covers all jobs $J$.

**Lemma**  Any matching in the combination gives an schedule of makespan $\leq 2T$. 
**Lemma** Any matching in the combination gives an schedule of makespan $\leq 2T$.

Proof.
**Lemma** Any matching in the combination gives an schedule of makespan $\leq 2T$.

Proof.
**Lemma** Any matching in the combination gives an schedule of makespan $\leq 2T$.

Proof.

- focus on machine $i$, let $i_1, i_2, \cdots, i_a$ be the sub-machines for $i$
Lemma  Any matching in the combination gives an schedule of makespan $\leq 2T$.

Proof.
- focus on machine $i$, let $i_1, i_2, \cdots, i_a$ be the sub-machines for $i$
- assume job $k_t$ is assigned to sub-machine $i_t$. 

- $p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_5}$
**Lemma** Any matching in the combination gives an schedule of makespan $\leq 2T$.

Proof.

- focus on machine $i$, let $i_1, i_2, \cdots, i_a$ be the sub-machines for $i$
- assume job $k_t$ is assigned to sub-machine $i_t$.

\[
(\text{load on } i) = \sum_{t=1}^{a} p_{ik_t} \leq p_{ik_1} + \sum_{t=2}^{a} \sum_{j} x_{i_{t-1}j} \cdot p_{ij} \\
\leq p_{ik_1} + \sum_{j} x_{ij} p_{ij} \leq T + T = 2T.
\]
- fix $i$, use $p_j$ for $p_{ij}$
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:
• fix $i$, use $p_j$ for $p_{ij}$
• $p_1 \geq p_2 \geq \cdots \geq p_7$
• worst case:
• fix $i$, use $p_j$ for $p_{ij}$
• $p_1 \geq p_2 \geq \cdots \geq p_7$
• worst case:
- fix $i$, use $p_j$ for $p_{ij}$
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:
- fix $i$, use $p_j$ for $p_{ij}$
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:
- fix $i$, use $p_j$ for $p_{ij}$
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:
• fix \( i \), use \( p_j \) for \( p_{ij} \)
• \( p_1 \geq p_2 \geq \cdots \geq p_7 \)
• worst case:
- fix $i$, use $p_j$ for $p_{ij}$
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:

![Graph with nodes and edges labeled with probabilities]
- fix $i$, use $p_j$ for $p_{ij}$
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:
- fix $i$, use $p_j$ for $p_{ij}$
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:
• fix $i$, use $p_j$ for $p_{ij}$
• $p_1 \geq p_2 \geq \cdots \geq p_7$
• worst case:
- fix $i$, use $p_j$ for $p_{ij}$
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:
- Fix $i$, use $p_j$ for $p_{ij}$
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- Worst case:
• fix $i$, use $p_j$ for $p_{ij}$

• $p_1 \geq p_2 \geq \cdots \geq p_7$

• worst case:
  • $1 \rightarrow i_1$, $2 \rightarrow i_2$
  • $4 \rightarrow i_3$, $7 \rightarrow i_4$
• fix $i$, use $p_j$ for $p_{ij}$
• $p_1 \geq p_2 \geq \cdots \geq p_7$
• worst case:
  • $1 \rightarrow i_1, 2 \rightarrow i_2$
  • $4 \rightarrow i_3, 7 \rightarrow i_4$
• fix \(i\), use \(p_j\) for \(p_{ij}\)

• \(p_1 \geq p_2 \geq \cdots \geq p_7\)

• worst case:
  • \(1 \rightarrow i1, 2 \rightarrow i2\)
  • \(4 \rightarrow i3, 7 \rightarrow i4\)

\[
\begin{align*}
p_1 & \leq T \\
p_2 & \leq 0.7p_1 + 0.3p_2 \\
p_4 & \leq 0.3p_2 + 0.5p_3 + 0.2p_4 \\
p_7 & \leq 0.1p_4 + 0.5p_5 + 0.2p_6 + 0.2p_7
\end{align*}
\]
fix $i$, use $p_j$ for $p_{ij}$

$p_1 \geq p_2 \geq \cdots \geq p_7$

worst case:

1 → $i_1$, 2 → $i_2$
4 → $i_3$, 7 → $i_4$

$p_1 \leq T$

$p_2 \leq 0.7p_1 + 0.3p_2$

$p_4 \leq 0.3p_2 + 0.5p_3 + 0.2p_4$

$p_7 \leq 0.1p_4 + 0.5p_5 + 0.2p_6 + 0.2p_7$

$p_1 + p_2 + p_4 + p_7 \leq T + (0.7p_1 + 0.3p_2) + (0.3p_2 + 0.5p_3 + 0.2p_4) + (0.1p_4 + 0.5p_5 + 0.2p_6 + 0.2p_7)$

$\leq T + (0.7p_1 + 0.6p_2 + 0.5p_3 + 0.3p_4 + 0.5p_5 + 0.2p_6 + 0.4p_7)$

$\leq T + T = 2T$
linear programming, simplex method, interior point method, ellipsoid method

integral LP polytopes: bipartite matching polytope, $s$-$t$ flow polytope, weighted interval scheduling polytope
Summary

- linear programming, simplex method, interior point method, ellipsoid method
- integral LP polytopes: bipartite matching polytope, $s$-$t$ flow polytope, weighted interval scheduling polytope
- approximation algorithm using LP rounding
  - 2-approximation algorithm for weighted vertex cover
  - 2-approximation for unrelated machine scheduling
Linear Program : 线性规划
Integer Program : 整数规划
Feasible Region : 解域
Polyhedron : 凸多面体
Polytope : 有界凸多面体
Vertex/Extreme Point : 顶点
Convex Combination : 凸组合
Convex Hull : 凸包
Dual : 对偶
Totally Unimodular : 完全单位模的