上海科技大学
ShanghaiTech University

CS240 Algorithm Design and Analysis

Lecture 26

Approximation Algorithms

Quan Li

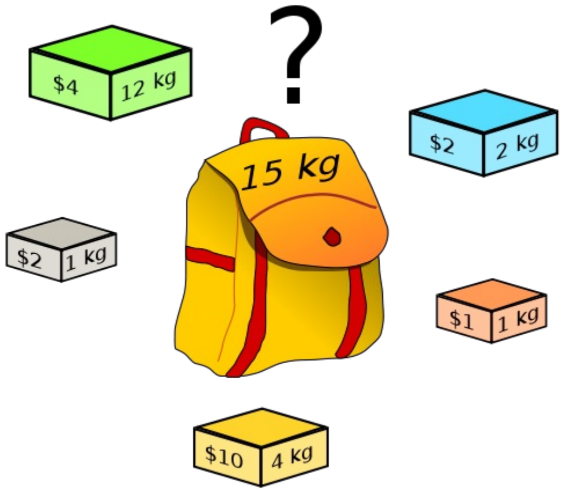
Fall 2024

2024.12.26

The Knapsack Problem

The Knapsack Problem

- We have a set of items, each having a weight and a value.
- We have a knapsack that can carry up to W amount of weight.
- We want to put items in the knapsack to maximize the total value, but not exceed the weight limit.
- **Ex** Items 3 and 4 are the highest value items with weight ≤ 11.
- Assume all items have weight $\leq$ W, i.e., any single item fits in knapsack.



W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

A Dynamic Programming for Knapsack

- Let OPT(i,v) = minimum weight of a subset of items 1,…,i that has value ≥ v.
- If optimal solution uses item i.
  - Then we pay w_i weight for item i and need to achieve value ≥ $v-v_i$ using items 1,…,i-1 using min weight.
  - So $OPT(i,v)=w_i+OPT(i-1,v-v_i)$.
- If optimal solution doesn't use item i.
  - Then we need to achieve value ≥ v using items 1,…,i-1.
  - So $OPT(i,v)=OPT(i-1,v)$.
- Choose the case that gives smaller weight.
- $$OPT(i,v) = \begin{cases} 0 & \text{if } v=0 \\ \infty & \text{if } i=0,\ v>0 \\ \min(OPT(i-1,v),\ w_i+OPT(i-1,v-v_i)) & \text{otherwise} \end{cases}$$

Running Time of Dynamic Programming

- Say there are n items, and the largest value of any item is v*.
- The max value we can pack into the knapsack is nv*, where v* is the largest v value.
- Solve all subproblems of the form OPT(i,v), where i $\leq$ n and v $\leq$ nv*.
  - This is a total of $O(n^2v^*)$ subproblems.
- The solution to Knapsack is the max value V that can be packed with weight $\leq$ W.
- Having solved all the subproblems, we can find V by finding the subproblem with the largest value that has optimum weight $\leq$ W.
  - $V = max_{v \leq nv^*} OPT(n, v) \leq W.$
- So solving Knapsack takes total time $O(n^2v^*)$.

Running Time of Dynamic Programming

- The DP gives an optimal solution to Knapsack and takes $O(n^2v^*)$ time. Have we found a polytime algorithm for an NP-complete problem?
- No. The problem size is $O(n \log(v^*))$, because it takes $\log(v^*)$ bits to express each item's value. But $O(n^2v^*)$ is not polynomial in $n \log(v^*)$.
- To make this DP fast, we have to make the largest value small.

PTAS (Polynomial Time Approximation Scheme)

- Let $\varepsilon>0$ be any number. We'll give a $(1+\varepsilon)$-approximation for knapsack.
- By setting ε sufficiently small, we can get as good an approximation as we want!
  - This type of algorithm is called a polynomial time approximation scheme, or PTAS.
- Contrast this with earlier algorithms we studied, which had worse approximation ratios, e.g., 2 or log n.
- But the running time will be $O(n^3/\varepsilon)$. Hence, we can't set $\varepsilon=0$ get the optimal solution.
- We're trading accuracy for time. The more accurate (smaller ε), the more time the algorithm takes.

Main Idea: Rounding

- Since we only need an approximate solution, we can change the values of the items a little (round the values) and not affect the solution much.
- We scale and round the original values to make them small.
- The previous DP took $O(n^2v^*)$ time. So if the rounded values are small, this DP is fast.

W = 11

Item	Value	Weight
1	134,221	1
2	656,342	2
3	1,810,013	5
4	22,217,800	6
5	28,343,199	7

W = 11

Item	Value	Weight
1	2	1
2	7	2
3	19	5
4	23	6
5	29	7

Rounding

- Let ε >0 be the precision we want.
- Set θ = ε v*/2n to be a scaling factor.
  - v* is the largest value of any item.
- Scale all values down by q then round up.
  - $v' = \lceil v/\theta \rceil$.
- Make a problem where each value v_i is replaced by v'_i.
  - Call this the scaled rounded problem.
- Let v^ be max value in the scaled rounded problem. Then v^ = $\lceil v*/\theta \rceil$ = $\left\lceil v*/(\frac{\varepsilon v*}{2n}) \right\rceil = \lceil 2n/\varepsilon \rceil$.
- Running time of DP on scaled rounded problem is O(n^2v^) =O(n^3/ε).

Solving the Original Problem

- Make another new problem in which each value v_i is replaced by $u_i = \lceil v_i/\theta \rceil * \theta$.
  - Call this the rounded problem.
  - We have $u_i \geq v_i$, and $u_i \leq v_i + \theta$.
- Note u values are equal to v′ values multiplied by θ.
  - Thus, the optimal solution for the rounded problem and the scaled rounded problem are the same.
- We now have 3 problems, the original problem, the scaled rounded problem, and the rounded problem.
- Let S be the optimal solution to the scaled rounded problem, which we can find in time $O(n^3/\varepsilon)$. S is also optimal for the rounded problem.
- We'll show S is a $1+\varepsilon$ approximation for the original problem.

Correctness

- **Thm** Let S* be the optimal solution to the original problem. Then $(1+\varepsilon)\sum_{i\in S} v_i \geq \sum_{i\in S^*} v_i$

Hence S is a (1+ε)-approximate solution.

- **Proof**

$$\sum_{i\in S^*} v_i \leq \sum_{i\in S^*} u_i$$ $u_i \geq v_i$

$$\leq \sum_{i\in S} u_i$$ S is optimal solution for rounded problem

$$\leq \sum_{i\in S} (v_i + \theta)$$ $u_i \leq v_i+\theta$

$$\leq \sum_{i\in S} v_i + n\theta$$ $|S| \leq n$

立志成才报国裕民

Correctness

- Suppose item j has the largest value, so $v^*=v_j$. Then $n\theta = \frac{\varepsilon}{2} v_j \le \frac{\varepsilon}{2} u_j \le \frac{\varepsilon}{2} \sum_{i \in S} u_i$
  - Last inequality because item j itself is feasible solution, so opt solution S is no smaller.
- So $\sum_{i \in S} v_i \ge \sum_{i \in S} u_i - n\theta \ge \left(\frac{2}{\varepsilon} - 1\right) n\theta$, where first inequality comes from last page.
- Assuming $\varepsilon \le 1$, then $n\theta \le \varepsilon \sum_{i \in S} v_i$
- Finally, we have

$$\sum_{i \in S^*} v_i \le \sum_{i \in S} v_i + n\theta \le \sum_{i \in S} v_i + \varepsilon \sum_{i \in S} v_i = (1+\varepsilon) \sum_{i \in S} v_i$$

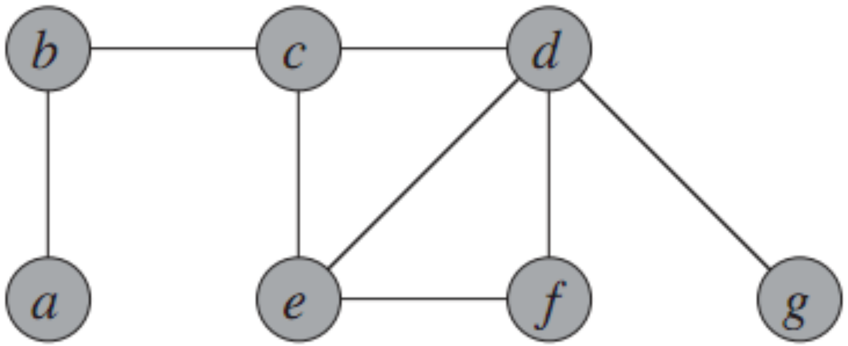
Summary

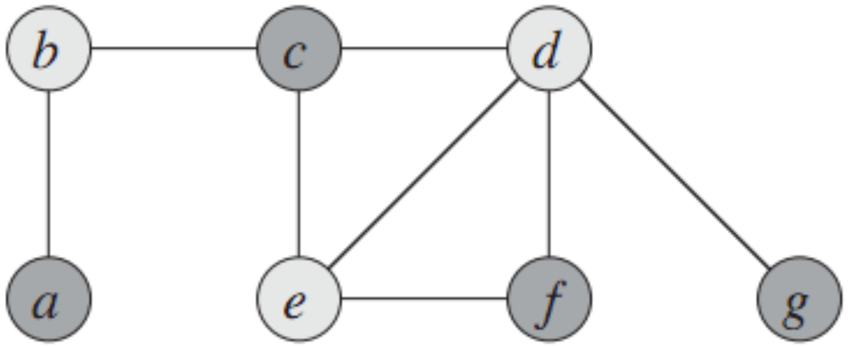
- We gave a DP for Knapsack.
- We scale and round to reduce number of different item values.
- Running the DP on the scaled rounded problem and using the solution for the original problem leads to an arbitrarily good approximation for Knapsack, a PTAS.
- There are PTAS's for a number of other problems.
  - Multiprocessor scheduling.
  - Bin packing.
  - Euclidean TSP.
- However, there are also many problems for which PTAS's do not exist, unless P=NP.

Vertex Cover

Vertex Cover

- **Input** A graph with vertices V and edges E.
- **Output** A subset V′ of the vertices, so that every edge in E touches some vertex in V′.
- **Goal** Make |V′| as small as possible.



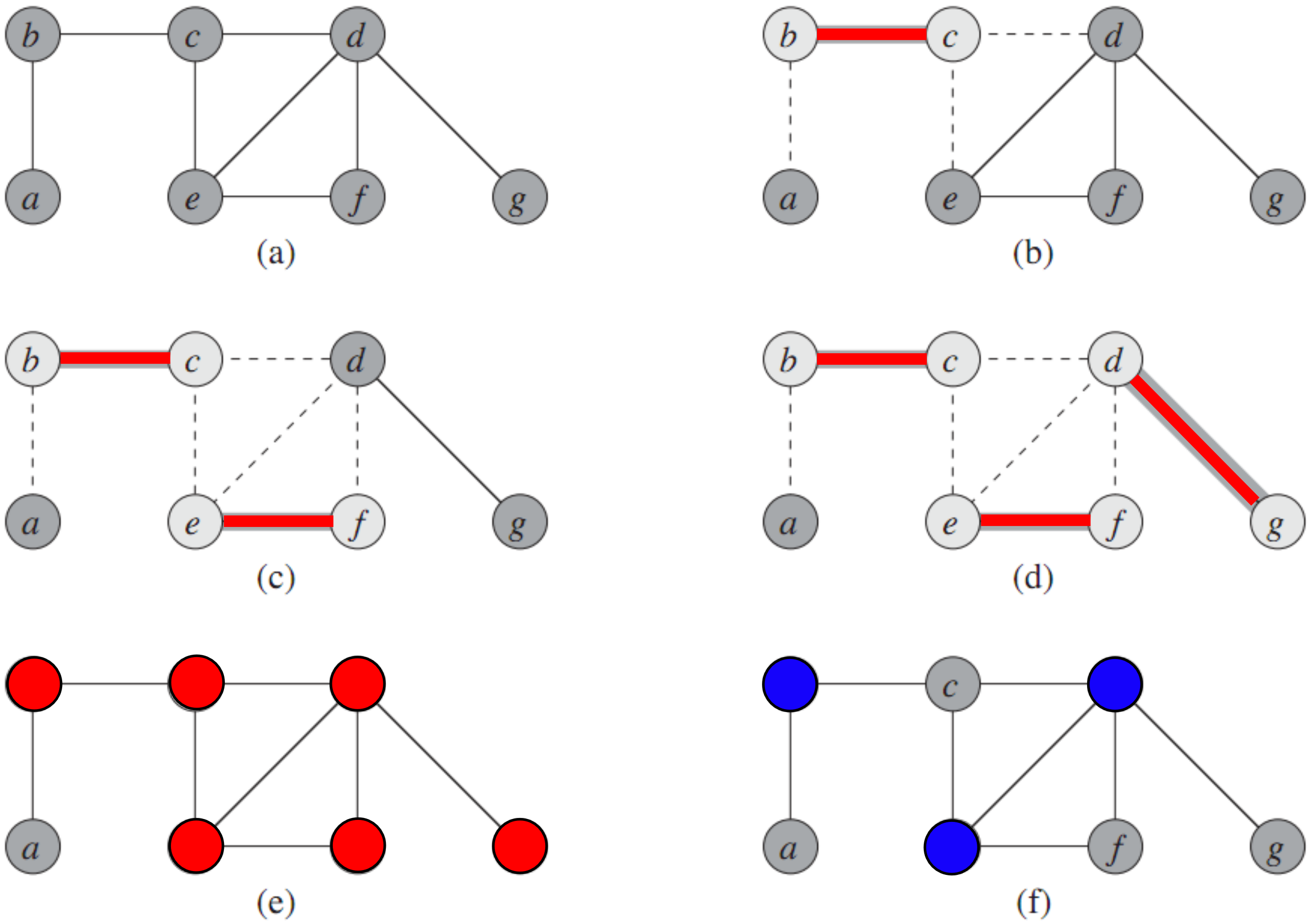


- Finding the minimum vertex cover is NP-complete.
- We'll see a simple 2 approximation for this problem.

A Vertex Cover Algorithm

- Initially, let D be all the edges in the graph, and C be the empty set.
  - C is our eventual vertex cover.
- Repeat as long as there are edge left in D.
  - Take any edge (u,v) in D.
  - Add {u,v} to C.
  - Remove all the edges adjacent to u or v from D.
- Output C as the vertex cover.

Example



Algorithm's vertex cover

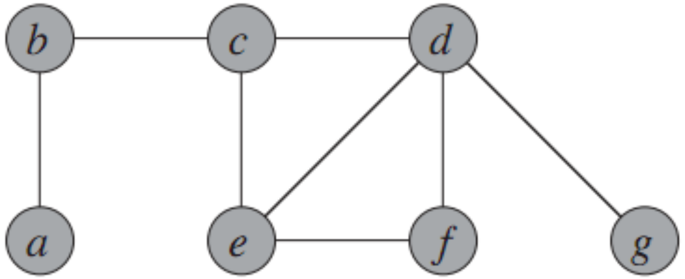
Optimal vertex cover

Proof of Correctness

- The output is certainly a vertex cover.
  - In each iteration, we only take out edges that get covered.
  - We keep adding vertices till all edges are covered.
- Now, we show it's a 2 approximation.
- Let C^* be an optimal vertex cover.
- Let A be the set of edges the algorithm picked.

Proof of Correctness

- None of the edges in A touch each other.
  - Each time we pick an edge, we remove all adjacent edges.
- So each vertex in C^* covers at most one edge in A.
  - The edges covered by a vertex all touch each other.
- Every edge in A is covered by a vertex in C^*.
  - Because C^* is a vertex cover.
- So $|C^*| \geq |A|$.
- The number of vertices the algorithm uses is $2|A|$.
  - If algorithm picks edge (u,v), it uses {u,v} in the cover.
- So (# vertices algorithm uses) / (# vertices in opt cover) = $2|A| / |C^*| \leq 2|A| / |A| = 2$.

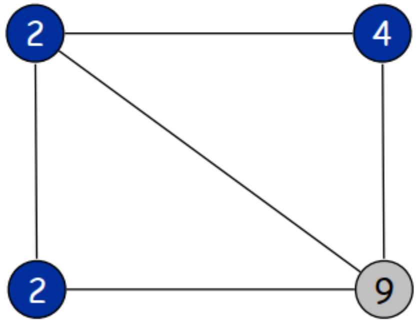


The Pricing Method: Vertex Cover

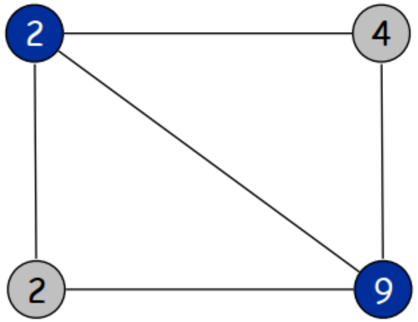
Weighted Vertex Cover

Weighted vertex cover. Given a graph G with vertex weights, find a vertex cover of minimum weight.

It's a special case of the set cover problem, so the $H(d^*)$ approximation ratio can be achieved by the greedy algorithm, where d^* = max degree



weight = 2 + 2 + 4 = 8



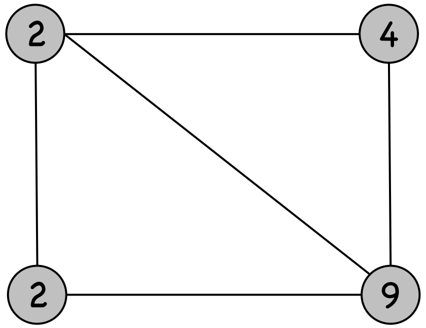
weight = 2 + 9 = 11

Weighted Vertex Cover

Pricing method. Each edge must be covered by some vertex i. Edge e pays price $p_e \geq 0$ to use vertex i.

Fairness. Edges incident to vertex i should pay $\leq w_i$ in total.

$$for\ each\ vertex\ i: \sum_{e=(i,j)} p_e \leq w_i$$



Claim. For any vertex cover S and any fair prices p_e: $\sum_e p_e \leq w(S)$.

Proof.

$$\sum_{e \in E} p_e \leq \sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in S} w_i = w(S) \quad \blacksquare$$

each edge e covered by at least one node in S

sum fairness inequalities for each node in S

Pricing Method

Pricing method. Set prices and find vertex cover simultaneously.

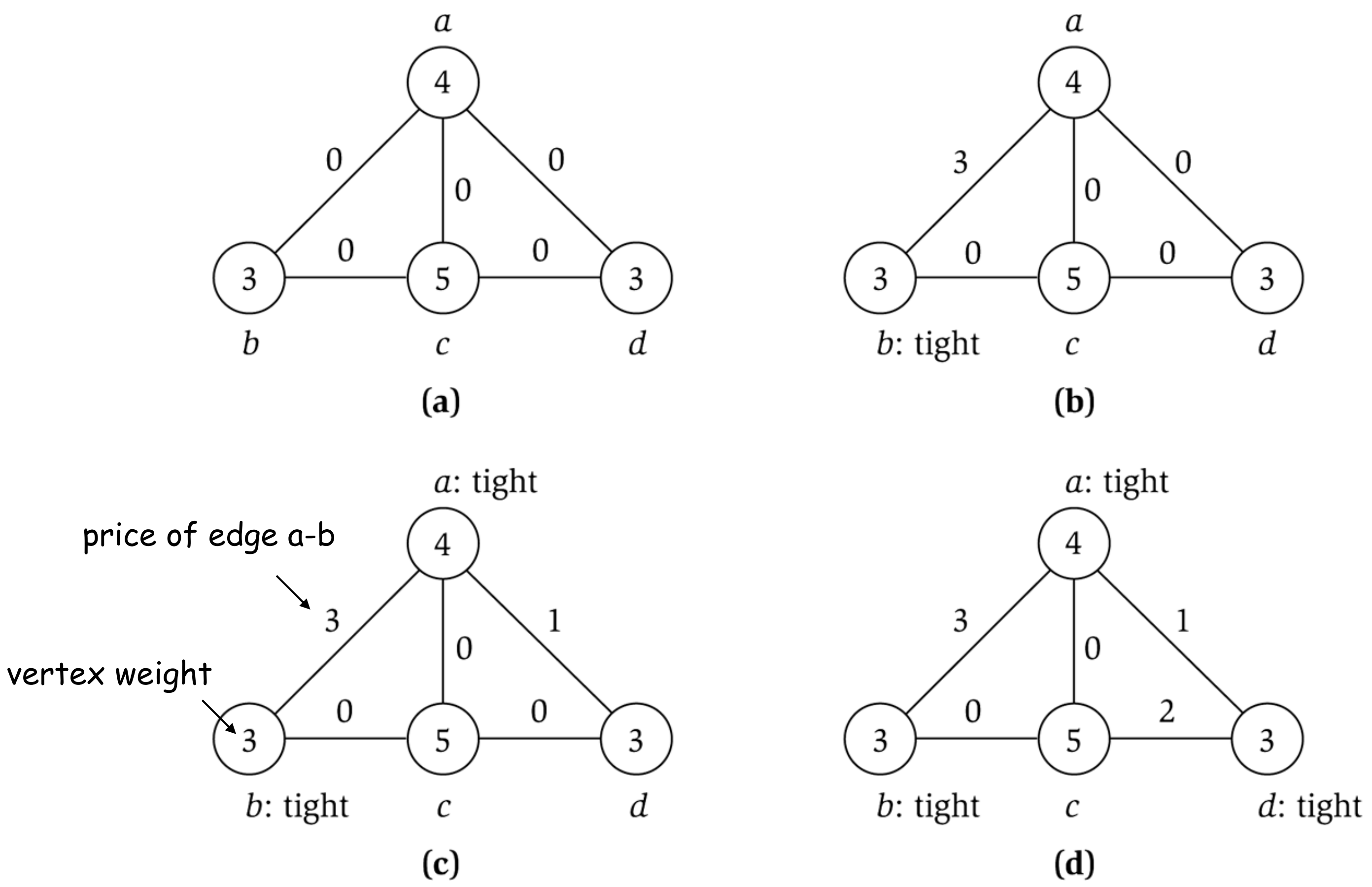
$$\sum_{e=(i,j)} p_e = w_i$$

```
Weighted-Vertex-Cover-Approx(G, w) {
   foreach e in E
      pe = 0

   while (∃ edge i-j such that neither i nor j are tight)
      select such an edge e
      increase pe without violating fairness
   }

   S ← set of all tight nodes
   return S
}
```

Pricing Method: Example



Pricing Method: Analysis

Theorem. Pricing method is a 2-approximation.

Pf.

- Algorithm terminates since at least one new node becomes tight after each iteration of while loop.

- Let S = set of all tight nodes upon termination of algorithm.
- S is a vertex cover: if some edge i-j is uncovered, then neither i nor j is tight. But then while loop would not terminate.

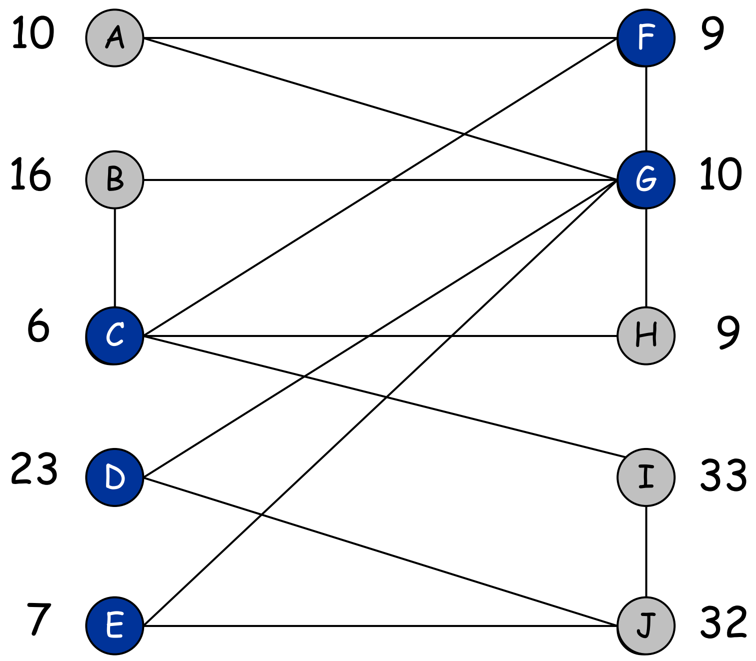
- Let S* be optimal vertex cover. We show w(S) ≤ 2w(S*).

$$w(S) = \underbrace{\sum_{i\in S} w_i = \sum_{i\in S}\sum_{e=(i,j)} p_e}_{\text{all nodes in S are tight}} \;\underbrace{\leq}_{S\subseteq V,\ \text{prices}\geq 0}\; \sum_{i\in V}\sum_{e=(i,j)} p_e \;\underbrace{=}_{\text{each edge counted twice}}\; 2\sum_{e\in E} p_e \;\underbrace{\leq}_{\text{fairness lemma}}\; 2w(S^*). \quad \blacksquare$$

LP Rounding: Vertex Cover

Weighted Vertex Cover

Weighted vertex cover. Given an undirected graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a minimum weight subset of nodes S such that every edge is incident to at least one vertex in S.



total weight = 55

Weighted Vertex Cover: Integer Linear Programming Formulation

Weighted vertex cover. Given an undirected graph G = (V, E) with vertex weights $w_i \geq 0$, find a minimum weight subset of nodes S such that every edge is incident to at least one vertex in S.

Integer programming formulation.

- Model inclusion of each vertex i using a 0/1 variable x_i.

$$x_i = \begin{cases} 0 & \text{if vertex } i \text{ is not in vertex cover} \\ 1 & \text{if vertex } i \text{ is in vertex cover} \end{cases}$$

- Objective function: minimize $\Sigma_i w_i x_i$.

- Must take either i or j: $x_i + x_j \geq 1$.

$$\begin{array}{llllll} (ILP) & \min & \displaystyle\sum_{i \in V} w_i x_i & & & \\ & \text{s.t.} & x_i + x_j & \geq & 1 & (i,j) \in E \\ & & x_i & \in & \{0,1\} & i \in V \end{array}$$

Integer Programming

INTEGER-PROGRAMMING. Given integers a_{ij} and b_i, find integers x_j that satisfy:

$$\begin{array}{llcl} \min & c^t x & & \\ \text{s. t.} & Ax & \geq & b \\ & x & \geq & 0 \\ & x \text{ integral} & & \end{array}$$

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i & 1 \leq i \leq m \\ x_j & \geq & 0 & 1 \leq j \leq n \\ x_j & & \text{integral} & 1 \leq j \leq n \end{array}$$

Observation. Vertex cover formulation proves that integer programming is NP-hard search problem.

even if all coefficients are 0/1 and
at most two variables per inequality

Integer Programming

上海科技大学 ShanghaiTech University

Linear programming. Max/min linear objective function subject to linear inequalities.

- Input: integers c_j, b_i, a_{ij} .
- Output: **real numbers** x_j.

$$\begin{array}{llrcl} \text{(LP)} & \min & c^t x & & \\ & \text{s.t.} & Ax & \geq & b \\ & & x & \geq & 0 \end{array}$$

$$\begin{array}{llrcll} \text{(LP)} & \min & \sum_{j=1}^{n} c_j x_j & & & \\ & \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i & 1 \leq i \leq m \\ & & x_j & \geq & 0 & 1 \leq j \leq n \end{array}$$

Linear. No x^2, xy, $\arccos(x)$, $x(1-x)$, etc.

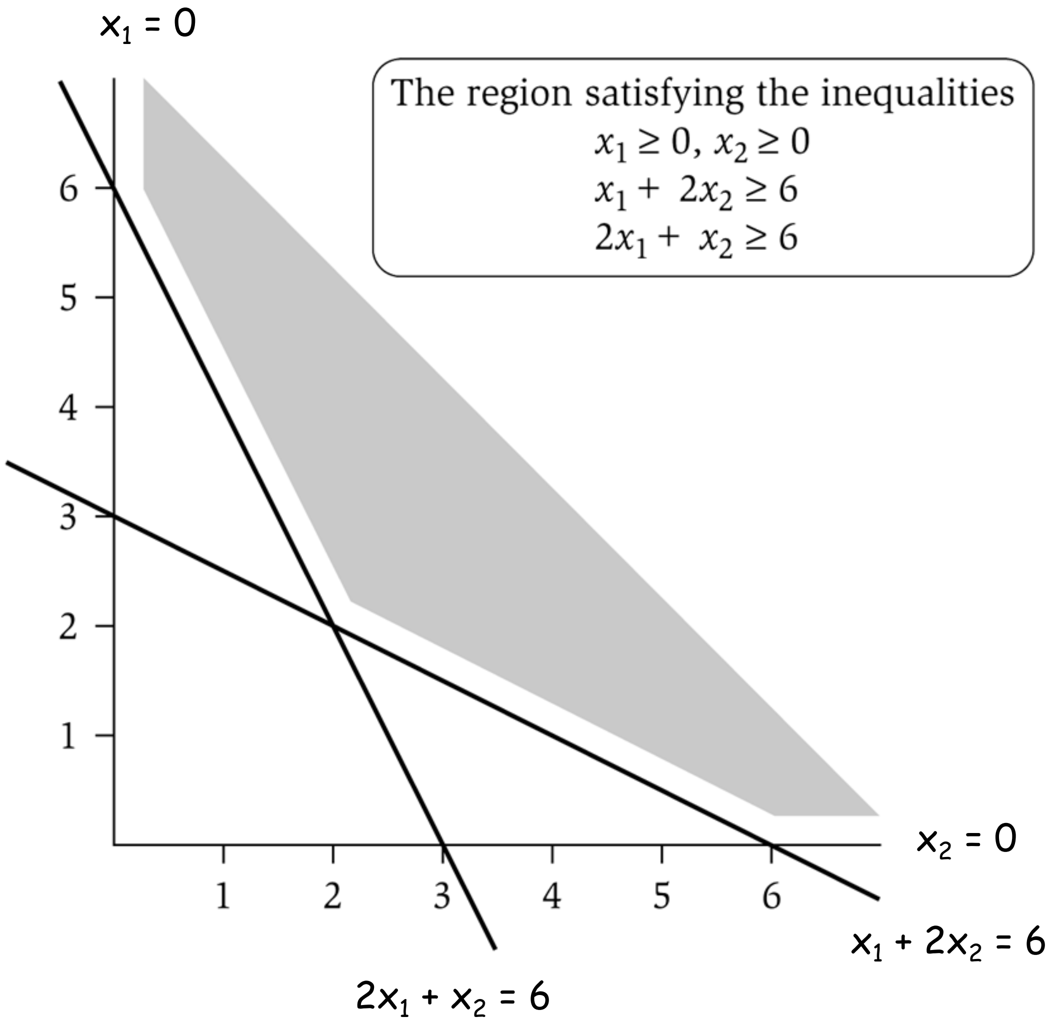
Simplex algorithm. [Dantzig 1947] Can solve LP in practice.
Ellipsoid algorithm. [Khachian 1979] Can solve LP in poly-time.

立志成才 报国裕民

LP Feasible Region

上海科技大学 ShanghaiTech University

LP geometry in 2D.



立志成才报国裕民

Weighted Vertex Cover: LP Relaxation

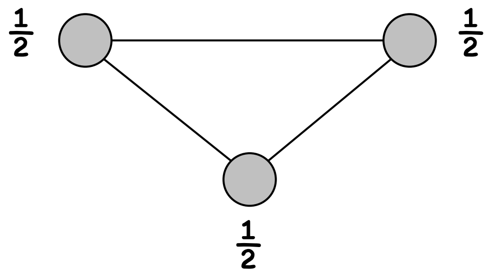
Weighted vertex cover. Linear programming formulation.

$$
\begin{array}{llll}
(LP) \ \min & \sum_{i \in V} w_i x_i & & \\
\text{s.t.} & x_i + x_j & \geq 1 & (i,j) \in E \\
& x_i & \geq 0 & i \in V
\end{array}
$$

Observation. Optimal value of (LP) is $\leq$ optimal value of (ILP).

Pf. LP has fewer constraints.

Note. LP is not equivalent to vertex cover.



Q. How can solving LP help us find a small vertex cover?

A. Solve LP and **round** fractional values.

Weighted Vertex Cover

Theorem. If x^* is optimal solution to (LP), then $S = \{i \in V : x^*_i \geq ½\}$ is a vertex cover whose weight is at most twice the min possible weight.

Pf. [S is a vertex cover]

- Consider an edge $(i, j) \in E$.
- Since $x^*_i + x^*_j \geq 1$, either $x^*_i \geq ½$ or $x^*_j \geq ½ \Rightarrow (i, j)$ covered.

Pf. [S has desired cost]

- Let S^* be optimal vertex cover. Then

$$\sum_{i \in S^*} w_i \underbrace{\geq}_{\text{LP is a relaxation}} \sum_{i \in S} w_i x_i^* \underbrace{\geq}_{x_i^* \geq \frac{1}{2}} \frac{1}{2} \sum_{i \in S} w_i$$