Algorithm Design and Analysis

CSE 565

LECTURE 29

Approximation Algorithms

- Load Balancing
- Weighted Vertex Cover

Reminder:

Fill out SRTEs online

- Don’t forget to click *submit*

Sofya Raskhodnikova

Approximation algorithm?

- Suppose a minimization problem is NP hard
  - Cannot find a polynomial-time algorithm that finds optimal solution on every instance
  - What if I can guarantee that my algorithm's solution is within 5% of optimal? That is,
    - Minimization problems: $OPT \le output \le 1.05 \times OPT$
    - Maximization problems: $\frac{OPT}{1.05} \le output \le OPT$
  - Good enough?
    - Depends on context
    - If data is already noisy or we don't know exact cost function, then approximation might be fine.

NP-Completeness as a Design Guide

Q. Suppose I need to solve an NP-complete problem. What should I do?
A. You are unlikely to find poly-time algorithm that works on all inputs.

Must sacrifice one of three desired features.

- Solve problem in polynomial time (→ *e.g.*, fast exponential algorithms)
- Solve arbitrary instances of the problem
- Solve problem to optimality (→ approximation algorithms)

ρ-approximation algorithm.

- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio ρ of true optimum.

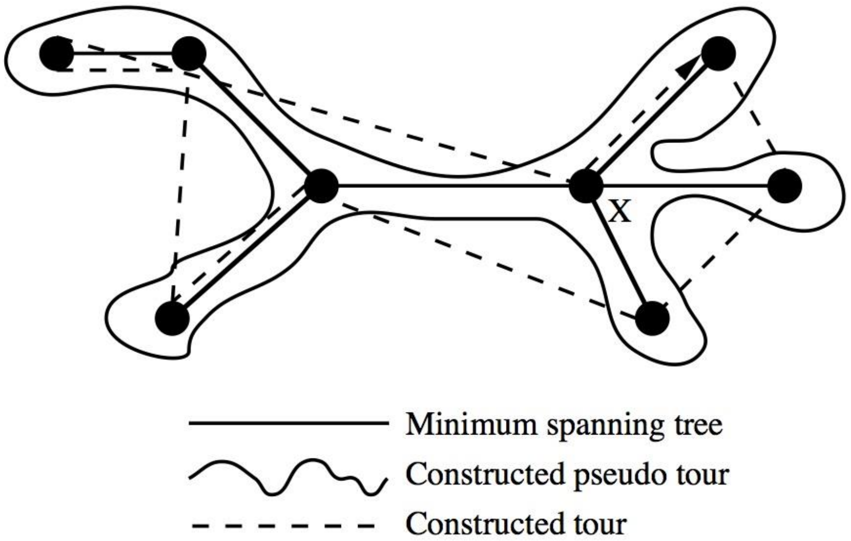
Challenge. Need to prove a solution's value is close to optimum, without even knowing what optimum value is!

Some Approximation Algorithms

- Knapsack
  - For any δ: there is a $(1+\delta)$-approximation in time $O(n/\delta)$ by rounding weights to multiples of δW
- Metric Traveling Salesman
  - Inorder traversal of MST is a 2-approx
  - Best known: $\frac{1+\sqrt{5}}{2}$-approximation
- Unweighted Vertex Cover
  - size of any maximal matching M is 2-approximation
  - (Sandwich: $|M| \le |\text{opt-VC}| \le 2|M|$)

Metric TSP

- Input: undirected G, non-negative edge lengths w
- Goal: path p that visits all vertices of minimum length
  - Metric TSP: Repeat visits are ok.
- Why metric?
  - Can always use shortest path to go from u to v
  - So may as well assume all edges are present and $w(u,v) = d(u,v)$ (where d = distance in original G)



- **Algorithm**: Look at MST.
  - Find MST T
  - Pick a root, order children of each vertex arbitrarily
  - Output: vertices along inorder traversal of T
- **Theorem**: Algorithm produces a 2-approximation to TSP

Metric TSP: Beyond the MST

- [Christofides 1976]: look at set of odd-degree vertices S
  - Add min-weight matching (Edmonds) on S
  - This adds OPT/2 to total weight since optimal tour of S can be written as union of two disjoint matchings
- [T. Momke and O. Svensson '11]
  - (1.46…)-approximation for special case of unweighted graphs (distances correspond to shortest paths in a graph with all edge lengths 1)

Load Balancing

Load Balancing

Input. m identical machines; n jobs, job j has processing time t_j.

- Job j must run contiguously on one machine.
- A machine can process at most one job at a time.

Def. Let J(i) be the subset of jobs assigned to machine i. The load of machine i is $L_i = \Sigma_{j \in J(i)} t_j$.

Def. The makespan is the maximum load on any machine $L = \max_i L_i$.

Load balancing. Assign each job to a machine to minimize makespan.

Hardness. Load Balancing is NP-complete even for 2 machines.
Exercise. Prove this statement.
Hint: to prove NP-hardness, reduce from Number Partitioning.

Today: approximation algorithms for Load Balalncing.

Load Balancing: List Scheduling

List-scheduling algorithm.

- Consider n jobs in some fixed order.
- Assign job j to machine whose load is smallest so far.

```
List-Scheduling(m, n, t1,t2,...,tn) {
   for i = 1 to m {
      Li ← 0                 ← load on machine i
      J(i) ← φ               ← jobs assigned to machine i
   }

   for j = 1 to n {
      i = argmink Lk         ← machine i has smallest load
      J(i) ← J(i) ∪ {j}      ← assign job j to machine i
      Li ← Li + tj           ← update load of machine i
   }
   return J
}
```

Implementation. O(n log n) using a priority queue.

Load Balancing: List Scheduling Analysis

Theorem. [Graham, 1966] Greedy algorithm is a 2-approximation.

- First worst-case analysis of an approximation algorithm.
- Need to compare resulting solution with optimal makespan L^*.

Lemma 1. The optimal makespan $L^* \geq \max_j t_j$.

Pf. Some machine must process the most time-consuming job. ▪

Lemma 2. The optimal makespan $L^* \geq \frac{1}{m} \sum_j t_j$.

Pf.

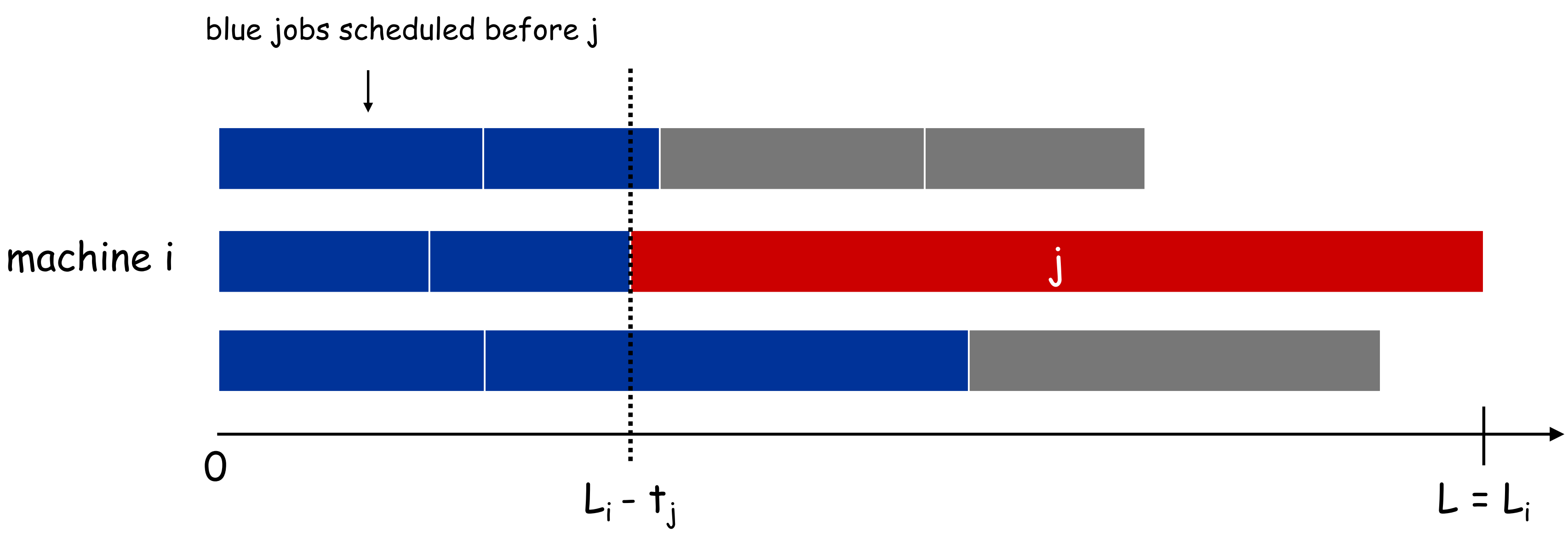
- The total processing time is $\sum_j t_j$.
- One of m machines must do at least a 1/m fraction of total work. ▪

Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load L_i of bottleneck machine i.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load. Its load before assignment is $L_i - t_j \Rightarrow L_i - t_j \leq L_k$ for all $1 \leq k \leq m$.



Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load L_i of bottleneck machine i.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load. Its load before assignment is $L_i - t_j \Rightarrow L_i - t_j \leq L_k$ for all $1 \leq k \leq m$.
- Sum inequalities over all k and divide by m:

$$\begin{aligned} L_i - t_j &\leq \tfrac{1}{m} \textstyle\sum_k L_k \\ &= \tfrac{1}{m} \textstyle\sum_k t_k \\ \text{Lemma 1} \longrightarrow \quad &\leq L^* \end{aligned}$$

- Now $$L_i = \underbrace{(L_i - t_j)}_{\leq L^*} + \underbrace{t_j}_{\leq L^*} \leq 2L^*. \quad ▪$$

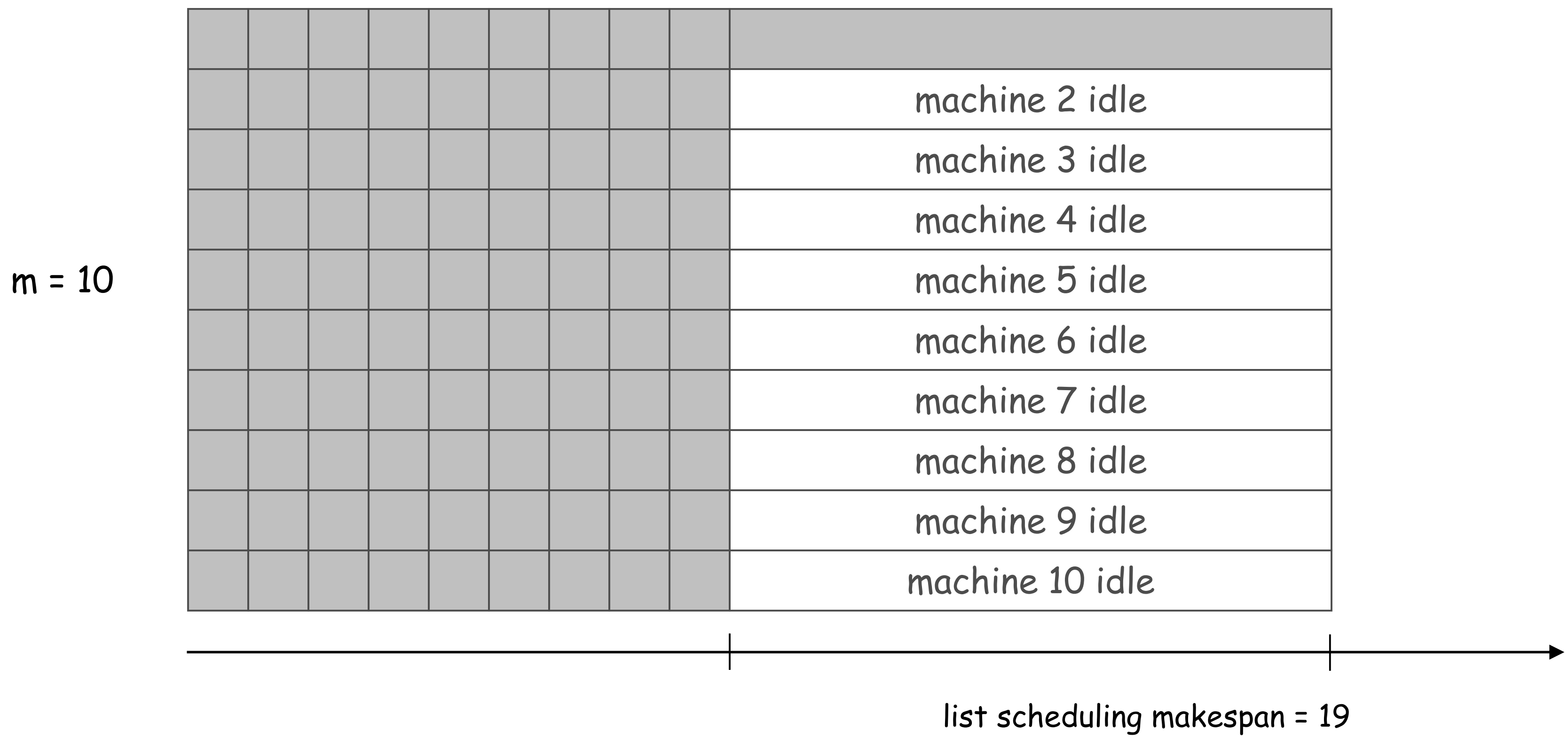
Lemma 2

Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?

A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m



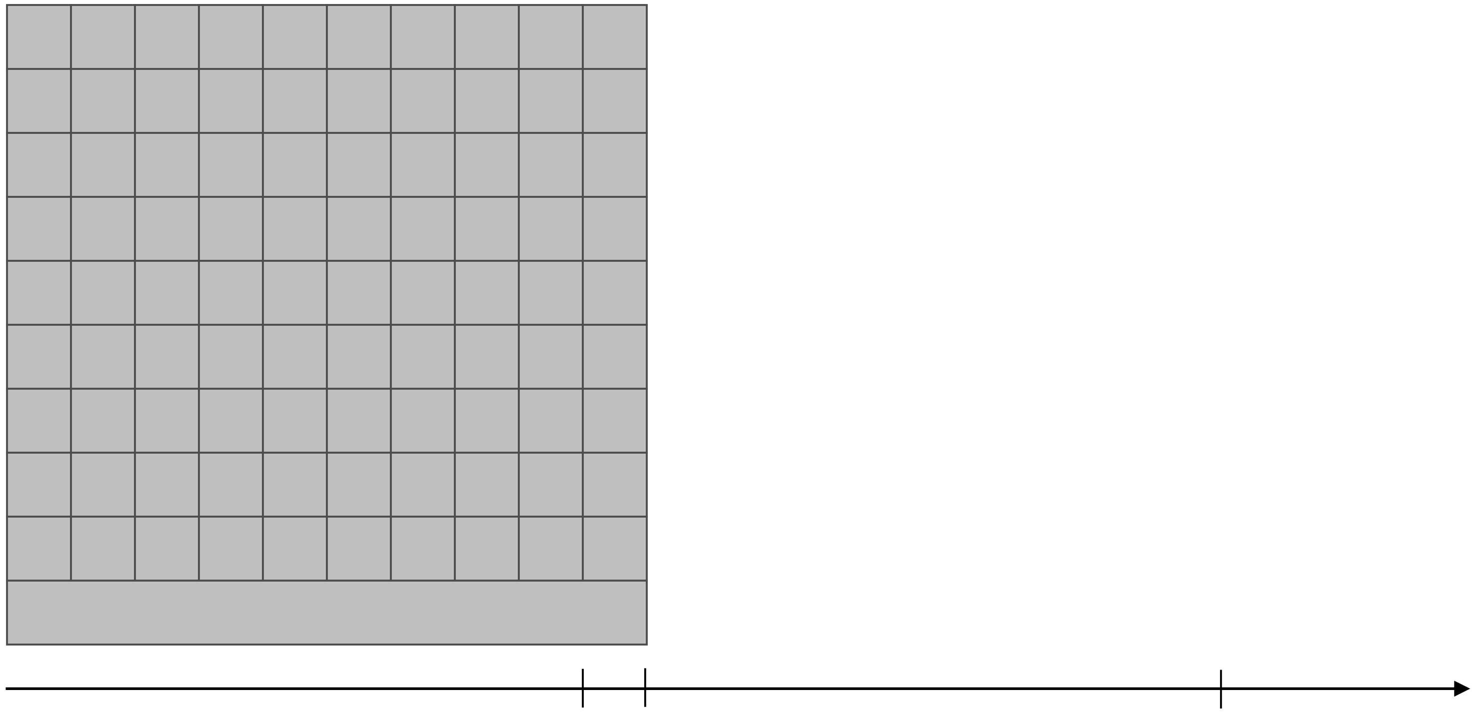
Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?

A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m

m = 10



optimal makespan = 10

Load Balancing: LPT Rule

Longest processing time (LPT). Sort n jobs in descending order of processing time, and then run list scheduling algorithm.

```
LPT-List-Scheduling(m, n, t1,t2,…,tn) {
   Sort jobs so that t1 ≥ t2 ≥ … ≥ tn

   for i = 1 to m {
      Li ← 0            ← load on machine i
      J(i) ← φ          ← jobs assigned to machine i
   }

   for j = 1 to n {
      i = argmink Lk        ← machine i has smallest load
      J(i) ← J(i) ∪ {j}     ← assign job j to machine i
      Li ← Li + tj          ← update load of machine i
   }
   return J
}
```

Load Balancing: LPT Rule

Observation. If at most m jobs, then list-scheduling is optimal.
Pf. Each job put on its own machine. ▪

Lemma 3. If there are more than m jobs, $L^* \geq 2\, t_{m+1}$.
Pf.

- Consider first m+1 jobs $t_1, \ldots, t_{m+1}$.
- Since the t_i's are in descending order, each takes at least t_{m+1} time.
- There are m+1 jobs and m machines, so by pigeonhole principle, at least one machine gets two jobs. ▪

Theorem. LPT rule is a 3/2 approximation algorithm.
Pf. Same basic approach as for list scheduling.

$$L_i = \underbrace{(L_i - t_j)}_{\leq\, L^*} + \underbrace{t_j}_{\leq\, \frac{1}{2} L^*} \leq \tfrac{3}{2} L^*. \quad \blacksquare$$

Lemma 3
(by observation, can assume number of jobs > m)

Load Balancing: LPT Rule

Q. Is our 3/2 analysis tight?
A. No.

Theorem. [Graham, 1969] LPT rule is a 4/3-approximation.
Pf. More sophisticated analysis of same algorithm.

Q. Is Graham's 4/3 analysis tight?
A. Essentially yes.

Ex: m machines, n = 2m+1 jobs, 2 jobs of length m+1, m+2, …, 2m-1 and one job of length m.
Exercise: Understand why.

Load Balancing: State of the Art

Polynomial Time Approximation Scheme (PTAS).

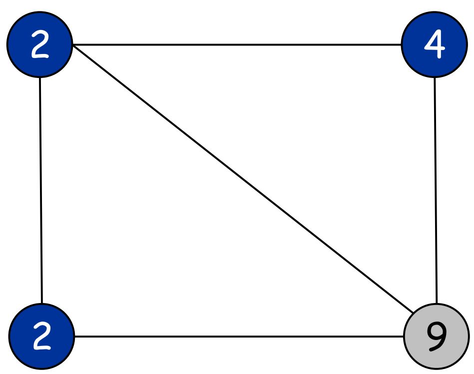
$(1 + \varepsilon)$-approximation algorithm for any constant $\varepsilon > 0$.
[Hochbaum-Shmoys 1987]

Consequence. PTAS produces arbitrarily high quality solution, but trades off accuracy for time.

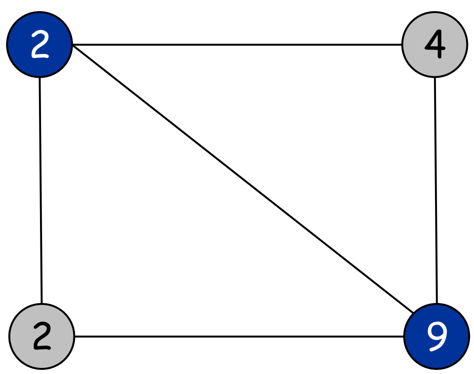
Weighted Vertex Cover

Weighted Vertex Cover

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



weight = 2 + 2 + 4



weight = 2+9=11

Linear Programming and Approximation

LP and Appoximation

- Major technique in ~~approximation~~ algorithms

1. Design a linear program whose integer solutions correspond to the problem you are solving
2. Solve the (fractional) linear program
3. Construct an integral solution from the fractional solution

Weighted Vertex Cover: IP 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.

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

  Vertex covers in 1-1 correspondence with 0/1 assignments:
  $S = \{i \in V : x_i = 1\}$

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

- Constraints for each edge (i,j): must take either i or j: $x_i + x_j \geq 1$.

Weighted Vertex Cover: IP Formulation

Weighted vertex cover. Integer programming formulation.

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

Observation. If x^* is optimal solution to (ILP), then $S = \{i \in V : x^*_i = 1\}$ is a min weight vertex cover.

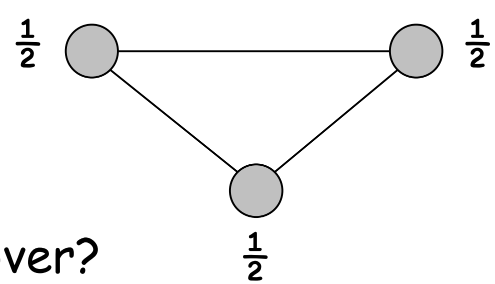
Weighted Vertex Cover: LP Relaxation

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 $S = \{i \in V : x^*_i \geq \frac{1}{2}\}$ 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 $x^*_i \geq \frac{1}{2}$ or $x^*_j \geq \frac{1}{2}$ $\Rightarrow$ (i, j) covered.

Pf. [S has desired cost]

- Let S^* be optimal vertex cover. Then

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

Weighted Vertex Cover

Theorem. 2-approximation algorithm for weighted vertex cover.

Theorem. [Dinur-Safra 2001] If P $\neq$ NP, then no ρ-approximation for $\rho < 1.3607$, even with unit weights.

$10\sqrt{5} - 21$

Theorem [Khot, Regev 2008]: If the "unique games conjecture" is true, then no ρ-approximation for $\rho < 2$, even with unit weights.

Open research problem. Close the gap.

Approximation algorithms summary

For many NP-complete problems, we can find “good” approximate solutions in polynomial time

- Often, we can prove an upper bound on the approximation factor

In practice, the techniques of approximation algorithms lead to very useful solutions

- For particular instances, the appoximation ratio is much better than proven worst-case guarantees