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

Algorithm Design Based on Linear Programming (LP)

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

Algorithm Design Based on Linear Programming (LP)

- Opti. Problem $X \Longleftrightarrow$ Integer Program (IP) $\xrightarrow{\text{relax}}$ LP
- Integer programming is NP-hard; linear programming is in P

Algorithm Design Based on Linear Programming (LP)

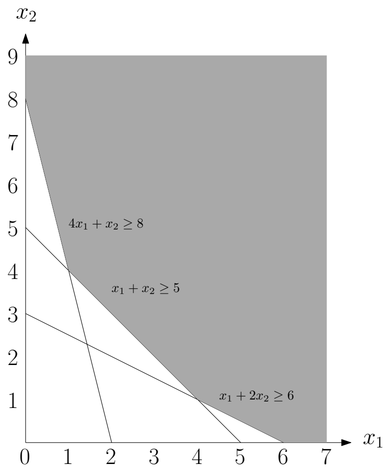
- Opti. Problem X $\Longleftrightarrow$ Integer Program (IP) $\xRightarrow{\text{relax}}$ LP
- Integer programming is NP-hard; linear programming is in P
- For some problems LP $\equiv$ IP $\Longrightarrow$ exact algorithms

Algorithm Design Based on Linear Programming (LP)

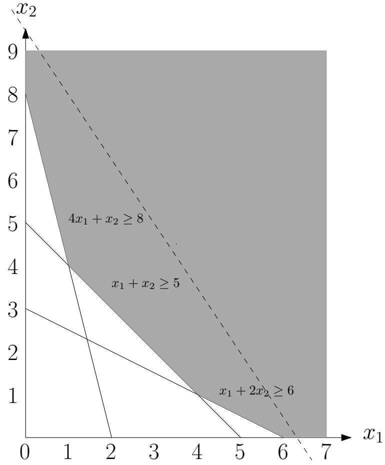
- Opti. Problem $X \Longleftrightarrow$ Integer Program (IP) $\xrightarrow{\text{relax}}$ LP
- Integer programming is NP-hard; linear programming is in P
- For some problems LP $\equiv$ IP $\Longrightarrow$ exact algorithms
- For some problems, LP $\not\equiv$ IP
  - solve LP to obtain a fractional solution,
  - round it to an integral solution

  $\Longrightarrow$ approximation algorithms

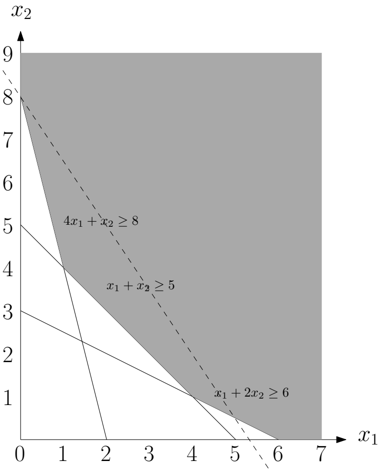
$$\begin{aligned} \min \quad & 7x_1 + 4x_2 \\ x_1 + x_2 &\geq 5 \\ x_1 + 2x_2 &\geq 6 \\ 4x_1 + x_2 &\geq 8 \\ x_1, x_2 &\geq 0 \end{aligned}$$



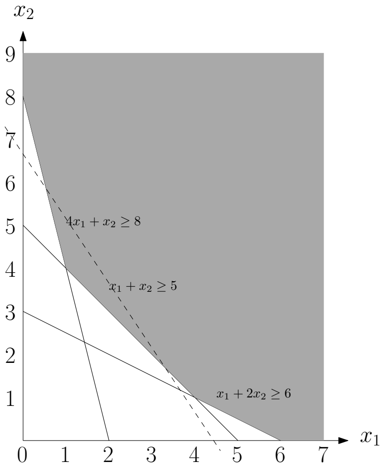
$$\begin{aligned} \min \quad & 7x_1 + 4x_2 \\ & x_1 + x_2 \geq 5 \\ & x_1 + 2x_2 \geq 6 \\ & 4x_1 + x_2 \geq 8 \\ & x_1, x_2 \geq 0 \end{aligned}$$



$$\begin{aligned} \min \quad & 7x_1 + 4x_2 \\ & x_1 + x_2 \geq 5 \\ & x_1 + 2x_2 \geq 6 \\ & 4x_1 + x_2 \geq 8 \\ & x_1, x_2 \geq 0 \end{aligned}$$

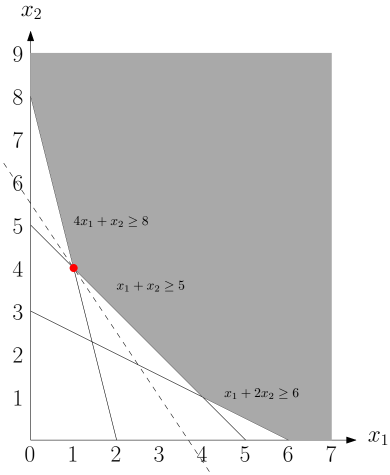


$$\begin{aligned} \min \quad & 7x_1 + 4x_2 \\ x_1 + x_2 &\geq 5 \\ x_1 + 2x_2 &\geq 6 \\ 4x_1 + x_2 &\geq 8 \\ x_1, x_2 &\geq 0 \end{aligned}$$



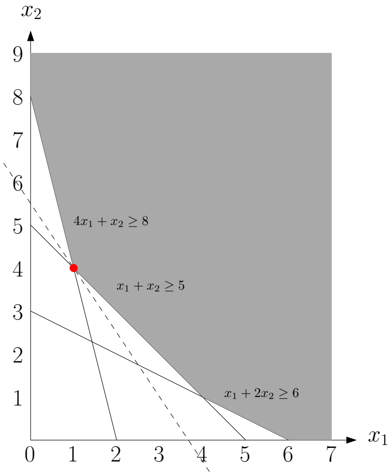
Linear Programming (LP), Linear Program (LP)

$$\begin{aligned} \min \quad & 7x_1 + 4x_2 \\ x_1 + x_2 &\geq 5 \\ x_1 + 2x_2 &\geq 6 \\ 4x_1 + x_2 &\geq 8 \\ x_1, x_2 &\geq 0 \end{aligned}$$



$$\begin{aligned} \min \quad & 7x_1 + 4x_2 \\ & x_1 + x_2 \geq 5 \\ & x_1 + 2x_2 \geq 6 \\ & 4x_1 + x_2 \geq 8 \\ & x_1, x_2 \geq 0 \end{aligned}$$

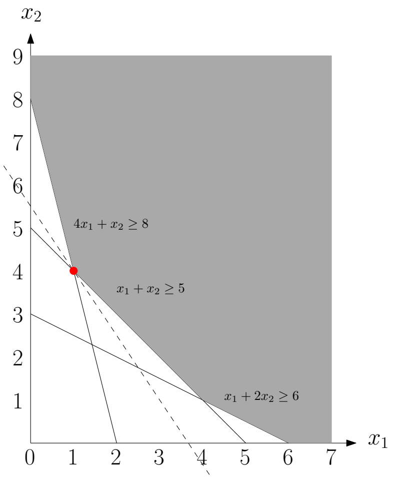
- optimum solution: $x_1 = 1, x_2 = 4$
- optimum value $= 7 \times 1 + 4 \times 4 = 23$



Linear Programming (LP), Linear Program (LP)

$$
\begin{aligned}
\min \quad & 7x_1 + 4x_2 \\
& x_1 + x_2 \geq 5 \\
& x_1 + 2x_2 \geq 6 \\
& 4x_1 + x_2 \geq 8 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

- 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



Standard Form of Linear Programs

$$
\begin{aligned}
\min \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &\geq b_1 \\
a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &\geq b_2 \\
\vdots \qquad \vdots \qquad \vdots \qquad \vdots \quad & \\
a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n &\geq b_m \\
x_1, x_2, \cdots, x_n &\geq 0
\end{aligned}
$$

- n: number of variables $\qquad$ m: number of constraints

Standard Form of Linear Programs

$$
\begin{aligned}
\min \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &\geq b_1 \\
a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &\geq b_2 \\
\vdots \qquad \vdots \qquad \vdots \qquad \vdots & \\
a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n &\geq b_m \\
x_1, x_2, \cdots, x_n &\geq 0
\end{aligned}
$$

- n: number of variables $\qquad$ m: number of constraints
- $\leq$ constraints? equlities?

Standard Form of Linear Programs

$$
\begin{aligned}
\min \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
& a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n \geq b_1 \\
& a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n \geq b_2 \\
& \quad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \\
& a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n \geq b_m \\
& x_1, x_2, \cdots, x_n \geq 0
\end{aligned}
$$

- n: number of variables $\quad$ m: number of constraints
- $\leq$ constraints? equlities?
- variables can be negative? maximization problem?

Standard Form of Linear Programs

$$x := \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n, \qquad c := \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} \in \mathbb{R}^n,$$

$$A := \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} \in \mathbb{R}^{n \times m}, \quad b := \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \in \mathbb{R}^m.$$

$$
\begin{aligned}
\min \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &\geq b_1 \\
a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &\geq b_2 \\
\vdots \qquad \vdots \qquad \vdots \qquad \vdots & \\
a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n &\geq b_m \\
x_1, x_2, \cdots, x_n &\geq 0
\end{aligned}
$$

Standard Form of Linear Program

$$
\begin{aligned}
\min \quad & c^{\mathrm{T}}x \\
Ax &\geq b \\
x &\geq 0
\end{aligned}
$$

- $\geq$: coordinate-wise less than or equal to

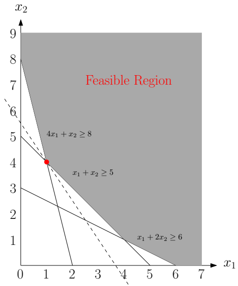
History

- [Fourier, 1827]: Fourier-Motzkin elimination method
- [Kantorovich, Koopmans 1939]: formulated the general linear programming problem

History

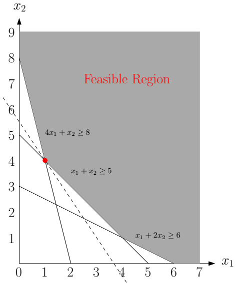
- [Fourier, 1827]: Fourier-Motzkin elimination method
- [Kantorovich, Koopmans 1939]: formulated the general linear programming problem
- [Dantzig 1946]: simplex method
- [Khachiyan 1979]: ellipsoid method, polynomial time, proved linear programming is in P
- [Karmarkar, 1984]: interior-point method, polynomial time, algorithm is pratical

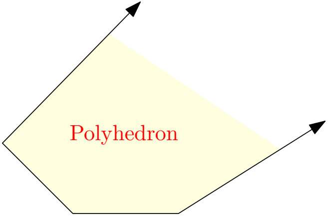
- feasible region: the set of x's satisfying $Ax \geq b, x \geq 0$



Preliminaries

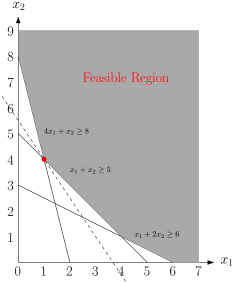
- feasible region: the set of x's satisfying $Ax \geq b, x \geq 0$
- feasible region is a polyhedron

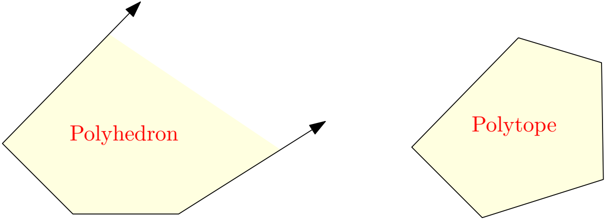




Preliminaries

- 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





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

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

Preliminaries

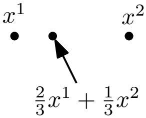
- x is a convex combination of $x^{(1)}, x^{(2)}, \cdots, x^{(t)}$ if the following condition holds: there exist $\lambda_1, \lambda_2, \cdots, \lambda_t \in [0,1]$ such that

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

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

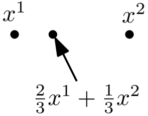
$$\lambda_1 + \lambda_2 + \cdots + \lambda_t = 1, \qquad \lambda_1 x^{(1)} + \lambda_2 x^{(2)} + \cdots + \lambda_t x^{(t)} = x$$

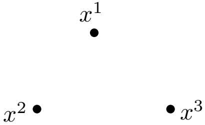


Preliminaries

- 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

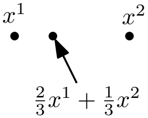
$$\lambda_1 + \lambda_2 + \cdots + \lambda_t = 1, \qquad \lambda_1 x^{(1)} + \lambda_2 x^{(2)} + \cdots + \lambda_t x^{(t)} = x$$

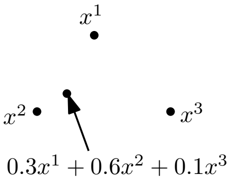




- x is a convex combination of $x^{(1)}, x^{(2)}, \cdots, x^{(t)}$ if the following condition holds: there exist $\lambda_1, \lambda_2, \cdots, \lambda_t \in [0, 1]$ such that

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



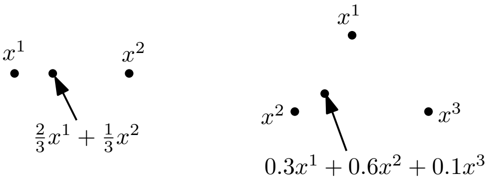


Preliminaries

- 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

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

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

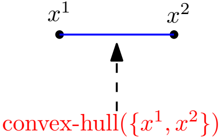


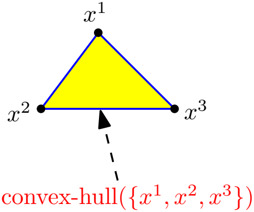
Preliminaries

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

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

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



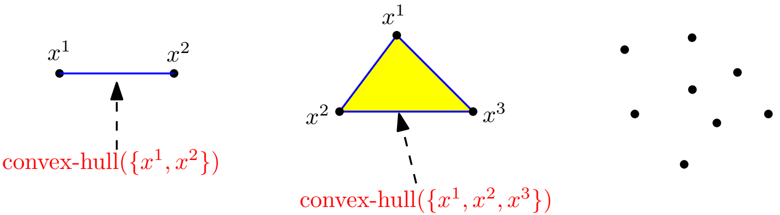


Preliminaries

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

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

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

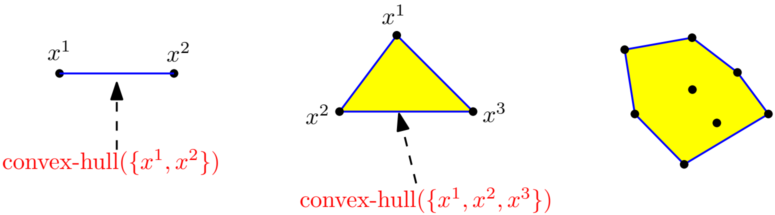


Preliminaries

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

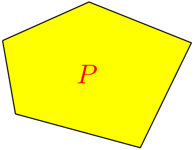
$$\lambda_1 + \lambda_2 + \cdots + \lambda_t = 1, \qquad \lambda_1 x^{(1)} + \lambda_2 x^{(2)} + \cdots + \lambda_t x^{(t)} = x$$

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



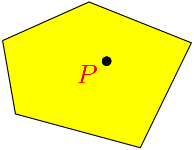
Preliminaries

- 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



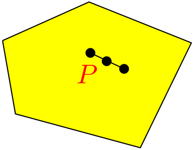
Preliminaries

- 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



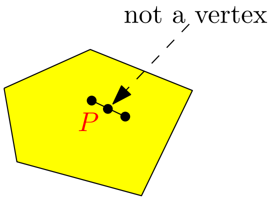
Preliminaries

- 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



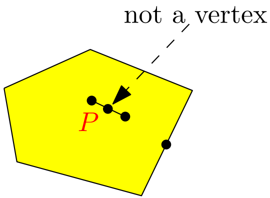
Preliminaries

- 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



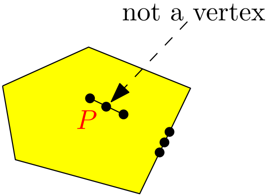
Preliminaries

- 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



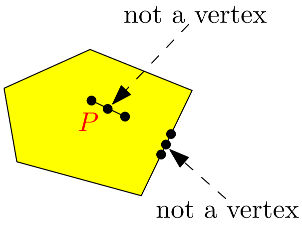
Preliminaries

- 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



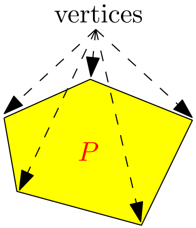
Preliminaries

- 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



Preliminaries

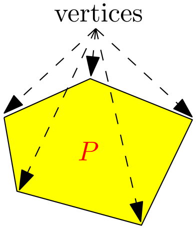
- let P be polytope, $x \in P$. If there are no other points $x', x'' \in P$ such that x is a convex combination of x' and x'', then x is called a vertex/extreme point of P



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

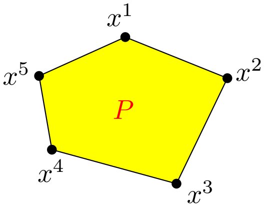
Lemma A polytope has finite number of vertices, and it is the convex hull of the vertices.



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

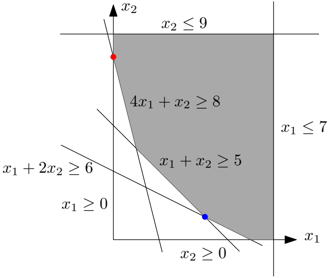
Lemma A polytope has finite number of vertices, and it is the convex hull of the vertices.



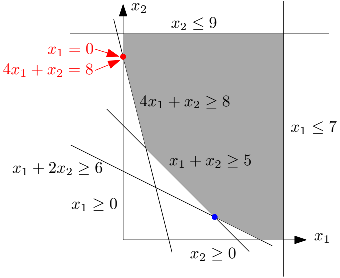
$$P = \text{convex-hull}(\{x^1, x^2, x^3, x^4, x^5\})$$

Preliminaries

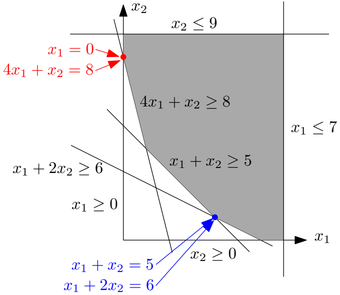
Lemma Let $x \in \mathbb{R}^n$ be an extreme point in a n-dimensional polytope. Then, there are n constraints in the definition of the polytope, such that x is the unique solution to the linear system obtained from the n constraints by replacing inequalities to equalities.



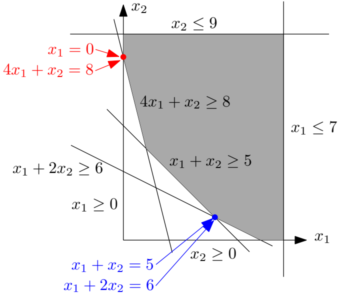
Lemma Let $x \in \mathbb{R}^n$ be an extreme point in a n-dimensional polytope. Then, there are n constraints in the definition of the polytope, such that x is the unique solution to the linear system obtained from the n constraints by replacing inequalities to equalities.



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.



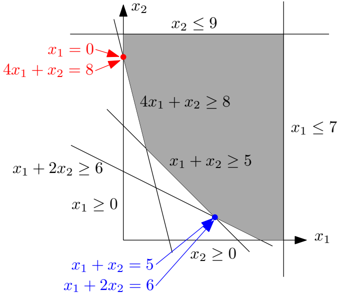
Lemma Let $x \in \mathbb{R}^n$ be an extreme point in a n-dimensional polytope. Then, there are n constraints in the definition of the polytope, such that x is the unique solution to the linear system obtained from the n constraints by replacing inequalities to equalities.



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

Preliminaries

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 opimum 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 ∞
- if the feasible region is unbounded, then its value can be $-\infty$

Algorithms for Linear Programming

algorithm	running time	practice
Simplex Method	exponential time	fast
Ellipsoid Method	polynomial time	slow
Interior Point Method	polynomial time	fast

Simplex Method

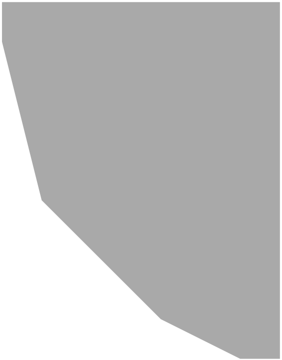
- [Dantzig, 1946]

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

Simplex Method

- [Dantzig, 1946]

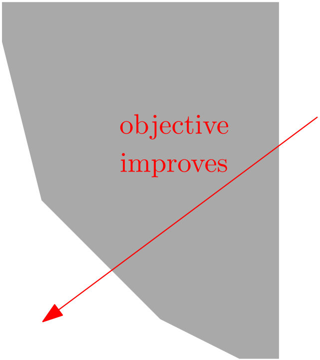
- move from one vertex to another, so as to improve the objective
- repeat until we reach an optimum vertex



Simplex Method

- [Dantzig, 1946]

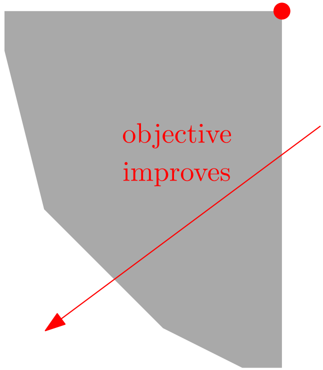
- move from one vertex to another, so as to improve the objective
- repeat until we reach an optimum vertex



Simplex Method

- [Dantzig, 1946]

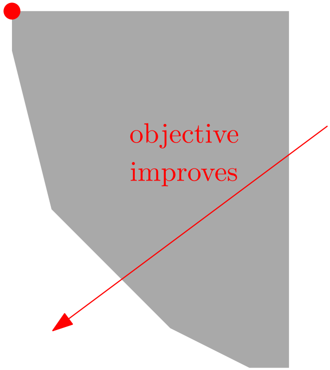
- move from one vertex to another, so as to improve the objective
- repeat until we reach an optimum vertex



Simplex Method

- [Dantzig, 1946]

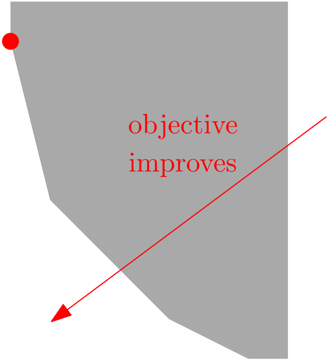
- move from one vertex to another, so as to improve the objective
- repeat until we reach an optimum vertex



Simplex Method

- [Dantzig, 1946]

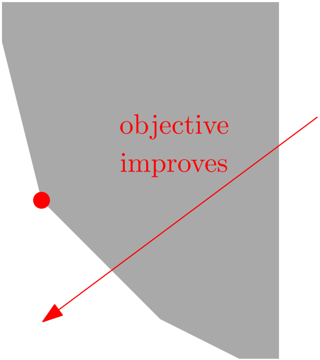
- move from one vertex to another, so as to improve the objective
- repeat until we reach an optimum vertex



Simplex Method

- [Dantzig, 1946]

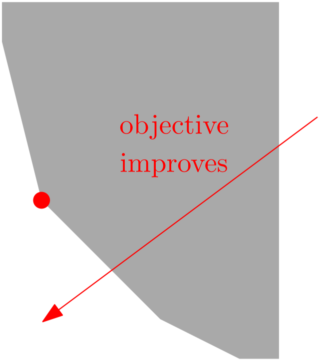
- move from one vertex to another, so as to improve the objective
- repeat until we reach an optimum vertex



Simplex Method

- [Dantzig, 1946]

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

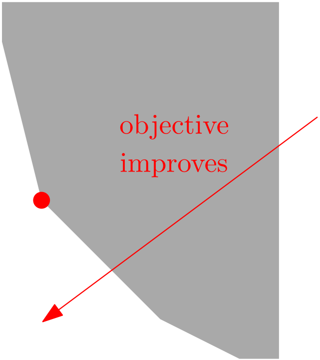


- the number of iterations might be expoentially large; but algorithm runs fast in practice

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 expoentially large; but algorithm runs fast in practice
- [Spielman-Teng,2002]: smoothed analysis

Interior Point Method

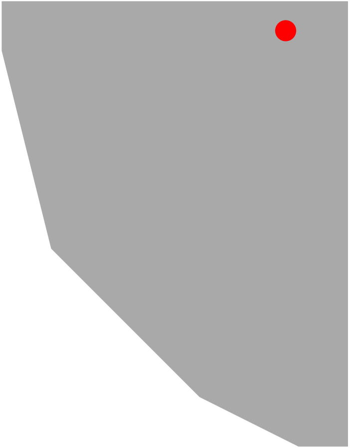
- [Karmarkar, 1984]

- 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]

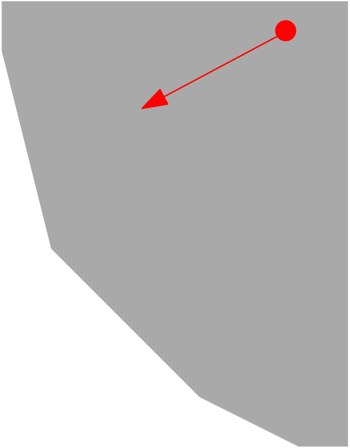
- keep the solution inside the polytope
- design penalty function so that the solution is not too close to the boundary
- the final solution will be arbitrarily close to the optimum solution



Interior Point Method

- [Karmarkar, 1984]

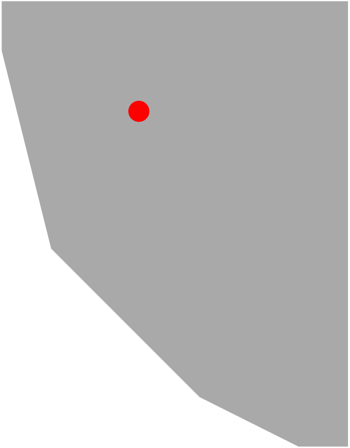
- keep the solution inside the polytope
- design penalty function so that the solution is not too close to the boundary
- the final solution will be arbitrarily close to the optimum solution



Interior Point Method

- [Karmarkar, 1984]

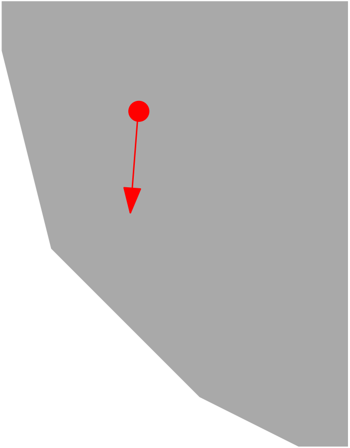
- keep the solution inside the polytope
- design penalty function so that the solution is not too close to the boundary
- the final solution will be arbitrarily close to the optimum solution



Interior Point Method

- [Karmarkar, 1984]

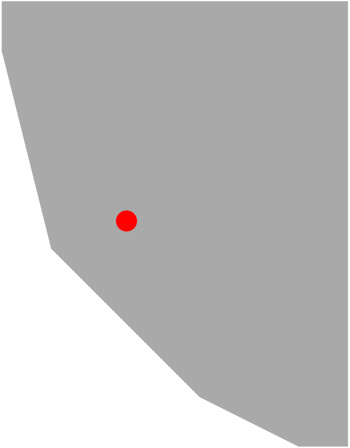
- keep the solution inside the polytope
- design penalty function so that the solution is not too close to the boundary
- the final solution will be arbitrarily close to the optimum solution



Interior Point Method

- [Karmarkar, 1984]

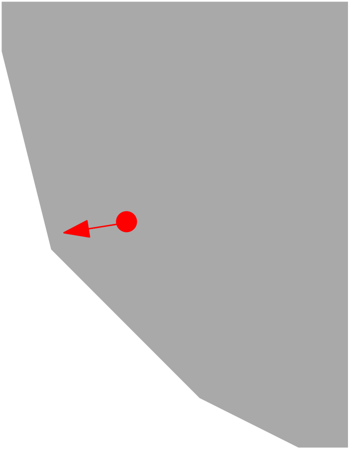
- keep the solution inside the polytope
- design penalty function so that the solution is not too close to the boundary
- the final solution will be arbitrarily close to the optimum solution



Interior Point Method

- [Karmarkar, 1984]

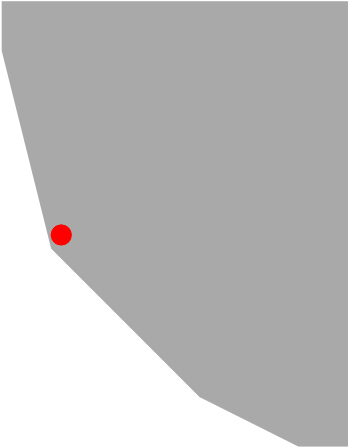
- keep the solution inside the polytope
- design penalty function so that the solution is not too close to the boundary
- the final solution will be arbitrarily close to the optimum solution



Interior Point Method

- [Karmarkar, 1984]

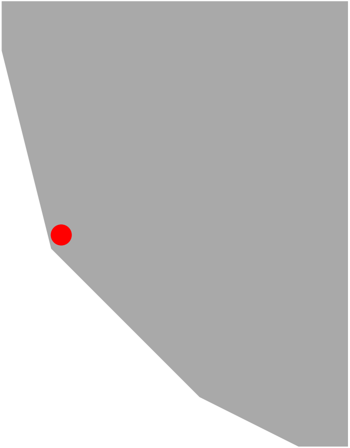
- keep the solution inside the polytope
- design penalty function so that the solution is not too close to the boundary
- the final solution will be arbitrarily close to the optimum solution



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



- polynomial time

Ellipsoid Method

- [Khachiyan, 1979]

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

Ellipsoid Method

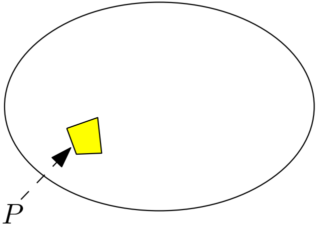
- [Khachiyan, 1979]
- used to decide if the feasible region is empty or not

- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the elliposid 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

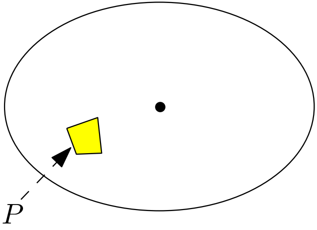
- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the elliposid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat



Ellipsoid Method

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

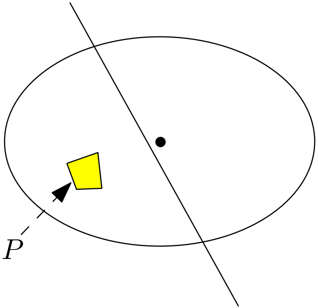
- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the elliposid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat



Ellipsoid Method

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

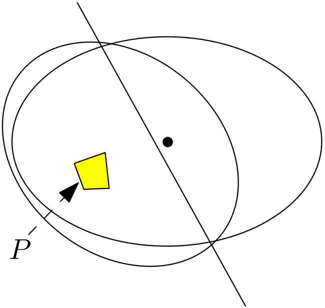
- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the elliposid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat



Ellipsoid Method

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

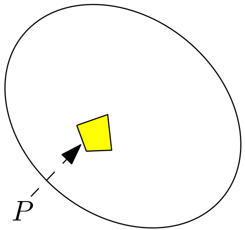
- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the elliposid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat



Ellipsoid Method

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

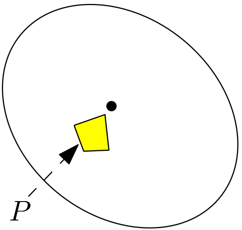
- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the elliposid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat



Ellipsoid Method

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

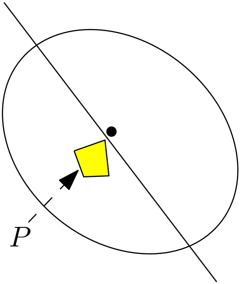
- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the elliposid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat



Ellipsoid Method

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

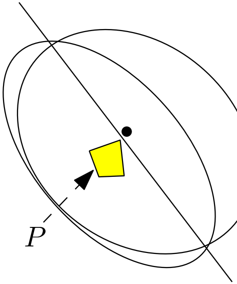
- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the elliposid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat



Ellipsoid Method

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

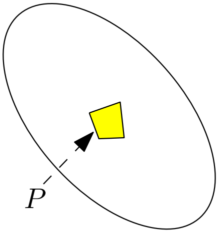
- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the elliposid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat



Ellipsoid Method

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

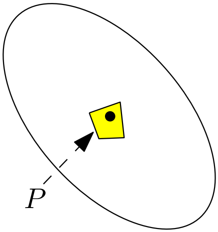
- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the elliposid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat



Ellipsoid Method

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

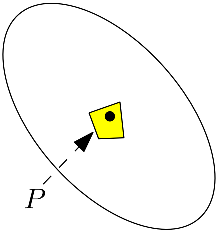
- maintain an ellipsoid that contains the feasible region
- query a separation oracle if the center of ellipsid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the elliposid in half, find smaller ellipsoid to enclose the half-ellipsoid, and repeat



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 ellipsid is in the feasible region:
  - yes: then the feasible region is not empty
  - no: cut the elliposid 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 metheod 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.

Beverage	Corn (pounds)	Hops (pounds)	Malt (pounds)	Profit ($)
Ale (barrel)	5	4	35	13
Beer (barrel)	15	4	20	23
Constraint	480	160	1190	

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.

Beverage	Corn (pounds)	Hops (pounds)	Malt (pounds)	Profit ($)
Ale (barrel)	5	4	35	13
Beer (barrel)	15	4	20	23
Constraint	480	160	1190	

- How can brewer maximize profits?

* `http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/LinearProgrammingI.pdf`

Brewery Problem *

Beverage	Corn (pounds)	Hops (pounds)	Malt (pounds)	Profit ($)
Ale (barrel)	5	4	35	13
Beer (barrel)	15	4	20	23
Constraint	480	160	1190	

Brewery Problem *

Beverage	Corn (pounds)	Hops (pounds)	Malt (pounds)	Profit ($)
Ale (barrel)	5	4	35	13
Beer (barrel)	15	4	20	23
Constraint	480	160	1190	

$$\begin{aligned} \max \quad & 13x + 23y & \text{Profit} \\ 5x + 15y &\le 480 & \text{Corn} \\ 4x + 4y &\le 160 & \text{Hops} \\ 35x + 20y &\le 1190 & \text{Malt} \\ x, y &\ge 0 \end{aligned}$$

* http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/LinearProgrammingI.pdf

Outline

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:

$$\max / \min \quad c^{\mathrm{T}} x \quad Ax \leq b.$$

Outline

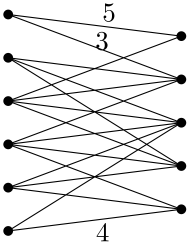
Example: Bipartite Matching Polytope

Maximum Weight Bipartite Matching

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

edge weights $w \in \mathbb{Z}_{>0}^E$

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



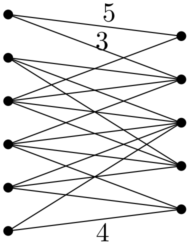
Example: Bipartite Matching Polytope

Maximum Weight Bipartite Matching

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

edge weights $w \in \mathbb{Z}_{>0}^E$

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



LP Relaxation

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

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

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

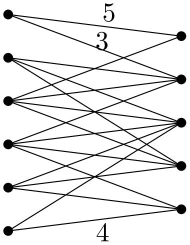
Example: Bipartite Matching Polytope

Maximum Weight Bipartite Matching

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

edge weights $w \in \mathbb{Z}_{>0}^{E}$

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



LP Relaxation

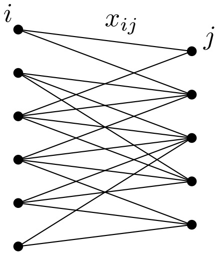
$$\max \quad \sum_{e \in E} w_e x_e$$

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

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

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

Example: Bipartite Matching Polytope



Maximum Weight Bipartite Matching

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

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: $x_e \in \{0, 1\}$: $e \in M$?

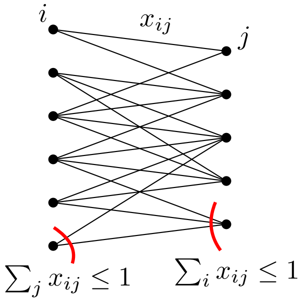
Example: Bipartite Matching Polytope

Maximum Weight Bipartite Matching

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

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

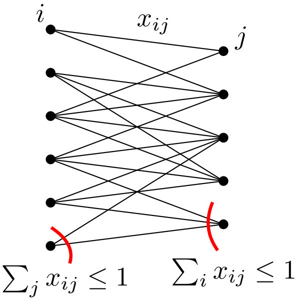
$$\max \quad \sum_{e \in E} w_e x_e$$

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

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

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

Example: Bipartite Matching Polytope



Maximum Weight Bipartite Matching

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

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

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

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

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

- In IP: $x_e \in \{0,1\}$: $e \in M$?
- $\chi^M \in \{0,1\}^E$: $\chi^M_e = 1$ 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.

$\square$

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

$\square$

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 $\Longrightarrow$ x non-vertex

$\square$

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 $\Longrightarrow$ x non-vertex
- find $x', x'' \in P$: $x' \neq x'', x = \frac{1}{2}(x' + x'')$

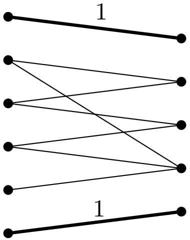
$\square$

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 $\Longrightarrow$ x non-vertex
- find $x', x'' \in P$: $x' \neq x'', x = \frac{1}{2}(x' + x'')$
- case 1: fractional edges contain a cycle

$\square$

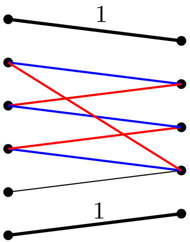


Theorem The LP polytope is integral: It is the convex hull of $\{\chi^M : M \text{ is a matching}\}$.

Proof.

- take x in the polytope P
- prove: x non integral $\Longrightarrow$ 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

$\square$

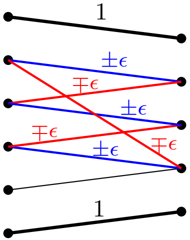


Theorem The LP polytope is integral: It is the convex hull of $\{\chi^M : M \text{ is a matching}\}$.

Proof.

- take x in the polytope P
- prove: x non integral $\Longrightarrow$ 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

$\square$

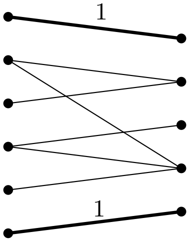


Theorem The LP polytope is integral: It is the convex hull of $\{\chi^M : M \text{ is a matching}\}$.

Proof.

- take x in the polytope P
- prove: x non integral $\Longrightarrow$ 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

$\square$

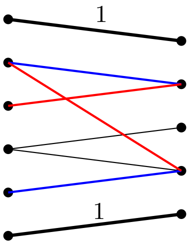


Theorem The LP polytope is integral: It is the convex hull of $\{\chi^M : M \text{ is a matching}\}$.

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

□

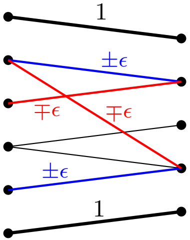


Theorem The LP polytope is integral: It is the convex hull of $\{\chi^M : M \text{ is a matching}\}$.

Proof.

- take x in the polytope P
- prove: x non integral $\Longrightarrow$ 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

□

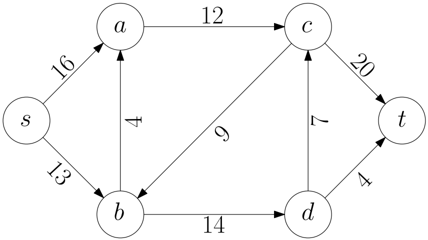


Outline

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 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)
- $\forall v \in V \setminus \{s,t\}$,

$$\sum_{e \in \delta^{\text{in}}(v)} f_e = \sum_{e \in \delta^{\text{out}}(v)} f_e \qquad \text{(flow conservation)}$$

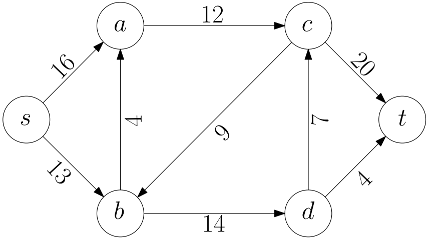
The value of flow f is defined as:

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

Maximum Flow Problem

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

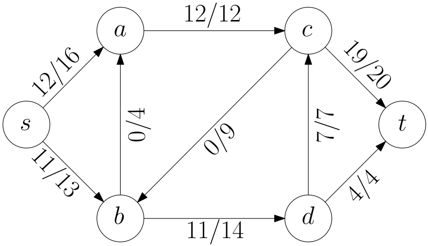
Output: maximum value of a s-t flow f



Maximum Flow Problem

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

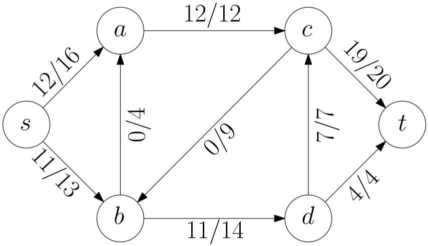
Output: maximum value of a s-t flow f



Maximum Flow Problem

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

Output: maximum value of a s-t flow f

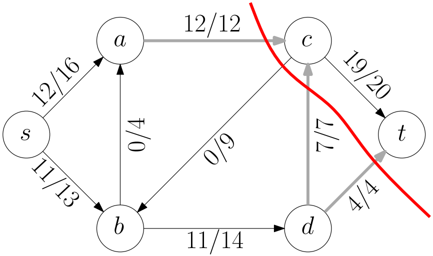


- Ford-Fulkerson method

Maximum Flow Problem

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

Output: maximum value of a s-t flow f

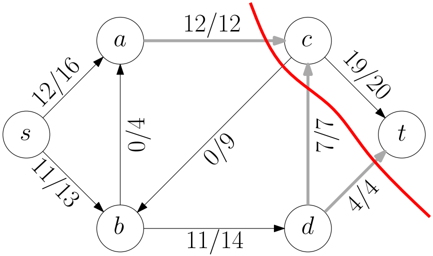


- Ford-Fulkerson method
- Maximum-Flow Min-Cut Theorem: value of the maximum flow is equal to the value of the minimum s-t cut

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

$$
\begin{aligned}
\max \quad & \sum_{e \in \delta^{\text{in}}(t)} x_e \\
& x_e \le c_e \qquad \forall e \in E \\
\sum_{e \in \delta^{\text{out}}(v)} x_e - \sum_{e \in \delta^{\text{in}}(v)} & x_e = 0 \qquad \forall v \in V \setminus \{s, t\} \\
& x_e \ge 0 \qquad \forall e \in E
\end{aligned}
$$

LP for Maximum Flow

$$\max \sum_{e\in\delta^{\text{in}}(t)} x_e$$

$$\begin{aligned} x_e &\leq c_e && \forall e \in E \\ \sum_{e\in\delta^{\text{out}}(v)} x_e - \sum_{e\in\delta^{\text{in}}(v)} x_e &= 0 && \forall v \in V \setminus \{s,t\} \\ x_e &\geq 0 && \forall e \in E \end{aligned}$$

Theorem The LP polytope is integral.

LP for Maximum Flow

$$\max \quad \sum_{e \in \delta^{\text{in}}(t)} x_e$$

$$\begin{aligned} x_e &\leq c_e & \forall e \in E \\ \sum_{e \in \delta^{\text{out}}(v)} x_e - \sum_{e \in \delta^{\text{in}}(v)} x_e &= 0 & \forall v \in V \setminus \{s,t\} \\ x_e &\geq 0 & \forall e \in E \end{aligned}$$

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 ≥ 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 cyle/path $\square$

Outline

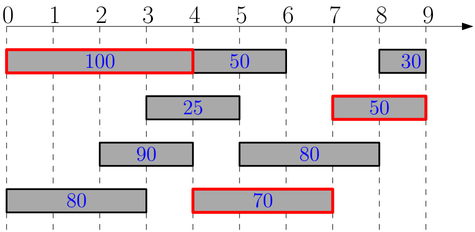
1. Linear Programming and Rounding
2. Exact Algorithms Using LP: Integral Polytopes
   - Bipartite Matching Polytope
   - s-t Flow Polytope
   - Weighted Interval Scheduling Problem
3. Approximation Algorithms Using LP: LP Rounding
   - 2-Approximation Algorithm for Weighted Vertex Cover
   - 2-Approximation 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



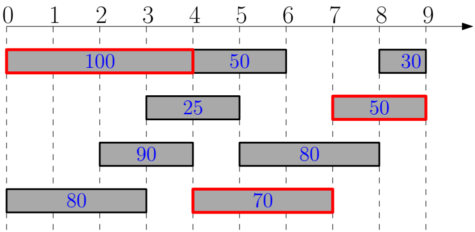
- optimum value$= 220$

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

$$\max \quad \sum_{j\in[n]} x_j w_j$$

$$\sum_{j\in[n]:t\in[s_j,f_j)} x_j \leq 1 \qquad \forall t \in [T]$$

$$x_j \geq 0 \qquad \forall j \in [n]$$

Weighted Interval Scheduling Problem

Linear Program

$$\max \quad \sum_{j \in [n]} x_j w_j$$

$$\sum_{j \in [n]: t \in [s_j, f_j)} x_j \leq 1 \qquad \forall t \in [T]$$

$$x_j \geq 0 \qquad \forall j \in [n]$$

Theorem The LP polytope is integral.

Weighted Interval Scheduling Problem

Linear Program

$$\max \quad \sum_{j\in[n]} x_j w_j$$

$$\sum_{j\in[n]:t\in[s_j,f_j)} x_j \leq 1 \qquad \forall t \in [T]$$

$$x_j \geq 0 \qquad \forall j \in [n]$$

Theorem The LP polytope is integral.

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

Weighted Interval Scheduling Problem

Linear Program

$$\max \quad \sum_{j \in [n]} x_j w_j$$

$$\sum_{j \in [n]: t \in [s_j, f_j)} x_j \leq 1 \qquad \forall t \in [T]$$

$$x_j \geq 0 \qquad \forall j \in [n]$$

Theorem The LP polytope is integral.

Def. A matrix $A \in \mathbb{R}^{m \times n}$ is said to be tototally 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.

Weighted Interval Scheduling Problem

Linear Program

$$\max \quad \sum_{j\in[n]} x_j w_j$$

$$\sum_{j\in[n]:t\in[s_j,f_j)} x_j \le 1 \qquad \forall t \in [T]$$

$$x_j \ge 0 \qquad \forall j \in [n]$$

Theorem The LP polytope is integral.

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

Theorem If a polytope P is defined by $Ax \ge b, x \ge 0$ 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): $\begin{pmatrix} A' & 0 \\ 0 & I \end{pmatrix} x = \begin{pmatrix} b' \\ 0 \end{pmatrix}$, 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'.

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: $x_i^1 = \frac{\det(A_i'|b)}{\det(A')}$ for every $i \implies x_i^1$ is integer

  $A_i'|b$: the matrix of A' with the i-th column replaced by b $\square$

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
- #edges $=$ #vertices $\implies$ underlying undirected graph contains a cycle $\implies$ det$(A') = 0$ $\square$

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
- $\#$edges $=$ $\#$vertices $\implies$ underlying undirected graph contains a cycle $\implies$ det$(A') = 0$ $\square$

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
- #edges $=$ #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

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

Example for the Proof

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

Example for the Proof

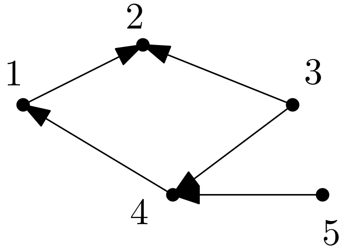
$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \color{red}{1} & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

Example for the Proof

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

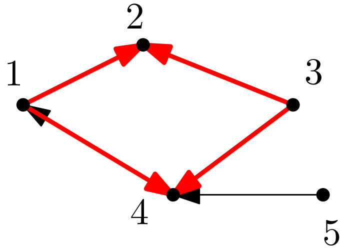
Example for the Proof

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



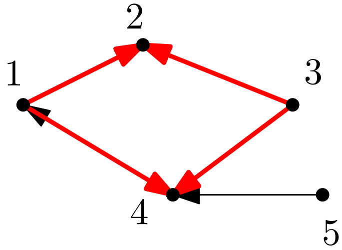
Example for the Proof

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



Example for the Proof

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



$$\begin{array}{cccccc} + & (1 & -1 & 0 & 0 & 0) \\ - & (0 & -1 & 1 & 0 & 0) \\ + & (0 & 0 & 1 & -1 & 0) \\ - & (1 & 0 & 0 & -1 & 0) \\ \\ = & (0 & 0 & 0 & 0 & 0) \end{array}$$

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,

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

$$M = \begin{pmatrix} 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -1 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}. \ \det(M) = 1.$$

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
$$M = \begin{pmatrix} 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -1 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}.\ \det(M) = 1.$$
- $\det(A'M) \in \{0, \pm 1\} \implies \det(A') \in \{0, \pm 1\}$. □

Example for the Proof

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

Example for the Proof

$$\begin{pmatrix} \color{red}{0} & \color{red}{1} & \color{red}{1} & \color{red}{1} & \color{red}{0} & 0 \\ \color{red}{1} & \color{red}{1} & \color{red}{1} & \color{red}{1} & \color{red}{0} & 0 \\ \color{red}{0} & \color{red}{0} & \color{red}{1} & \color{red}{1} & \color{red}{1} & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ \color{red}{0} & \color{red}{0} & \color{red}{0} & \color{red}{1} & \color{red}{1} & 0 \\ \color{red}{0} & \color{red}{1} & \color{red}{1} & \color{red}{1} & \color{red}{1} & 0 \end{pmatrix}$$

Example for the Proof

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

Example for the Proof

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

- $(\text{col } 1, \text{col } 2 - \text{col } 1, \text{col } 3 - \text{col } 2, \text{col } 4 - \text{col } 3, \text{col } 5 - \text{col } 4)$

Example for the Proof

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

- $(\text{col } 1, \text{col } 2 - \text{col } 1, \text{col } 3 - \text{col } 2, \text{col } 4 - \text{col } 3, \text{col } 5 - \text{col } 4)$

Example for the Proof

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

- $(\text{col } 1, \text{col } 2 - \text{col } 1, \text{col } 3 - \text{col } 2, \text{col } 4 - \text{col } 3, \text{col } 5 - \text{col } 4)$
- 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.

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

Example

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

Proof.

- $G = (L \uplus R, E)$: 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.

- $G = (L \uplus R, E)$: 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.

- $G = (L \uplus R, E)$: 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 $\square$

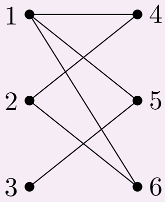
Example

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

Proof.

- $G = (L \uplus R, E)$: 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 $\square$

Example

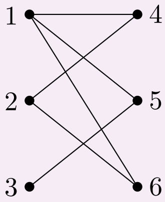


Lemma The edge-vertex incidence matrix A of a bipartite graph is totally-unimodular.

Proof.

- $G = (L \uplus R, E)$: 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 $\square$

Example



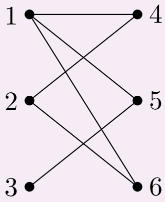
$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

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

Proof.

- $G = (L \uplus R, E)$: 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 $\square$

Example



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

- remark: bipartiteness is needed. The edge-vertex incidence matrix $\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$ of a triangle has determinent 2.

- 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 determinent 2.

Coro. Bipartite matching polytope is integral.

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

Approximation Algorithm based on LP Rounding

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

Approximation Algorithm based on LP Rounding

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

0/1 Integer Program

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x \\ Ax & \geq b \\ x & \in \{0,1\}^n \end{aligned}$$

Linear Program Relaxation

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

Approximation Algorithm based on LP Rounding

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

0/1 Integer Program

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x \\ Ax \geq b & \\ x \in \{0,1\}^n & \end{aligned}$$

Linear Program Relaxation

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

- LP $\leq$ IP

Approximation Algorithm based on LP Rounding

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

0/1 Integer Program

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x \\ Ax &\geq b \\ x &\in \{0,1\}^n \end{aligned}$$

Linear Program Relaxation

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

- LP $\leq$ IP
- Integer programming is NP-hard, linear programming is in P

Approximation Algorithm based on LP Rounding

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

0/1 Integer Program

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

Linear Program Relaxation

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

- LP $\leq$ IP
- Integer programming is NP-hard, linear programming is in P
- 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

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x \\ Ax & \geq b \\ x & \in \{0,1\}^n \end{aligned}$$

Linear Program Relaxation

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

- 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) $\xrightarrow{\text{relax}}$ LP

0/1 Integer Program

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x \\ Ax & \geq b \\ x & \in \{0,1\}^n \end{aligned}$$

Linear Program Relaxation

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

- 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 $c^{\mathrm{T}}\tilde{x} \leq \alpha \cdot c^{\mathrm{T}}x$, then $c^{\mathrm{T}} \cdot \tilde{x} \leq \alpha \cdot \mathrm{LP} \leq \alpha \cdot \mathrm{IP} = \alpha \cdot \mathrm{opt}$

Approximation Algorithm based on LP Rounding

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

0/1 Integer Program

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x \\ Ax &\geq b \\ x &\in \{0,1\}^n \end{aligned}$$

Linear Program Relaxation

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

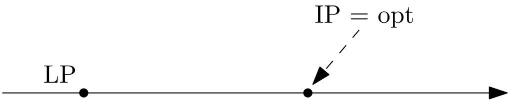
- LP $\leq$ IP
- Integer programming is NP-hard, linear programming is in P
- Solve LP to obtain a fractional $x \in [0,1]^n$.
- Round it to an integral $\tilde{x} \in \{0,1\}^n \iff$ solution for X
- Prove $c^{\mathrm{T}}\tilde{x} \leq \alpha \cdot c^{\mathrm{T}} x$, then $c^{\mathrm{T}} \cdot \tilde{x}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

IP

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x \\ Ax &\geq b \\ x &\in \{0,1\}^n \end{aligned}$$

LP Relaxation

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

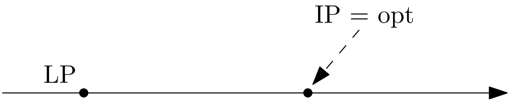


IP

$$\begin{aligned} \min \quad & c^{\mathrm{T}}x \\ Ax &\geq b \\ x &\in \{0,1\}^n \end{aligned}$$

LP Relaxation

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



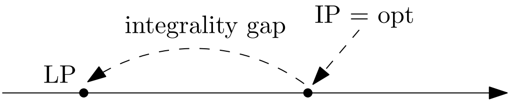
Def. The ratio between IP $=$ opt and LP is called the integrality gap of the LP relaxation.

IP

$$\begin{aligned} \min \quad & c^{\mathrm{T}}x \\ Ax \geq b & \\ x \in \{0,1\}^n & \end{aligned}$$

LP Relaxation

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



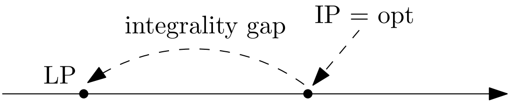
Def. The ratio between IP $=$ opt and LP is called the integrality gap of the LP relaxation.

IP

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x \\ Ax & \geq b \\ x & \in \{0,1\}^n \end{aligned}$$

LP Relaxation

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



Def. The ratio between IP $=$ opt and LP is called the integrality gap of the LP relaxation.

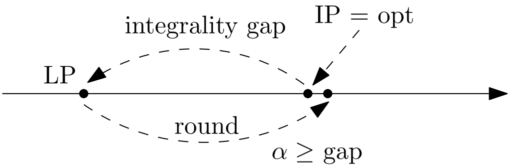
- The approximation ratio based on this analysis can not be better than the worst integrality gap.

IP

$$\begin{aligned} \min \quad & c^{\mathrm{T}}x \\ Ax & \geq b \\ x & \in \{0,1\}^n \end{aligned}$$

LP Relaxation

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



Def. The ratio between $\mathrm{IP} = \mathrm{opt}$ and LP is called the integrality gap of the LP relaxation.

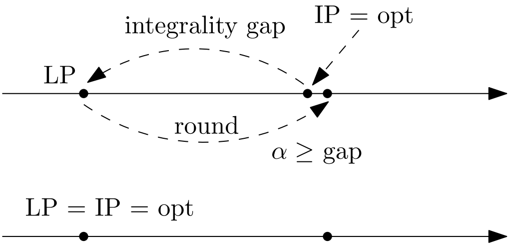
- The approximation ratio based on this analysis can not be better than the worst integrality gap.

IP

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x \\ Ax & \geq b \\ x & \in \{0,1\}^n \end{aligned}$$

LP Relaxation

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



Def. The ratio between IP $=$ opt and LP is called the integrality gap of the LP relaxation.

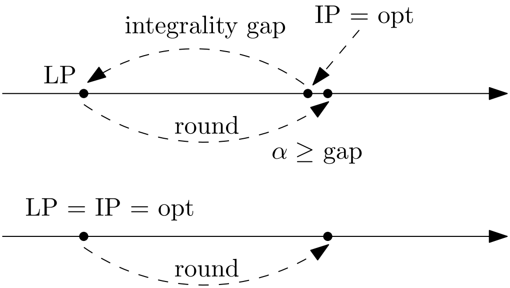
- The approximation ratio based on this analysis can not be better than the worst integrality gap.

IP

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x \\ Ax &\geq b \\ x &\in \{0,1\}^n \end{aligned}$$

LP Relaxation

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

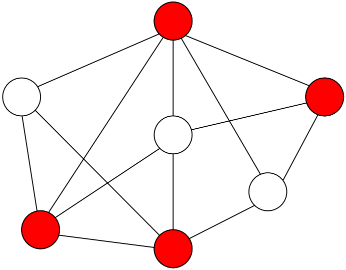


Def. The ratio between IP $=$ opt and LP is called the integrality gap of the LP relaxation.

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

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 $w \in \mathbb{Z}_{>0}^V$

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

Integer Program

$$\min \quad \sum_{v \in V} w_v x_v$$
$$x_u + x_v \geq 1 \qquad \forall (u,v) \in E$$
$$x_v \in \{0,1\} \qquad \forall v \in V$$

LP Relaxation

$$\min \quad \sum_{v \in V} w_v x_v$$
$$x_u + x_v \geq 1 \qquad \forall (u,v) \in E$$
$$x_v \in [0,1] \qquad \forall v \in V$$

- $x_v \in \{0,1\}, \forall v \in V$: indicate if we include v in the vertex cover

Integer Program

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

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

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

LP Relaxation

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

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

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

- IP := value of integer program, LP := value of linear program
- LP $\leq$ IP $= \mathrm{opt}$

Rounding Algorithm

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

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

Rounding Algorithm

Solve LP to obtain solution $\{x^*_u\}_{u \in V}$
$\triangleright$ So, LP $= \sum_{u \in V} w_u x^*_u \leq$ IP

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 $= \sum_{u \in V} w_u x_u^* \leq$ 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 $\{x^*_u\}_{u \in V}$

$\triangleright$ So, LP $= \sum_{u \in V} w_u x^*_u \leq$ 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 $\{x_u^*\}_{u\in V}$

$\triangleright$ So, LP $= \sum_{u\in V} w_u x_u^* \leq$ 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 $v \in S$. □

Rounding Algorithm

1: Solve LP to obtain solution $\{x^*_u\}_{u \in V}$

$\triangleright$ So, LP $= \sum_{u \in V} w_u x^*_u \leq$ 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 $\{x_u^*\}_{u \in V}$

$\triangleright$ So, $\mathrm{LP} = \sum_{u \in V} w_u x_u^* \leq \mathrm{IP}$

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

Lemma S is a vertex cover of G.

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

Rounding Algorithm

1: Solve LP to obtain solution $\{x^*_u\}_{u \in V}$

$\triangleright$ So, LP $= \sum_{u \in V} w_u x^*_u \le$ IP

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

Lemma S is a vertex cover of G.

Lemma $\mathrm{cost}(S) := \sum_{u \in S} w_u \le 2 \cdot \mathrm{LP}$.

Proof.

$$\mathrm{cost}(S) = \sum_{u \in S} w_u \le \sum_{u \in S} w_u \cdot 2x^*_u = 2 \sum_{u \in S} w_u \cdot x^*_u$$

$$\le 2 \sum_{u \in V} w_u \cdot x^*_u = 2 \cdot \mathrm{LP}. \qquad \square$$

Rounding Algorithm

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

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

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

Lemma S is a vertex cover of G.

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

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

Rounding Algorithm

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

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

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

Lemma S is a vertex cover of G.

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

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

Proof.

cost$(S) \leq 2 \cdot$ LP $\leq 2 \cdot$ IP $= 2 \cdot$ (optimum value) $\square$

Outline

Unrelated Machine Scheduling

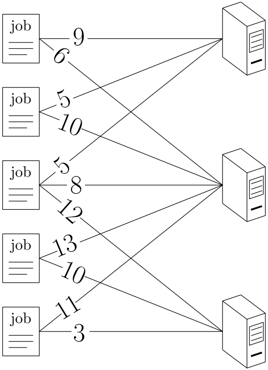
Input: $J, |J| = n$: jobs

$M, |M| = m$: machines

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

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

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

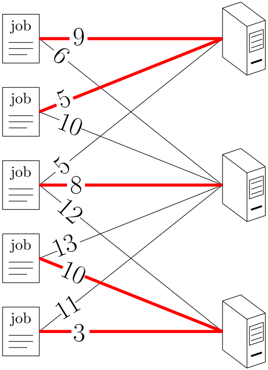


Unrelated Machine Scheduling

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:

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



Unrelated Machine Scheduling

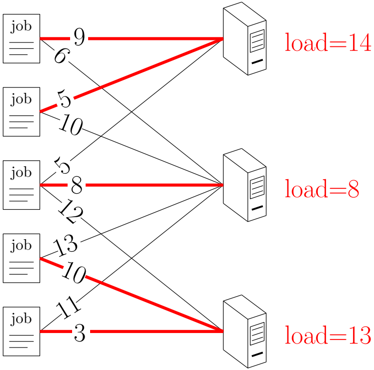
Input: $J, |J| = n$: jobs

$M, |M| = m$: 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

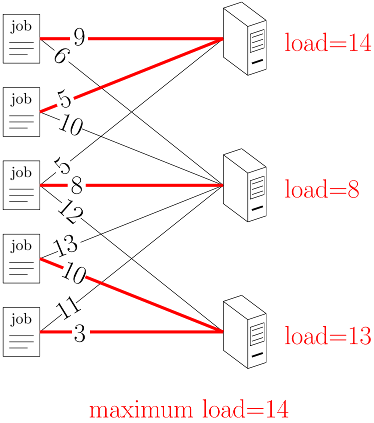
Input: $J, |J| = n$: jobs

$M, |M| = m$: 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

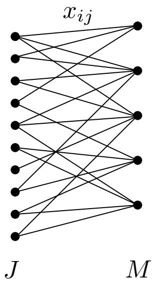
$$
\begin{aligned}
\sum_i x_{ij} &= 1 && \forall j \in J \\
\sum_j p_{ij} x_{ij} &\le T && \forall i \in M \\
x_{ij} &\ge 0 && \forall ij
\end{aligned}
$$

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

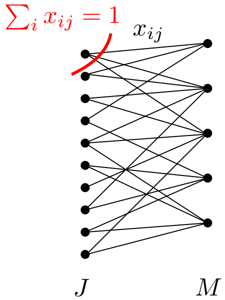
- x_{ij}: fraction of j assigned to i

$$
\begin{aligned}
\sum_i x_{ij} &= 1 & \forall j \in J \\
\sum_j p_{ij} x_{ij} &\le T & \forall i \in M \\
x_{ij} &\ge 0 & \forall ij
\end{aligned}
$$

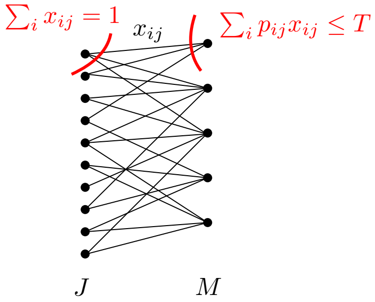
2-Approximate Rounding Algorithm of Shmoys-Tardos



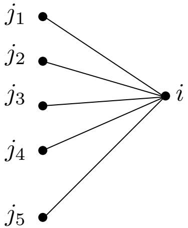
2-Approximate Rounding Algorithm of Shmoys-Tardos



2-Approximate Rounding Algorithm of Shmoys-Tardos

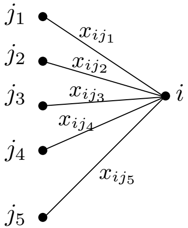


2-Approximate Rounding Algorithm of Shmoys-Tardos



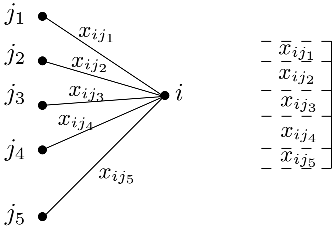
$p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_5}$

2-Approximate Rounding Algorithm of Shmoys-Tardos



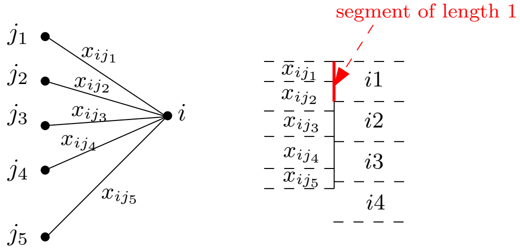
$p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_5}$

2-Approximate Rounding Algorithm of Shmoys-Tardos



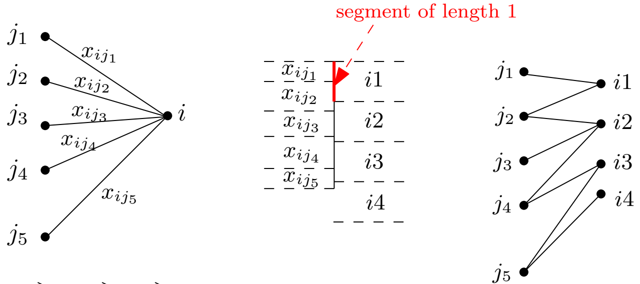
$p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_5}$

2-Approximate Rounding Algorithm of Shmoys-Tardos



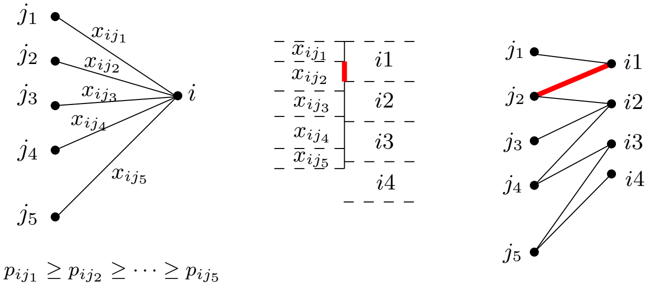
$p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_5}$

2-Approximate Rounding Algorithm of Shmoys-Tardos

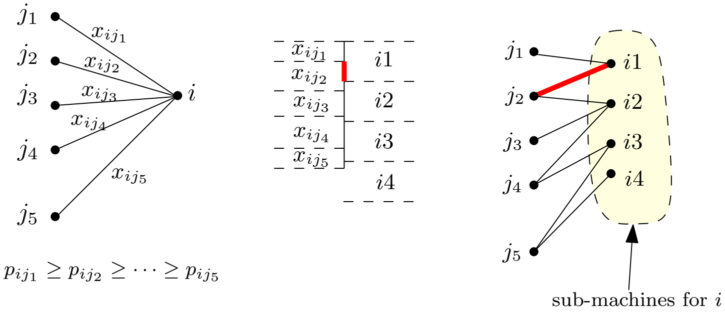


$p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_5}$

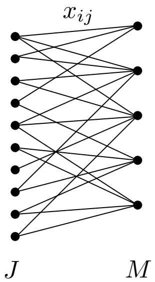
2-Approximate Rounding Algorithm of Shmoys-Tardos

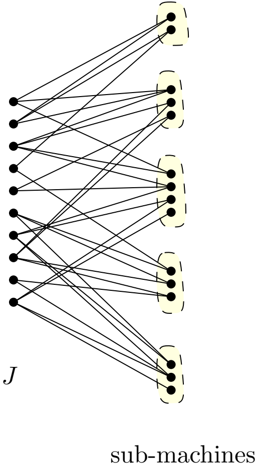


2-Approximate Rounding Algorithm of Shmoys-Tardos

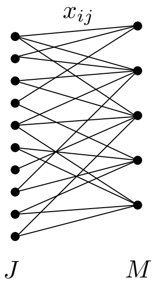


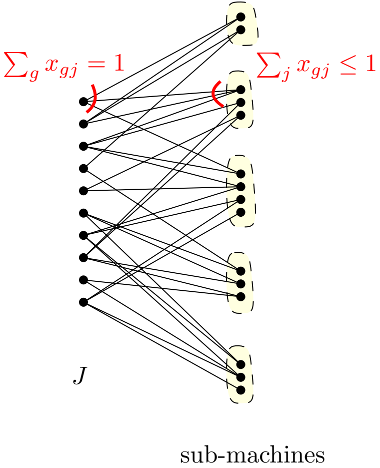
2-Approximate Rounding Algorithm of Shmoys-Tardos



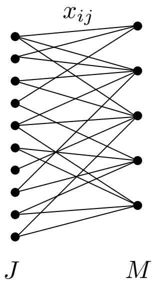


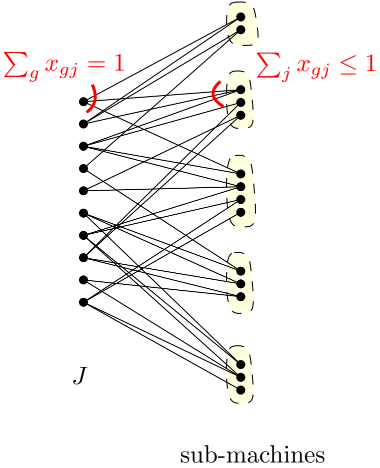
2-Approximate Rounding Algorithm of Shmoys-Tardos





2-Approximate Rounding Algorithm of Shmoys-Tardos





Obs. x 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.

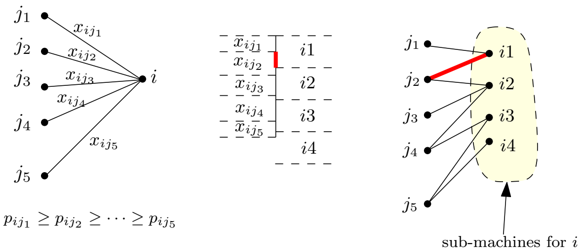
- Recall bipartite matching polytope is integral.
- x is a convex combination of matchings.

- 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.

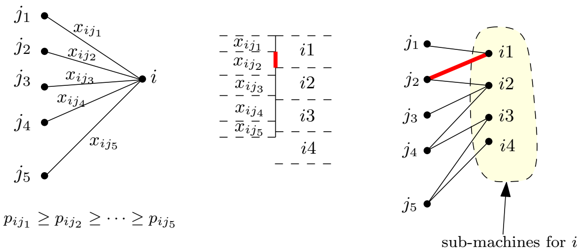
Lemma Any matching in the combination gives an schedule of makespan $\leq 2T$.

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



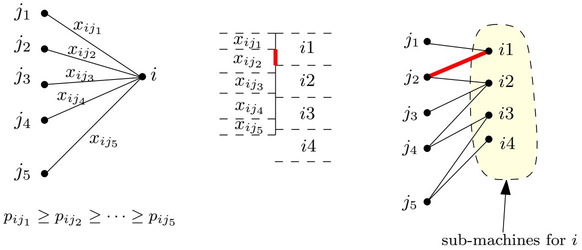
Proof.

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



Proof.

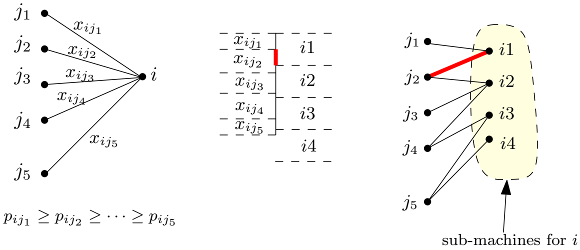
Lemma Any matching in the combination gives an schedule of makespan $\leq 2T$.



Proof.

- focus on machine i, let $i_1, i_2, \cdots, i_a$ be the sub-machines for i

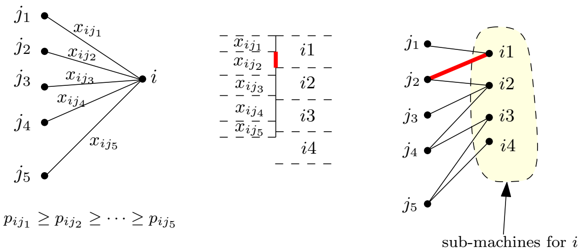
Lemma Any matching in the combination gives an schedule of makespan $\leq 2T$.



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.

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



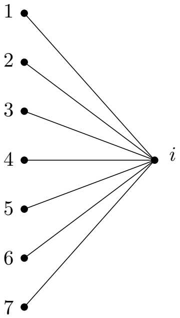
Proof.

- 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.

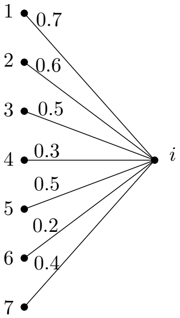
$$\text{(load on } i\text{)} = \sum_{t=1}^{a} p_{ik_t} \leq p_{ik_1} + \sum_{t=2}^{a} \sum_{j} x_{i_{t-1}j} \cdot p_{ij}$$

$$\leq p_{ik_1} + \sum_{j} x_{ij} p_{ij} \leq T + T = 2T. \qquad \square$$

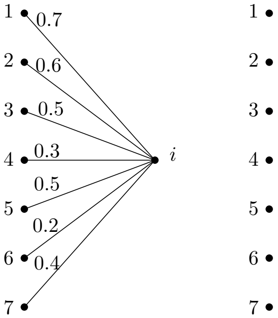
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



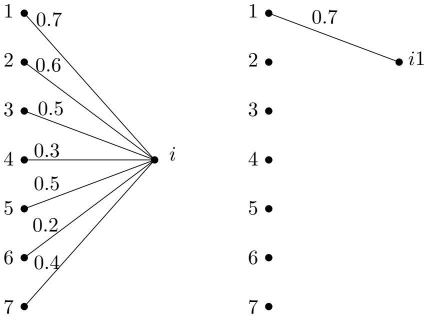
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



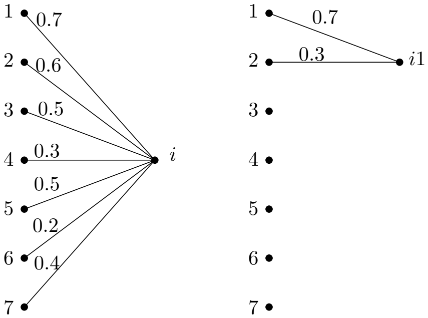
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



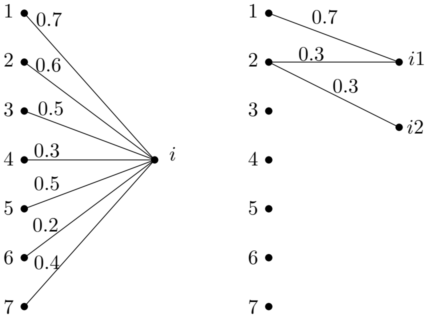
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



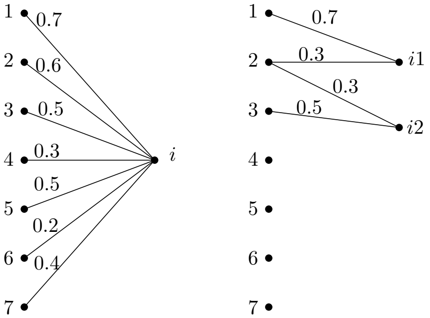
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



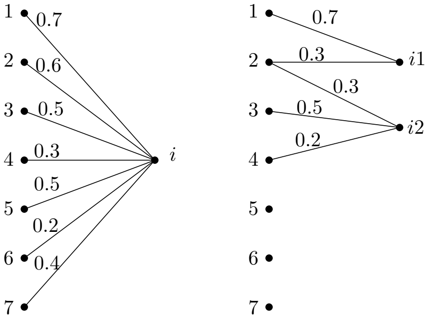
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



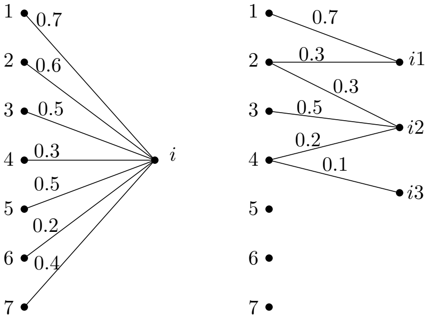
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



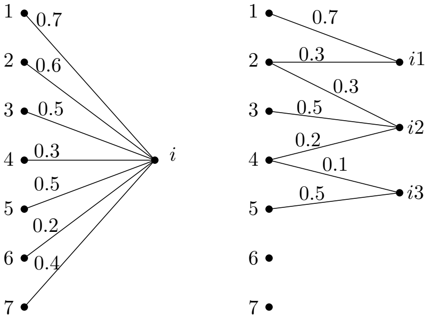
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



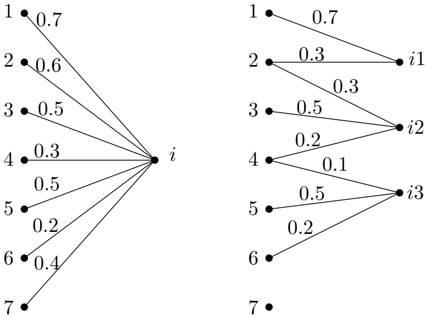
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



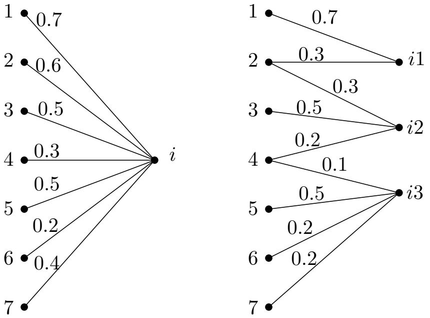
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



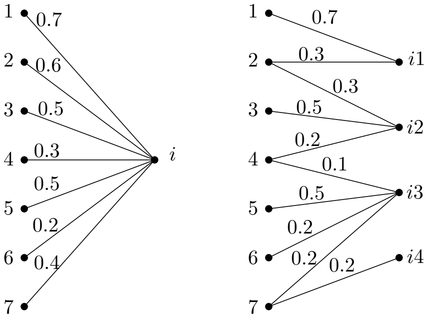
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



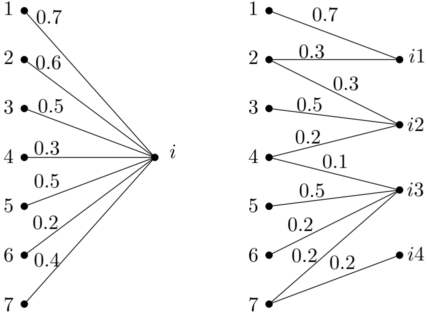
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



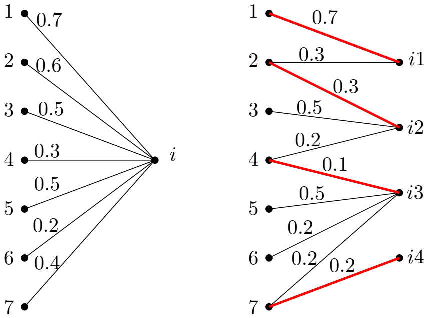
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:



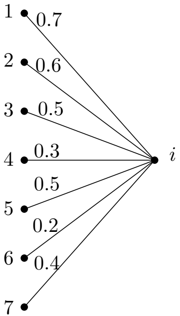
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:
  - $1 \to i1, 2 \to i2$
  - $4 \to i3, 7 \to i4$

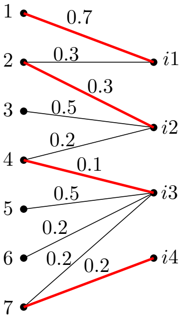


- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:
  - $1 \to i1, 2 \to i2$
  - $4 \to i3, 7 \to i4$



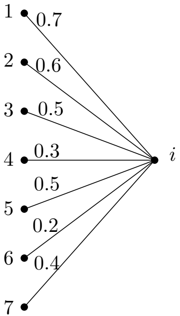
- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:
  - $1 \to i1, 2 \to i2$
  - $4 \to i3, 7 \to i4$

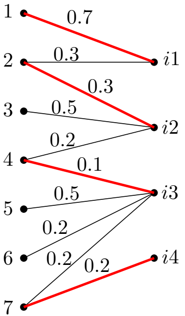




$$p_1 \leq T$$
$$p_2 \leq 0.7p_1 + 0.3p_2$$
$$p_4 \leq 0.3p_2 + 0.5p_3 + 0.2p_4$$
$$p_7 \leq 0.1p_4 + 0.5p_5 + 0.2p_6 + 0.2p_7$$

- fix i, use p_j for p_{ij}
- $p_1 \geq p_2 \geq \cdots \geq p_7$
- worst case:
  - $1 \to i1, 2 \to i2$
  - $4 \to i3, 7 \to i4$





$$p_1 \leq T$$

$$p_2 \leq 0.7p_1 + 0.3p_2$$

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

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

$$\begin{aligned}
&p_1 + p_2 + p_4 + p_7 \leq T + (0.7p_1 + 0.3p_2) + (0.3p_2 + 0.5p_3 + 0.2p_4) \\
&\quad + (0.1p_4 + 0.5p_5 + 0.2p_6 + 0.2p_7) \\
&\leq T + (0.7p_1 + 0.6p_2 + 0.5p_3 + 0.3p_4 + 0.5p_5 + 0.2p_6 + 0.4p_7) \\
&\leq T + T = 2T
\end{aligned}$$

Summary

- 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

English-Chinese Translation

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