CSE 431/531: Algorithm Analysis and Design (Spring 2018)

Linear Programming

Lecturer: Shi Li

Department of Computer Science and Engineering
University at Buffalo

Outline

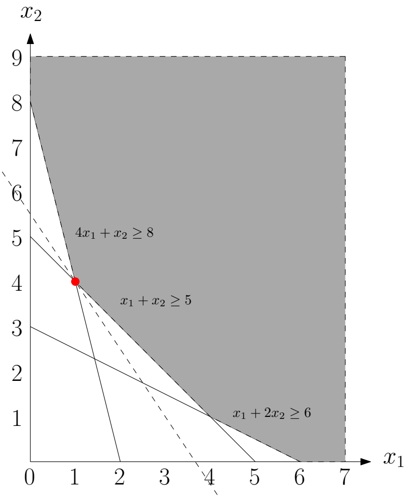
Outline

1. Linear Programming
   - Introduction
2. Network Flow
   - Ford-Fulkerson Method
3. Bipartite Matching Problem
4. 2-Approximation for Weighted Vertex Cover
5. Linear Programming Duality

Example of Linear Programming

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

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



Standard Form of Linear Programming

$$\min \qquad c_1x_1 + c_2x_2 + \cdots + c_nx_n \qquad \text{s.t.}$$

$$\sum A_{1,1}x_1 + A_{1,2}x_2 + \cdots + A_{1,n}x_n \geq b_1$$

$$\sum A_{2,1}x_1 + A_{2,2}x_2 + \cdots + A_{2,n}x_n \geq b_2$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

$$\sum A_{m,1}x_1 + A_{m,2}x_2 + \cdots + A_{m,n}x_n \geq b_m$$

$$x_1, x_2, \cdots, x_n \geq 0$$

Standard Form of Linear Programming

Let $x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$, $\qquad c = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}$,

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} \end{pmatrix}, \qquad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}.$$

Then, LP becomes $\quad \min \quad c^{\mathrm{T}} x \quad$ s.t.

$$Ax \geq b$$
$$x \geq 0$$

- $\geq$ means coordinate-wise greater than or equal to

Standard Form of Linear Programming

$$\min \quad c^{\mathrm{T}}x \quad \text{s.t.}$$
$$Ax \geq b$$
$$x \geq 0$$

- Linear programmings can be solved in polynomial time

Algorithm	Theory	Practice
Simplex Method	Exponential Time	Works Well
Ellipsoid Method	Polynomial Time	Slow
Internal Point Methods	Polynomial Time	Works Well

Applications of Linear Programming

- Design polynomial-time exact algorithms
- Design polynomial-time approximation algorithms
- Branch-and-bound algorithms to solve integer programmings

Brewery Problem (from Kevin Wayne's Notes*)

- 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 (from Kevin Wayne's Notes*)

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

- Devote all resources to ale: 34 barrels of ale $\Rightarrow$ \$442
- Devote all resources to beer: 32 barrels of beer $\Rightarrow$ \$736
- 7.5 barrels of ale, 29.5 barrels of beer $\Rightarrow$ \$776
- 12 barrels of ale, 28 barrels of beer $\Rightarrow$ \$800

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

Brewery Problem (from Kevin Wayne's Notes*)

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

$$\max \quad 13A + 23B \qquad \text{profit}$$

$$\begin{aligned} 5A + 15B &\le 480 && \text{Corn} \\ 4A + 4B &\le 160 && \text{Hops} \\ 35A + 20B &\le 1190 && \text{Malt} \\ A, B &\ge 0 \end{aligned}$$

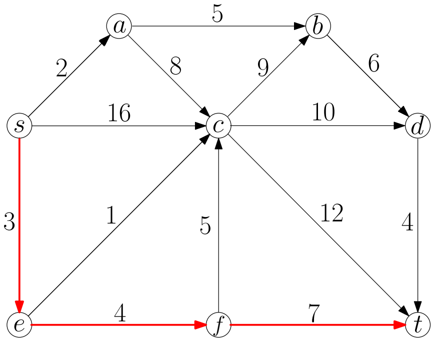
* `http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/LinearProgrammingI.pdf`

s-t Shortest Path

Input: (directed or undirected) graph $G = (V, E)$, $s, t \in V$

$w : E \to \mathbb{R}_{\geq 0}$

Output: shortest path from s to t



s-t Shortest Path Using Linear Programming

$$\max \quad d_t$$
$$d_s = 0$$
$$d_v \leq d_u + w(u,v) \qquad \forall (u,v) \in E$$

Lemma Let P be any $s \to t$ path. Then value of LP $\leq \sum_{e \in P} w_e$.

Coro. value of LP $\leq$ dist(s,t).

Lemma Let d_v be the length of the shortest path from s to v. Then $(d_v)_{v \in V}$ satisfies all the constraints in LP.

Lemma value of LP $=$ dist(s,t).

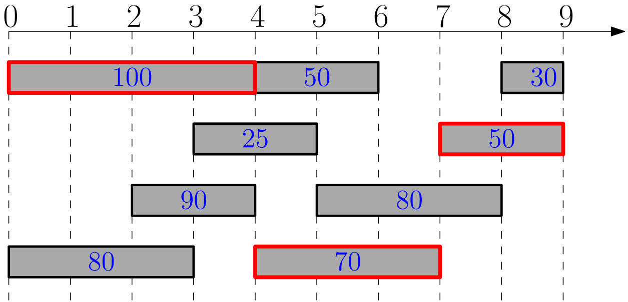
Weighted Interval Scheduling

Input: n jobs, job i with start time s_i and finish time f_i

each job has a weight (or value) $v_i > 0$

i and j are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ are disjoint

Output: a maximum-weight subset of mutually compatible jobs



Weighted Interval Scheduling Problem

Integer Programming

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

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

$$x_j \in \{0,1\} \qquad \forall j \in [n]$$

Linear Programming

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

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

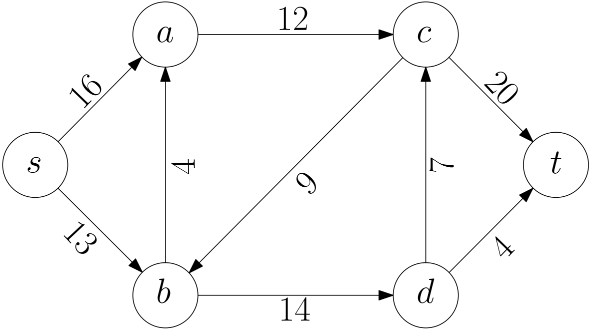
$$x_j \in [0,1] \qquad \forall j \in [n]$$

- In general, integer programming is an NP-hard problem.
- Most optimization problems can be formulated as integer programming.
- However, the above IP is equivalent to the LP!

Outline

Flow Network

- Abstraction of fluid flowing through edges
- Digraph $G = (V, E)$ with source $s \in V$ and sink $t \in V$
  - No edges enter s
  - No edges leave t
- Edge capacity $c(e) \in \mathbb{R}_{>0}$ for every $e \in E$



Def. An s-t flow is a function $f : E \to \mathbb{R}$ such that

- for every $e \in E$: $0 \le f(e) \le c(e)$ (capacity conditions)
- for every $v \in V \setminus \{s, t\}$:

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e). \qquad \text{(conservation conditions)}$$

The value of a flow f is

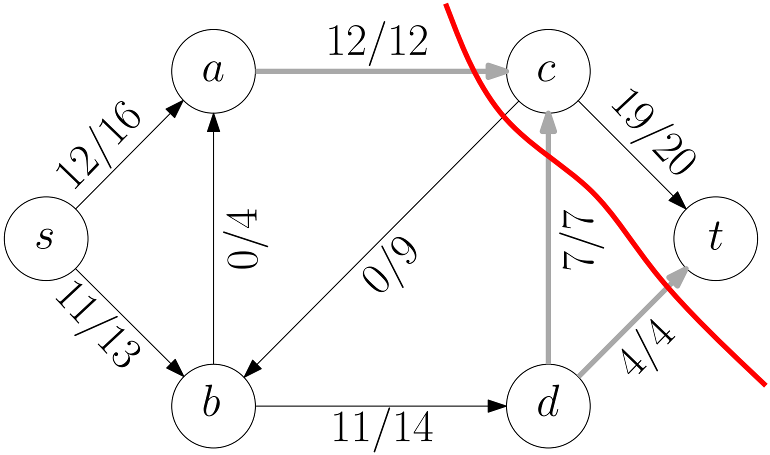
$$\mathrm{val}(f) = \sum_{e \text{ out of } s} f(e).$$

Maximum Flow Problem

Input: directed network $G = (V, E)$, capacity function $c : E \to \mathbb{R}_{>0}$, source $s \in V$ and sink $t \in V$

Output: an s-t flow f in G with the maximum $\mathrm{val}(f)$

Maximum Flow Problem: Example



Linear Programming for Max-Flow

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

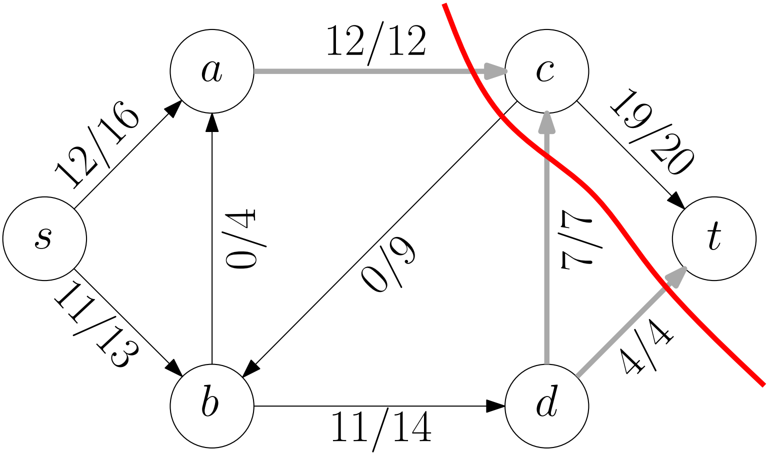
$$\begin{aligned} x_e &\le c(e) && \forall e \in E \\ \sum_{e \in \delta^{\text{in}}(v)} x_e &= \sum_{e \in \delta^{\text{out}}(v)} x_e && \forall v \in V \setminus \{s, t\} \\ x_e &\ge 0 && \forall e \in E \end{aligned}$$

Outline

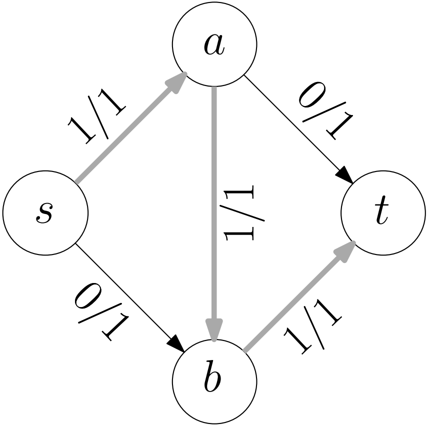
Greedy Algorithm

- Start with empty flow: $f(e) = 0$ for every $e \in E$
- Define the residual capacity of e to be $c(e) - f(e)$
- Find an augmenting path: a path from s to t, where all edges have positive residual capacity
- Augment flow along the path as much as possible
- Repeat until we got stuck

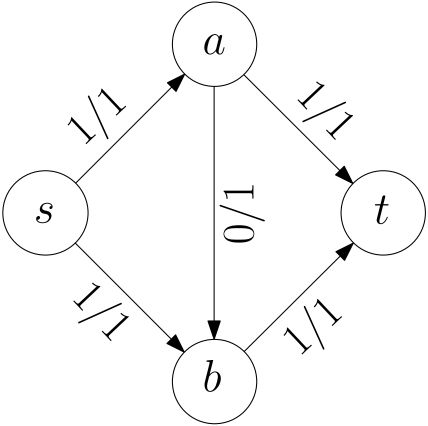
Greedy Algorithm: Example



Greedy Algorithm Does Not Always Give a Optimum Solution



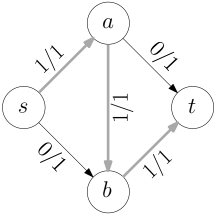
Fix the Issue: Allowing "Undo" Flow Sent



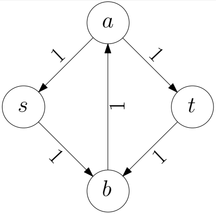
Assumption (u, v) and (v, u) can not both be in E

Def. For a s-t flow f, the residual graph G_f of $G = (V, E)$ w.r.t f contains:

- the vertex set V,
- for every $e = (u, v) \in E$ with $f(e) < c(e)$, a forward edge $e = (u, v)$, with residual capacity $c_f(e) = c(e) - f(e)$,
- for every $e = (u, v) \in E$ with $f(e) > 0$, a backward edge $e' = (v, u)$, with residual capacity $c_f(e') = f(e)$.

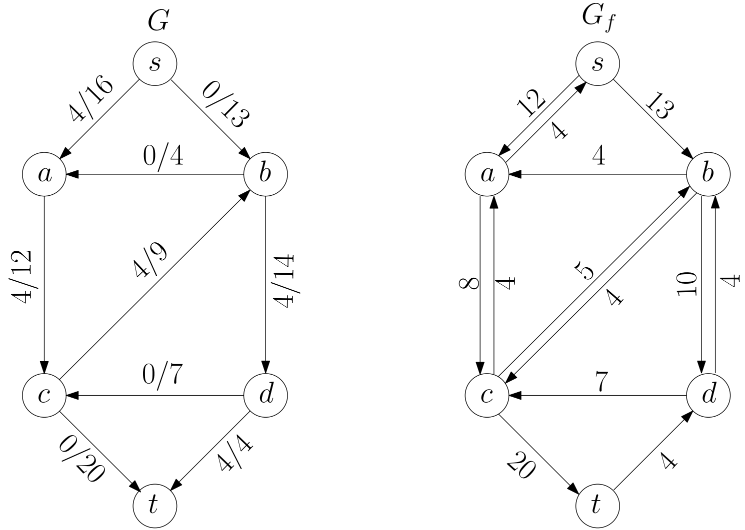


Original graph G and f



Residual Graph G_f

Residual Graph: One More Example



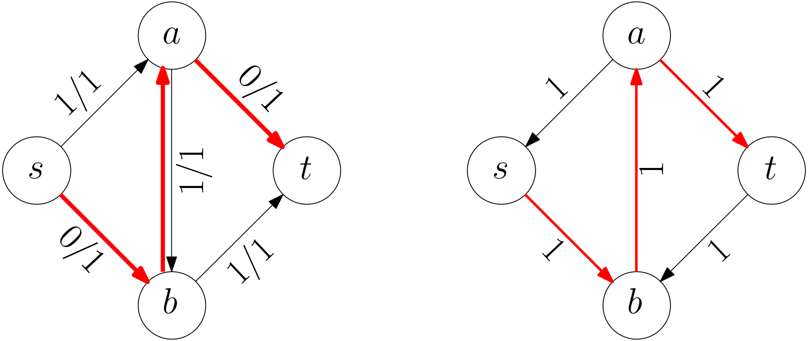
Agumenting Path

Augmenting the flow along a path P from s to t in G_f

Augment(P)

1. $b \leftarrow \min_{e \in P} c_f(e)$
2. for every $(u, v) \in P$
3.   if (u, v) is a forward edge
4. $f(u, v) \leftarrow f(u, v) + b$
5.   else   \\\\ (u, v) is a backward edge
6. $f(v, u) \leftarrow f(v, u) - b$
7.   return f

Example for Augmenting Along a Path

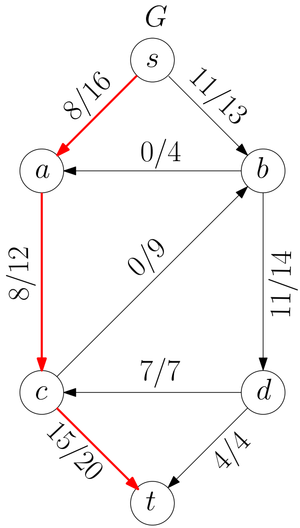


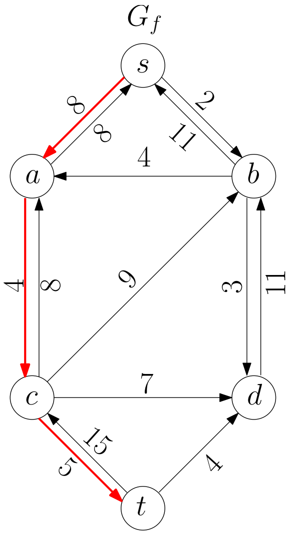
Ford-Fulkerson's Method

Ford-Fulkerson(G, s, t, c)

1. let $f(e) \leftarrow 0$ for every e in G
2. while there is a path from s to t in G_f
3. let P be any simple path from s to t in G_f
4. $f \leftarrow$augment(f, P)
5. return f

Ford-Fulkerson: Example





Correctness of Ford-Fulkerson Method

- Flow conservation conditions are satisfied
- When algorithm terminates, there is a cut in the residual graph

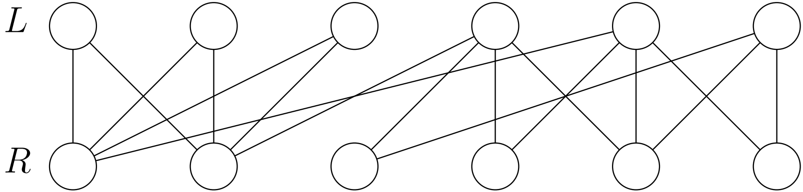
Running Time of Ford-Fulkerson Method

- Depends on #iterations
- #iterations could be exponential if augmenting paths are chosen by adversary
- #iterations=polynomial if in each iteration, we choose
  - the shortest augmenting path,
  - or the augmenting path with largest bottleneck capacity.

Outline

Bipartite Graphs

Def. A graph $G = (V, E)$ is bipartite if the vertices V can be partitioned into two subsets L and R such that every edge in E is between a vertex in L and a vertex in R.

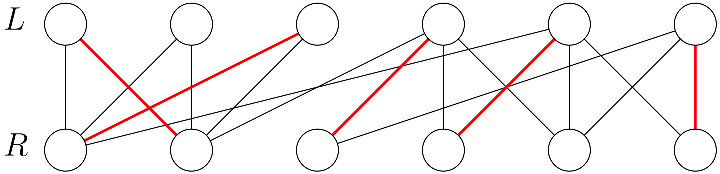


Def. Given a bipartite graph $G = (L \cup R, E)$, a matching in G is a set $M \subseteq E$ of edges such that every vertex in V is an endpoint of at most one edge in M.

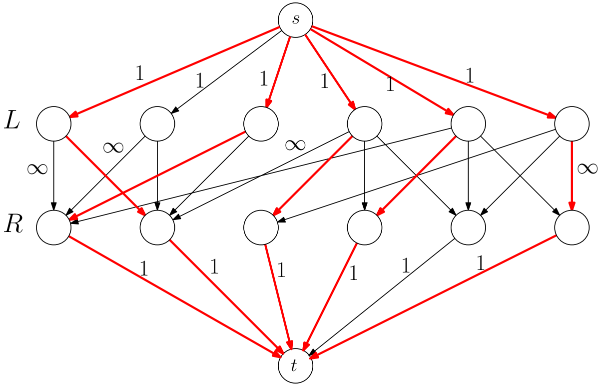
Maximum Bipartite Matching Problem

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

Output: a matching M in G of the maximum size



Reduce Max. Bipartite Matching to Max. Flow



- The maximum flow $\leftrightarrow$ maximum matching
- Need to use the fact that the maximum flow has integer flow values, if all capacities are integers.

Solving Bipartite Matching via Linear Programming

Integer Programming

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

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

$$x_e \in \{0, 1\} \qquad \forall e \in E$$

Linear Programming

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

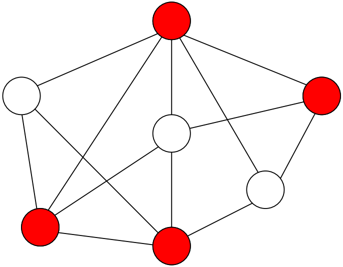
$$\sum_{e \in \delta(v)} x_e \leq 1 \qquad \forall v \in L \cup R$$

$$x_e \in [0, 1] \qquad \forall e \in E$$

Lemma The above integer programming and linear programming are equivalent.

Outline

Def. Given a graph $G = (V, E)$, a vertex cover of G is a subset $S \subseteq V$ such that for every $(u, v) \in E$ then $u \in S$ or $v \in S$.



Weighted Vertex-Cover Problem

Input: $G = (V, E)$ with vertex weights $\{w_v\}_{v \in V}$

Output: a vertex cover S with minimum $\sum_{v \in S} w_v$

Integer Programming for Weighted Vertex Cover

- For every $v \in V$, let $x_v \in \{0,1\}$ indicate whether we select v in the vertex cover S
- The integer programming for weighted vertex cover:

$$
\begin{array}{llll}
(\text{IP}_{\text{WVC}}) & \min & \displaystyle\sum_{v \in V} w_v x_v & \text{s.t.} \\
& & x_u + x_v \geq 1 & \forall (u,v) \in E \\
& & x_v \in \{0,1\} & \forall v \in V
\end{array}
$$

- $(\text{IP}_{\text{WVC}}) \Leftrightarrow$ weighted vertex cover
- Thus it is NP-hard to solve integer programmings in general

- Integer programming for WVC:

$$(\text{IP}_{\text{WVC}}) \qquad \min \quad \sum_{v\in V} w_v x_v \qquad \text{s.t.}$$
$$x_u + x_v \geq 1 \qquad \forall (u,v)\in E$$
$$x_v \in \{0,1\} \qquad \forall v \in V$$

- Linear programming relaxation for WVC:

$$(\text{LP}_{\text{WVC}}) \qquad \min \quad \sum_{v\in V} w_v x_v \qquad \text{s.t.}$$
$$x_u + x_v \geq 1 \qquad \forall (u,v)\in E$$
$$x_v \in [0,1] \qquad \forall v \in V$$

- let IP = value of (IP_{WVC}), LP = value of (LP_{WVC})
- Then, LP $\leq$ IP

Algorithm for Weighted Vertex Cover

Algorithm for Weighted Vertex Cover

1. Solving (LP_{WVC}) to obtain a solution $\{x^*_u\}_{u\in V}$
2. Thus, LP $= \sum_{u\in V} w_u x^*_u \leq$ IP
3. Let $S = \{u \in V : x_u \geq 1/2\}$ and output S

Lemma S is a vertex cover of G.

Proof.

- Consider any edge $(u, v) \in E$: we have $x^*_u + x^*_v \geq 1$
- Thus, either $x^*_u \geq 1/2$ or $x^*_v \geq 1/2$
- Thus, either $u \in S$ or $v \in S$. $\square$

Algorithm for Weighted Vertex Cover

Algorithm for Weighted Vertex Cover

1. Solving (LP_{WVC}) to obtain a solution $\{x^*_u\}_{u \in V}$
2. Thus, $\text{LP} = \sum_{u \in V} w_u x^*_u \leq \text{IP}$
3. Let $S = \{u \in V : x_u \geq 1/2\}$ and output S

Lemma S is a vertex cover of G.

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

Proof.

$$\begin{aligned} \text{cost}(S) &= \sum_{u \in S} w_u \leq \sum_{u \in S} w_u \cdot 2x^*_u = 2\sum_{u \in S} w_u \cdot x^*_u \\ &\leq 2\sum_{u \in V} w_u \cdot x^*_u = 2 \cdot \text{LP}. \end{aligned}$$

□

Algorithm for Weighted Vertex Cover

Algorithm for Weighted Vertex Cover

1. Solving (LP_{WVC}) to obtain a solution $\{x^*_u\}_{u \in V}$
2. Thus, $\text{LP} = \sum_{u \in V} w_u x^*_u \leq \text{IP}$
3. Let $S = \{u \in V : x^*_u \geq 1/2\}$ and output S

Lemma S is a vertex cover of G.

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

Theorem Algorithm is a 2-approximation algorithm for WVC.

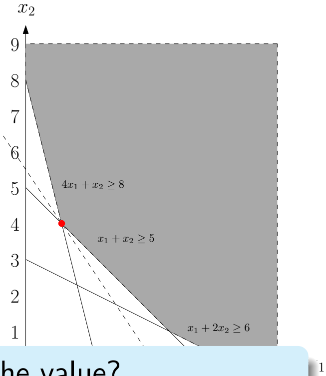
Proof.

$\text{cost}(S) \leq 2 \cdot \text{LP} \leq 2 \cdot \text{IP} = 2 \cdot \text{cost(best vertex cover)}$. □

Outline

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

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



Q: How can we prove a lower bound for the value?

- $7x_1 + 4x_2 \geq 2(x_1 + x_2) + (x_1 + 2x_2) \geq 2 \times 5 + 6 = 16$
- $7x_1 + 4x_2 \geq (x_1 + 2x_2) + 1.5(4x_1 + x_2) \geq 6 + 1.5 \times 8 = 18$
- $7x_1 + 4x_2 \geq (x_1 + x_2) + (x_1 + 2x_2) + (4x_1 + x_2) \geq 5 + 6 + 8 = 19$
- $7x_1 + 4x_2 \geq 4(x_1 + x_2) \geq 4 \times 5 = 20$
- $7x_1 + 4x_2 \geq 3(x_1 + x_2) + (4x_1 + x_2) \geq 3 \times 5 + 8 = 23$

Primal LP

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

Dual LP

$$\max \quad 5y_1 + 6y_2 + 8y_3 \quad \text{s.t.}$$

$$\begin{aligned} y_1 + y_2 + 4y_3 &\leq 7 \\ y_1 + 2y_2 + y_3 &\leq 4 \\ y_1, y_2 &\geq 0 \end{aligned}$$

A way to prove lower bound on the value of primal LP

$$\begin{aligned} & 7x_1 + 4x_2 \qquad (\text{if } 7 \geq y_1 + y_2 + 4y_3 \text{ and } 4 \geq y_1 + 2y_2 + y_3) \\ \geq\ & y_1(x_1 + x_2) + y_2(x_1 + 2x_2) + y_3(4x_1 + x_2) \quad (\text{if } y_1, y_2, y_3 \geq 0) \\ \geq\ & 5y_1 + 6y_2 + 8y_3. \end{aligned}$$

- Goal: need to maximize $5y_1 + 6y_2 + 8y_3$

Primal LP

$$\min \quad 7x_1 + 4x_2$$
$$x_1 + x_2 \geq 5$$
$$x_1 + 2x_2 \geq 6$$
$$4x_1 + x_2 \geq 8$$
$$x_1, x_2 \geq 0$$

Dual LP

$$\max \quad 5y_1 + 6y_2 + 8y_3 \quad \text{s.t.}$$
$$y_1 + y_2 + 4y_3 \leq 7$$
$$y_1 + 2y_2 + y_3 \leq 4$$
$$y_1, y_2 \geq 0$$

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 4 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 5 \\ 6 \\ 8 \end{pmatrix} \quad c = \begin{pmatrix} 7 \\ 4 \end{pmatrix}$$

$$\min \quad c^T x \quad \text{s.t.}$$
$$Ax \geq b$$
$$x \geq 0$$

$$\max \quad b^T y \quad \text{s.t.}$$
$$A^T y \leq c$$
$$y \geq 0$$

Primal LP

$$\min \quad c^T x \qquad \text{s.t.}$$
$$Ax \geq b$$
$$x \geq 0$$

Dual LP

$$\max \quad b^T y \qquad \text{s.t.}$$
$$A^T y \leq c$$
$$y \geq 0$$

- $P =$ value of primal LP
- $D =$ value of dual LP

Theorem (weak duality theorem) $D \leq P$.

Theorem (strong duality theorem) $D = P$.

- Can always prove the optimality of the primal solution, by adding up primal constraints.

Example

Primal LP

$$\min \quad 5x_1 + 6x_2 + x_3 \quad \text{s.t.}$$

$$\begin{aligned} 2x_1 + 5x_2 - 3x_3 &\geq 2 \\ 3x_1 - 2x_2 + x_3 &\geq 5 \\ x_1 + 2x_2 + 3x_3 &\geq 7 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

Dual LP

$$\max \quad 2y_1 + 5y_2 + 7y_3 \quad \text{s.t.}$$

$$\begin{aligned} 2y_1 + 3y_2 + y_3 &\leq 5 \\ 5y_1 - 2y_2 + 2y_3 &\leq 6 \\ -3y_1 + y_2 + 3y_3 &\geq 1 \\ y_1, y_2, y_3 &\geq 0 \end{aligned}$$

Primal Solution

$$x_1 = 1.6, x_2 = 0.6$$

$$x_3 = 1.4, \text{value} = 13$$

Dual Solution

$$y_1 = 1, y_2 = 5/8$$

$$y_3 = 9/8, \text{value} = 13$$

$$
\begin{aligned}
&\quad 5x_1 + 6x_2 + x_3 \\
&\geq (2x_1 + 5x_2 - 3x_3) + \frac{5}{8}(3x_1 - 2x_2 + x_3) + \frac{9}{8}(x_1 + 2x_2 + 3x_3) \\
&\geq 2 + \frac{5}{8} \times 5 + \frac{9}{8} \times 7 \\
&= 13
\end{aligned}
$$