COT 6936: Topics in Algorithms

Giri Narasimhan

ECS 254A / EC 2443; Phone: x3748

giri@cs.fiu.edu

https://moodle.cis.fiu.edu/v2.1/course/view.php?id=**612**

Reading

- Read Background
  - Algorithms & Discrete Math Fundamentals
    - Cormen, et al., Chapters 1-16, 22-25
  - NP-Completeness
    - Cormen et al., Chapter 34
    - Appendix (p187-288) form Garey & Johnson
- Next Class
  - Approximation Algorithms
    - Cormen et al., Chapter 35
    - Kleinberg, Tardos, Chapter 11
    - Books by Vazirani and Hochbaum/Shmoys

What are *NP-Complete* problems?

- These are the hardest problems in *NP*.
- A problem p is *NP-Complete* if
  - there is a polynomial-time reduction from **<u>every</u>** problem in *NP* to p.
  - $p \in NP$
- How to prove that a problem is *NP-Complete*?

> - **Cook's Theorem**: [1972]
>   - The <u>SAT</u> problem is *NP-Complete*.

Steve Cook, Richard Karp, Leonid Levin

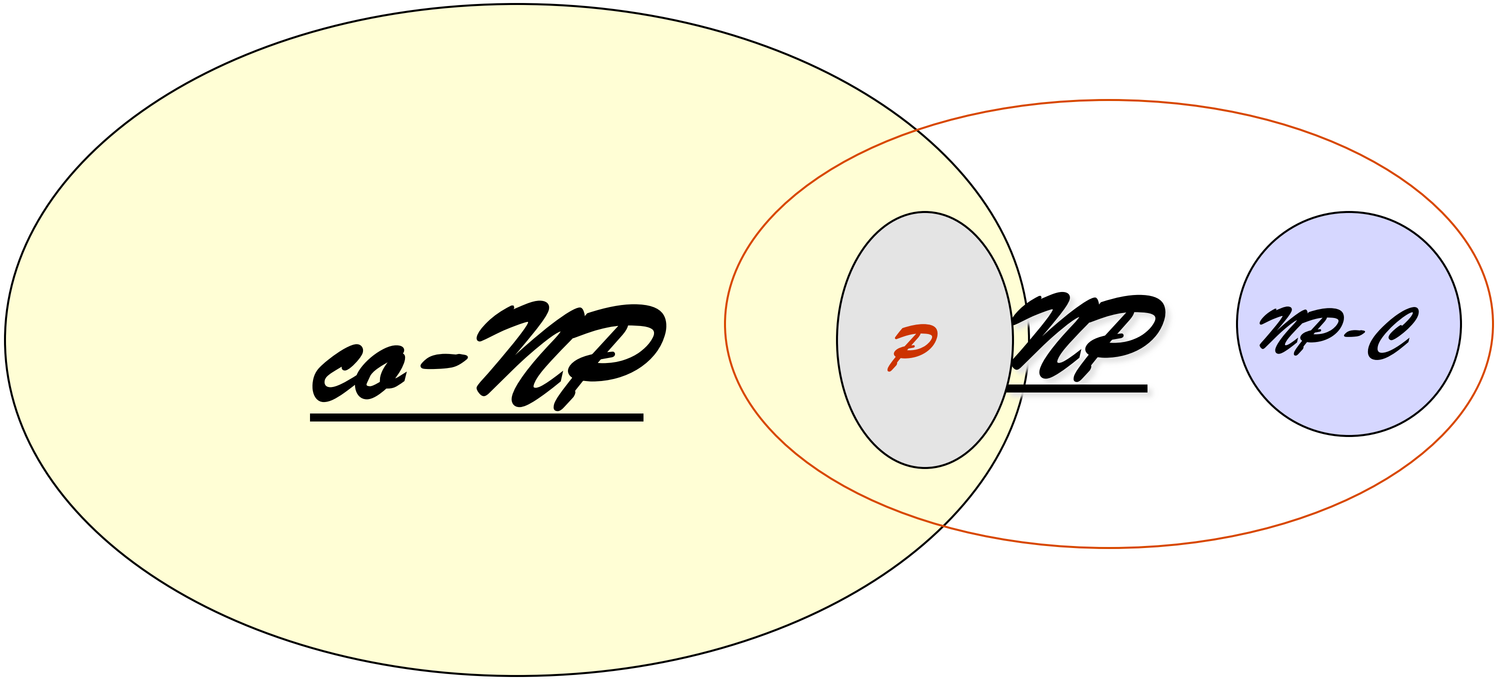
How to prove problem p is *NP-Complete*?

- Show a polynomial-time reduction from **<u>every</u>** problem in *NP* to problem p;
- OR, Show a polynomial-time reduction from **<u>any NP-complete</u>** problem to problem p;

What is a reduction?

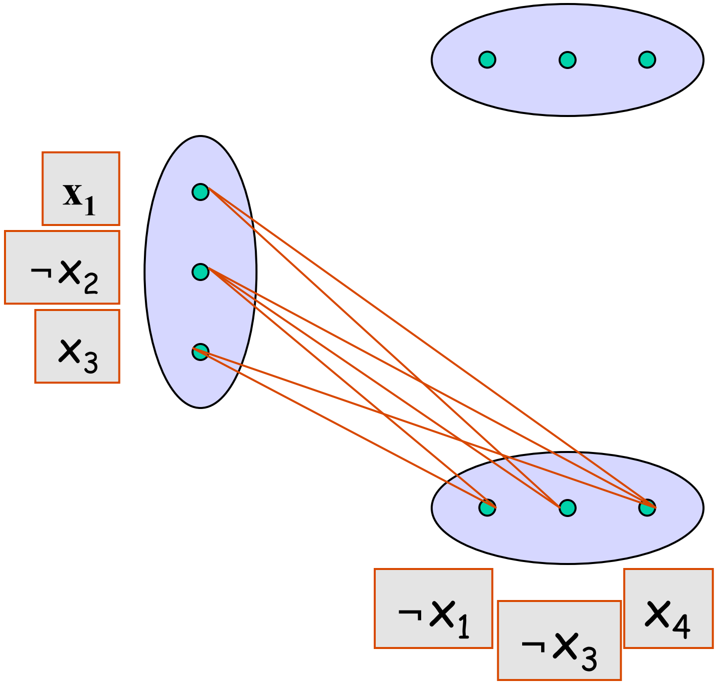
- A reduction from problem q to problem p is an algorithm A such that
  - Algorithm A takes an instance of problem q (call it I_q) and outputs an instance of problem p (call it I_p), and
  - I_q is a YES-instance iff I_p is a YES-instance
- So what is a polynomial-time reduction?

The problem classes and their relationships



CLIQUE is *NP-Complete*

- CLIQUE is in *NP*.
- Reduce 3SAT to CLIQUE in polynomial time.
- $F = (x_1 \vee \neg x_2 \vee x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(x_2 \vee x_3 \vee \neg x_4)(\neg x_1 \vee \neg x_2 \vee x_3)$

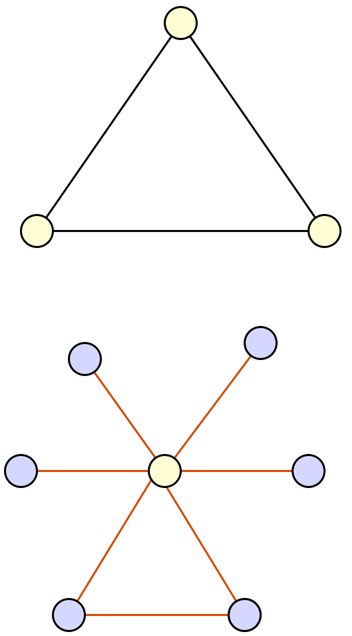


F is satisfiable if and only if G has a clique of size k where k is the number of clauses in F.

Vertex Cover

A **vertex cover** is a set of vertices that "covers" all the edges of the graph.

Examples



Hamiltonian Cycle Problem (HCP)

Input: Graph G

Question: Does G contain a hamiltonian cycle?

- HCP is in *NP*.
- There exists a polynomial-time reduction from 3SAT to HCP.
- Thus HCP is *NP-Complete.*

Shortest Path vs Longest Path

Input: Graph G with edge weights, vertices u and v, bound B

Question: Does G contain a path from u to v of length at most B? (SHORTEST PATH)

Question: Does G contain a path from u to v of length at least B? (LONGEST PATH)

Homework: Listen to Cool MP3:

http://www.cs.princeton.edu/~wayne/kleinberg-tardos/longest-path.mp3

Perfect (2-D) Matching vs 3-D Matching

1. Input: Bipartite graph, G(U,V,E)
   Question: Does G have a perfect matching?

2. Input: Sets U and V, and E = subset of U×V
   Question: Is there a subset of E of size |U| that covers U and V? [Related to 1.]

3. Input: Sets U,V,W, & E = subset of U×V×W
   Question: Is there a subset of E of size |U| that covers U, V and W?

Coping with NP-Completeness

- Approximation: Search for an "almost" optimal solution with provable quality.
- Randomization: Design algorithms that find "provably" good solutions with high prob and/or run fast on the average.
- Restrict the inputs (e.g., planar graphs), or fix some input parameters.
- Heuristics: Design algorithms that work "reasonably well".

Optimization Problems

- Problem:
  - A problem is a function (relation) from a set I of instances of the problem to a set S of solutions.
    - $p: I \rightarrow S$
- Decision Problem:
  - Problem with S = {TRUE, FALSE}
- Optimization Problem:
  - Problem with a mapping from set S of solutions to a positive rational number called the solution value
    - $p: I \rightarrow S \rightarrow m(I,S)$

Optimization Versions of NP-Complete Problems

- TSP
- CLIQUE
- Vertex Cover & Set Cover
- Hamiltonian Cycle
- Hamiltonian Path
- SAT & 3SAT
- 3-D matching

Optimization Versions of NP-Complete Problems

- Computing a minimum TSP tour is NP-hard (every problem in NP can be reduced to it in polynomial time)
- BUT, it is not known to be in NP
- If a problem P is NP-Complete, then its optimization version is NP-hard (i.e., it is at least as hard as any problem in NP, but may not be in NP)
  - Proof by contradiction!

Performance Ratio

- Approximation Algorithm A
  - $A(I)$
- Optimal Solution
  - $OPT(I)$
- Performance Ratio on input I for minimization problems
  - $R_A(I) = \max \{A(I)/OPT(I), OPT(I)/A(I)\}$
- Performance Ratio of approximation algorithm A
  - $R_A = \inf \{r \geq 1 \mid R_A(I) \leq r, \text{ for all instances}\}$

Metric Space

- It generalizes concept of Euclidean space
- Set with a distance function (metric) defined on its elements
  - D: M X M → R (assigns a real number to distance between every pair of elements from the metric space M)
    - $D(x,y) = 0$ iff $x = y$
    - $D(x,y) \geq 0$
    - $D(x,y) = D(y,x)$
    - $D(x,y) + D(y,z) \geq D(x,z)$

Examples of metric spaces

- Euclidean distance
- L_p metrics
- Graph distances
  - Distance between elements is the length of the shortest path in the graph

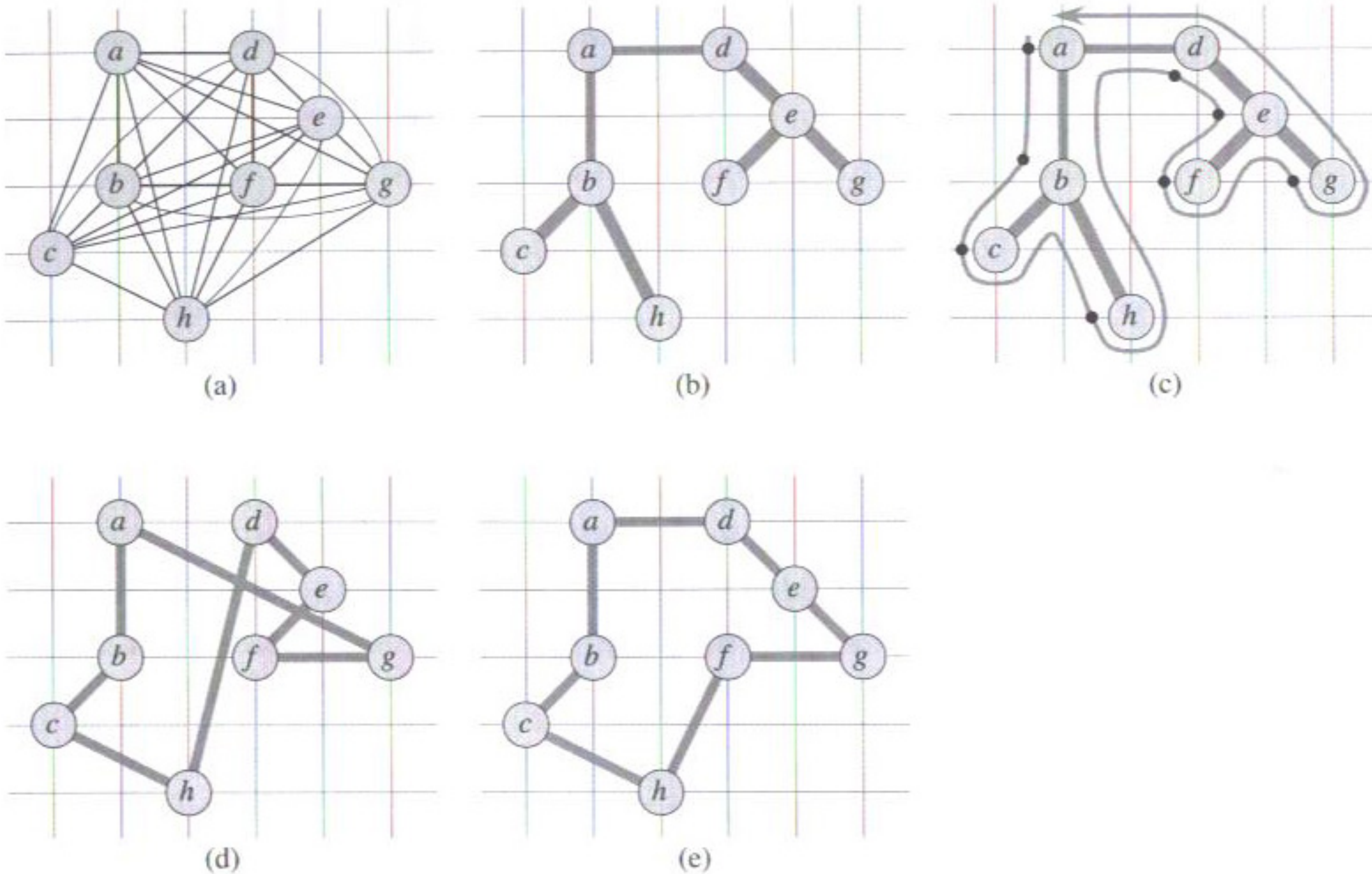
TSP

- TSP in general graphs cannot be approximated to within a constant (Why?)
  - What is the approach?
    - Prove that it is hard to approximate!
- TSP in general metric spaces holds promise!
  - NN heuristic [Rosenkrantz, et al. 77]
    - $NN(I) \le \frac{1}{2}(\text{ceil}(\log_2 n) + 1)\ OPT(I)$
  - 2-OPT, 3-OPT, k-OPT, Lin-Kernighan Heuristic
- Can TSP in general metric spaces be approximated to within a constant?

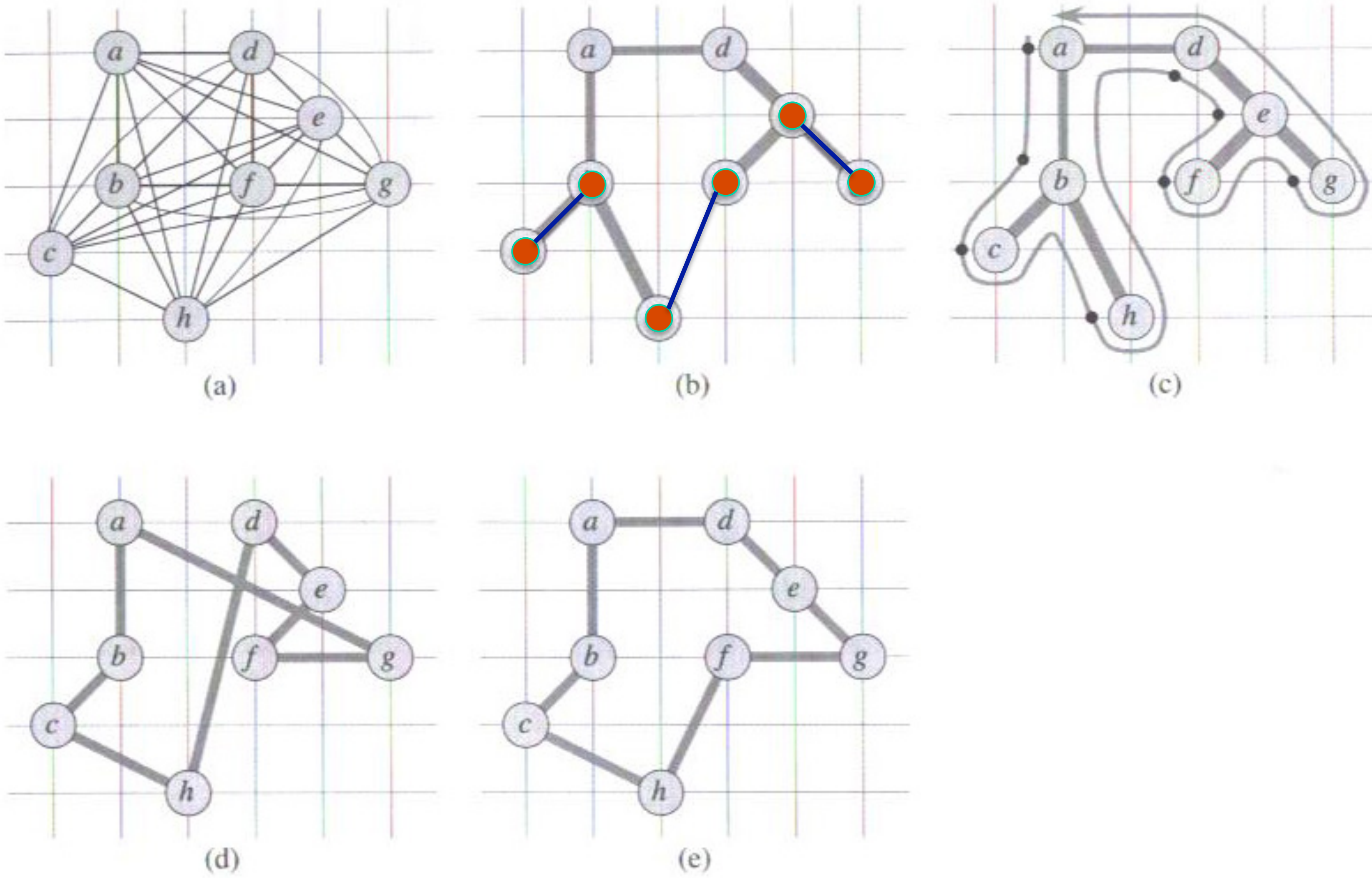
TSP in Euclidean Space

- TSP in Euclidean space can be approximated.
  - MST Doubling (DMST) Algorithm
    - Compute a MST, M
    - Double the MST to create a tour, T_1
    - Modify the tour to get a TSP tour, T
  - Theorem: DMST is a 2-approximation algorithm for Euclidean metrics, i.e., DMST(I) < 2 OPT(I)
  - Analysis:
    - $L(T) \leq L(T_1) = 2L(M) \leq 2L(T_{OPT})$
  - Is the analysis tight?

Example of MST Doubling Algorithm



Example of Christofides Algorithm



TSP in Euclidean Metric

- Improved algorithms
  - $MM(I) < 3/2\ OPT(I)$ [Christofides]
    - Christofides observed that DMST has 4 stages:
      - Find MST
      - Double all edges
      - Find Eulerian tour of resulting graph
      - Convert Eulerian tour into TSP tour
    - He modified step 2 to the following
      - Add a matching of odd degree vertices
  - $PTAS(I) < (1+\varepsilon)\ OPT(I)$ [Arora]

TSP Approximation Algorithm

Theorem: The <u>MST doubling algorithm</u> is a 2-approximation algorithm for inputs from any metric space.

Greedy Vertex Cover

- Algorithm
  - While graph G has at least one edge
    - Pick vertex v of highest degree in G and add to VC
    - Remove all edges incident on v in G
- Analysis
  - $|VC| \leq \log n \, |VC_{OPT}|$ [Is this tight?]

Greedy Vertex Cover: Analysis

- Pay $1 for each vertex picked
- If vertex v was chosen in an iteration, then each edge e deleted in that iteration was covered with cost(e) = $ 1/deg(v)
- Thus, in each iteration, picking vertex with max degree is same as picking vertex with least average cost per incident edge
- Size of VC picked = sum of edge costs
- Goal is to bound sum of edge costs

Greedy Vertex Cover: Analysis

- Let by C be an optimal vertex cover of size K
- Label edges in deletion order $e_1, e_2, \ldots, e_m$
- Let e_j be edge deleted in iteration i
- At least $m-j+1$ edges remain at start of iteration i which can be covered by C with average cost $K/(m-j+1)$
- Total cost of all edges $\leq$ $\Sigma_j K/(m-j+1)$
- $\leq K \log m$

Greedy Vertex Cover: Analysis

- Performance ratio ≤ log n
- Is the analysis tight?
  - Goal is to find graph such that after K rounds, we are left with half the edges uncovered
  - Make the graph recursive so that we need log n such rounds before all edges are covered.
- Challenge!
- Another challenge: try to generalize to weighted vertex cover problem

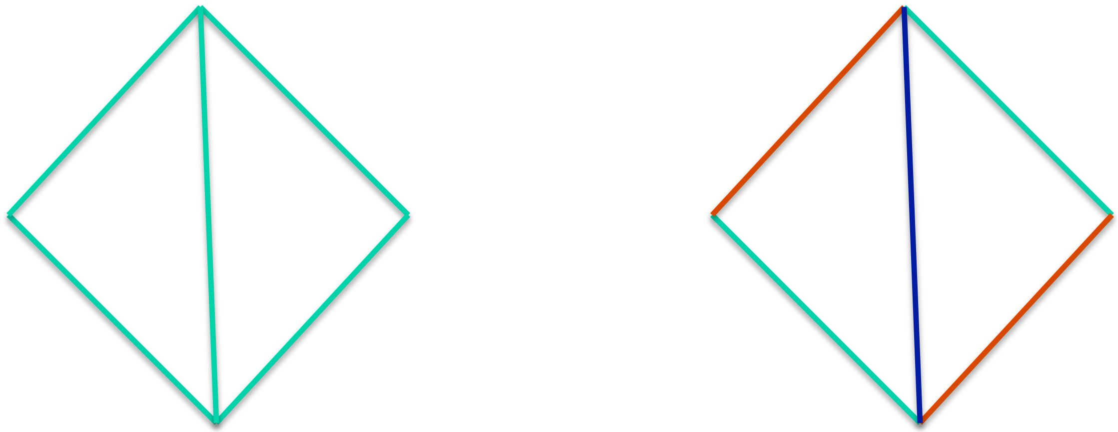
Vertex Cover

- Find the smallest set of vertices that are adjacent to all edges in the graph.
- Approximation Algorithm:
  - Initialize vertex cover C = empty set
  - while (an edge remains in the graph)
    - Choose arbitrary edge e = (u,v)
    - Add u and v to vertex cover C
    - Remove all edges incident on u or v
  - Output set C
- Analysis: $|C| \leq 2|C_{OPT}|$ [Is this tight?]

Complements and Approx Algorithms

- Complement of a clique subgraph is an independent set (i.e., a subgraph with no edges connecting any of the vertices)
- If a vertex cover is removed (including all incident edges), what remains?
  - ??
- If the minimum vertex cover problem can be 2-approximated, what about the maximum clique or maximum independent set?
  - ??

Edge Colorings Example



Edge Colorings

- Theorem: Every graph can be edge colored with at most Δ+1 colors, where Δ is the maximum degree of the graph.
- Theorem: No graph can be edge colored with less than Δ colors.
- Theorem: It is NP-complete to decide whether a graph can be edge colored with Δ colors [Holyer, 1981]
  - Thus it can be approximated to within an additive constant. Can't do better than that!

Some NP-Complete Number Problems

- Input: set S of n integers
- Question 1: Is there a subset of S that adds up to 0?
  - Example: { –7, –3, –2, 5, 8}

SUBSET-SUM

- Input: set S of n integers, and integer B
- Question 2: Is there a subset of S that adds up to B (part of input)?
  - Example

    S = {267,493,869,961,1000,1153,1246,1598, 1766,1922} and B = 5842

SUBSET-SUM

More NP-Complete Number Problems

- Input: set S of n integers
- Question 3: Is there a partition of S into two subsets each with the same sum?
  - Example: { -7, -3, -2, 1, 5, 8}

PARTITION

- Input: set S of 3n integers
- Question 4: Is there a partition of S into |S|/3 subsets each of size 3 and each of which adds up to the same value?
  - Strongly NP-Complete!

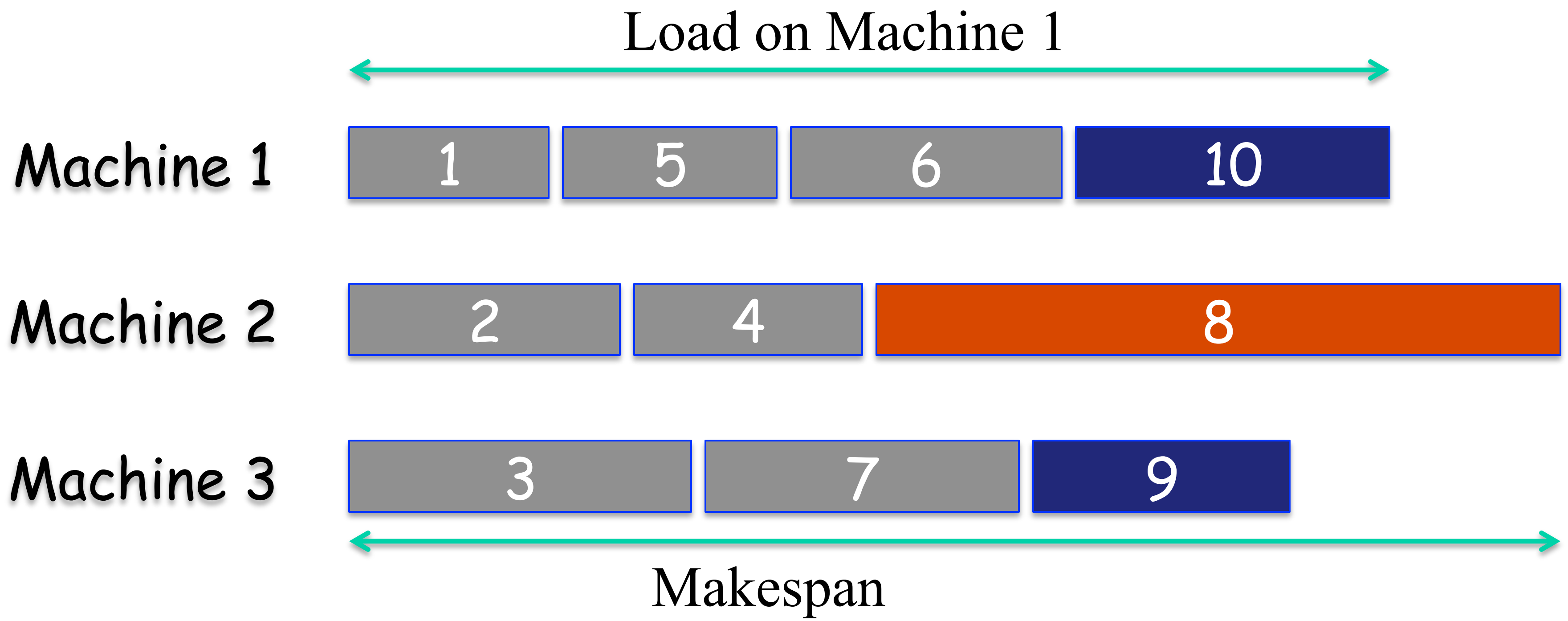
3-PARTITION

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: The load of machine i is L_i = sum of processing times of assigned jobs.
- 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. NP-Complete problem

Example from Kleinberg & Tardos; Slides inspired by Kevin Wayne

Example

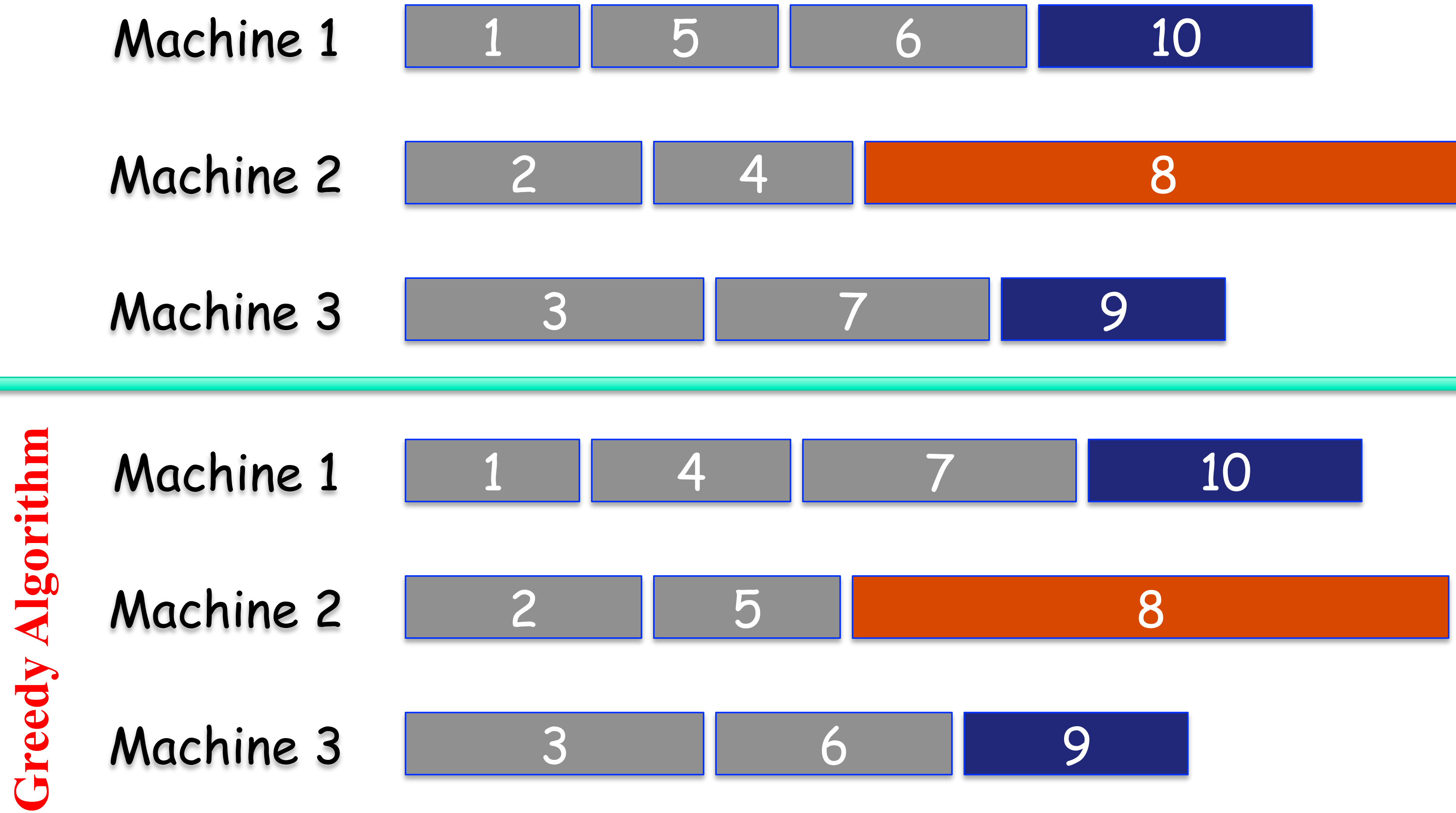


Load on Machine 1
Machine 1
1
5
6
10
Machine 2
2
4
8
Machine 3
3
7
9
Makespan

Greedy Algorithm

- Algorithm:
  - for jobs 1 to n (in any order)
    - Assign job j to machine with least load
- Observations:
  1. $L_{OPT} \geq \max\{t_1, \ldots, t_n\}$
  2. $L_{OPT} \geq \Sigma_i\, t_i/m$ (average load on a machine)
  3. If $n > m$, then $L_{OPT} \geq 2t_{small}$

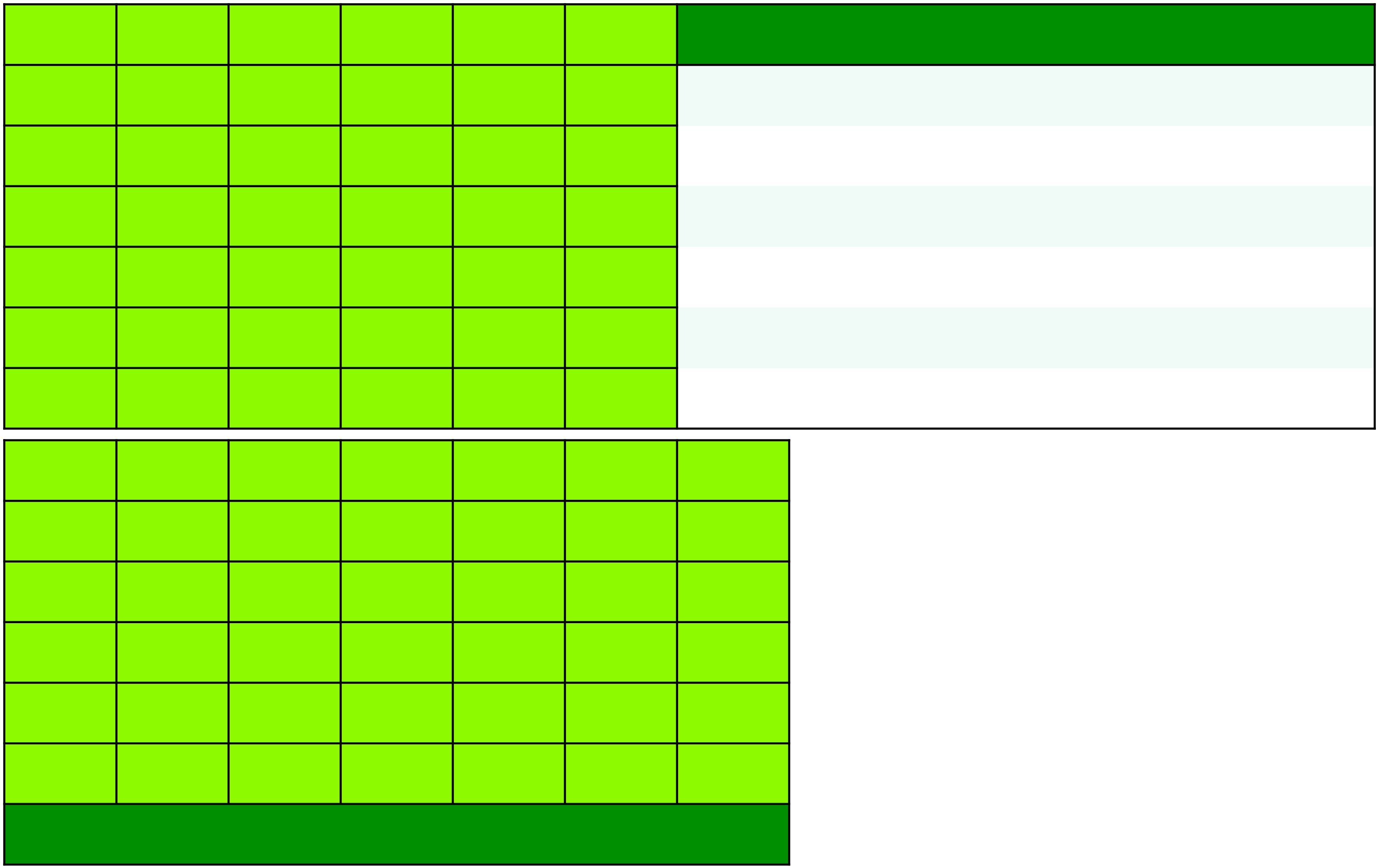
Example


Machine 1
1
5
6
10
Machine 2
2
4
8
Machine 3
3
7
9
Greedy Algorithm
Machine 1
1
4
7
10
Machine 2
2
5
8
Machine 3
3
6
9

Analysis

- Theorem: Greedy Algorithm is 2-approximate
- Proof:
  - Let i be machine with maximum load L_i. Let j be last job scheduled on it.
  - Before j was assigned, machine i had least load.
  - Thus $L_i - t_j \leq$ average load $\leq L_{OPT}$
  - $t_j \leq L_{OPT}$
  - $L_i \leq 2L_{OPT}$
- Is the analysis tight?

Analysis is tight!



Longest Processing Time (LPT) Algorithm

- Algorithm:
  - for jobs 1 to n (in decreasing order of time)
    - Assign job j to machine with least load
- Proof:
  - Let i be machine with maximum load L_i. Let j be last job scheduled on it.
  - The last job is the shortest and is at most $L_{OPT}/2$
  - Thus L_i is at most $(3/2)L_{OPT}$ [if $n > m$]
- Is the analysis tight?
  - No! (4/3)-approximation exists [Graham, 1969]

Fractional Knapsack Problem

- Burglar's choices:

  n bags of valuables: $x_1, x_2, \ldots, x_n$

  Unit Value: $v_1, v_2, \ldots, v_n$

  Max number of units in bag: $q_1, q_2, \ldots, q_n$

  Weight per unit: $w_1, w_2, \ldots, w_n$

  Getaway Truck has a weight limit of B.

  Burglar can take "fractional" amount of any item.

  How can burglar maximize value of the loot?
- Greedy Algorithm works!

  Pick maximum quantity of highest value per weight item. Continue until weight limit B is reached.

0-1 Knapsack Problem

- Burglar's choices:
  Items: $x_1, x_2, \ldots, x_n$
  Value: $v_1, v_2, \ldots, v_n$
  Weight: $w_1, w_2, \ldots, w_n$
  Getaway Truck has a weight limit of B.
  "Fractional" amount of items NOT allowed
  How can burglar maximize value of the loot?
- Greedy Algorithm does not work! Why?
- Need dynamic programming!

0-1 Knapsack Problem: Example

B = 12

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

0-1 Knapsack Problem

- Subproblems?
  - V[j, L] = Optimal solution for knapsack problem assuming truck weight limit L & choice of items from set {1,2,…, j}.
  - V[n, B] = Optimal solution for original problem
  - V[1, L] = easy to compute for all values of L.
- Recurrence Relation? [Either x_j included or not]
  - $V[j, L] = \max\{ V[j-1, L] \;,\; v_j + V[j-1, L-w_j] \}$
- Table of solutions?
  - V[1..n, 1..B]
- Ordering of subproblems?
  - Row-wise

Another NP-Complete Number Problem

- Input: set S of n items each with values $\{v_1, \ldots, v_n\}$ and weights $\{w_1, \ldots, w_n\}$; Knapsack with weight limit B and value V
- Question: Is there a choice of items from S whose weights add up to at most B and whose value adds up to at least V?

KNAPSACK

Knapsack Problem

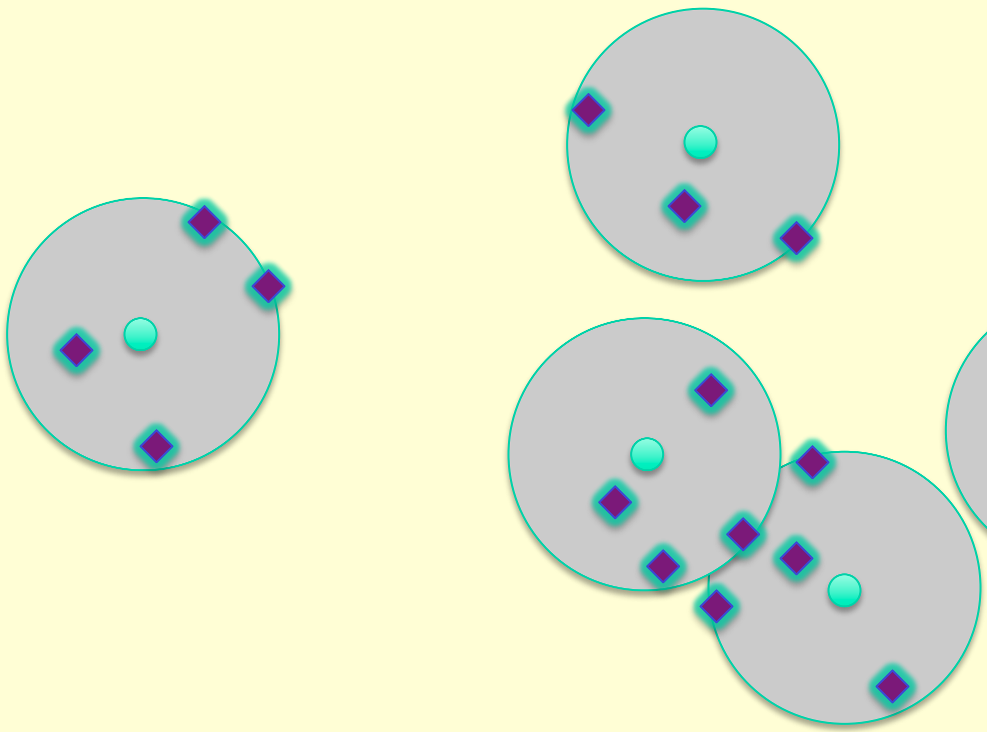
- The 0-1 Knapsack problem is NP-Complete.
- The 0-1 Knapsack problem can be solved exactly in O(nB) time.
- Does this mean $\mathcal{P}$ = $\mathcal{NP}$? What is going on here?
- What we have here is a pseudo-polynomial time algorithm. Why?

Knapsack: Approximations

- Greedy Algorithm is 2-approximate
  - Sort items by value/weight
  - Greedily add items to knapsack if it does not exceed the weight limit
- Improved algorithm is (1 + 1/k)-approximate [Sahni, 1975]
  - Time complexity is polynomial in n, logV, and logB
  - Time complexity is exponential in k
  - This is a "approximation scheme"
  - Implies cannot get to within an additive constant!

Clustering

- Set of points $\{p_1,\ldots,p_n\}$ in R^d
- Typical data mining problem is to find k clusters in this data

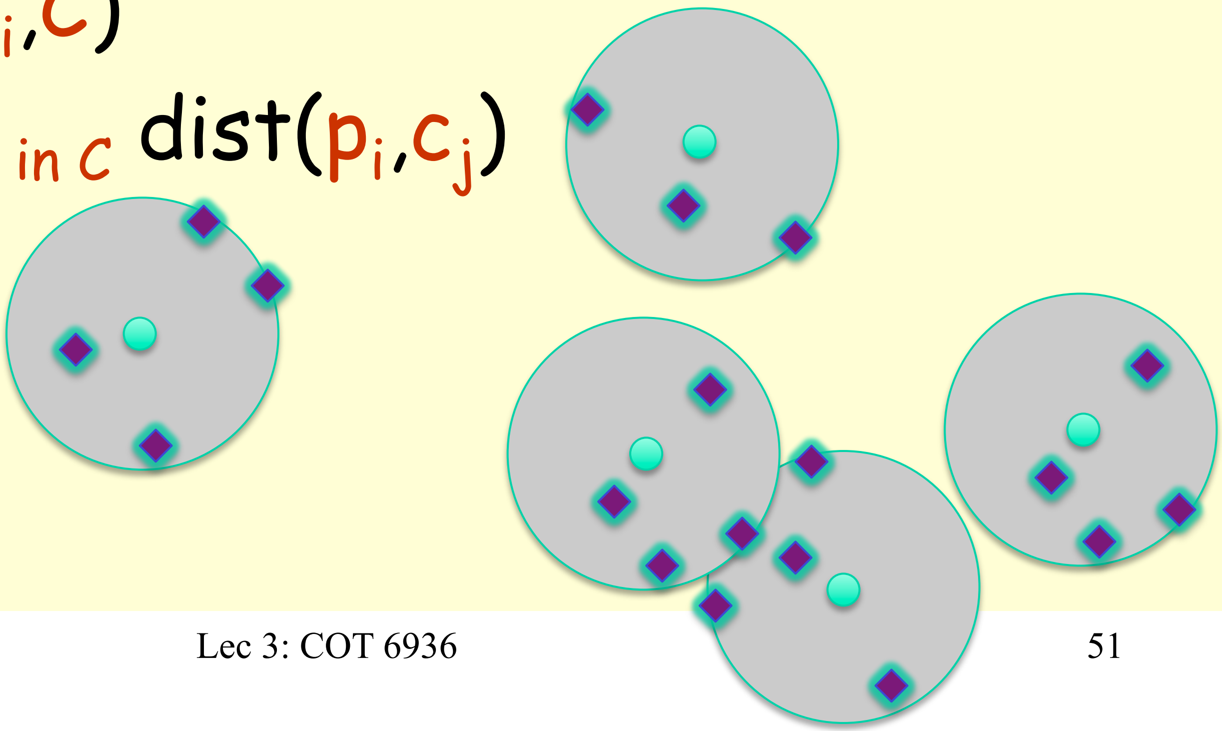


Clustering

- Requires a distance function
  - Euclidean distance (L_2 distance) and L_p metrics
  - Mahalanobis distance
  - Pearson Correlation Coefficient
  - General metric distance
- Requires an objective function to optimize
  - Maximum distance to a center
  - Sum of distances to a center
  - Median of distance to a center
- Can any point be center? (finite vs infinite)

Clustering

- Set of points $S = \{p_1,\ldots,p_n\}$ in R^d
- Find a set of k centers such that the maximum of the distance of a point to its closest center is minimized.
- $\text{Min}_C \text{ Max}_i \; d(p_i, C)$
- $d(p_i, C) = \text{Min}_{c_j \text{ in } C} \; \text{dist}(p_i, c_j)$

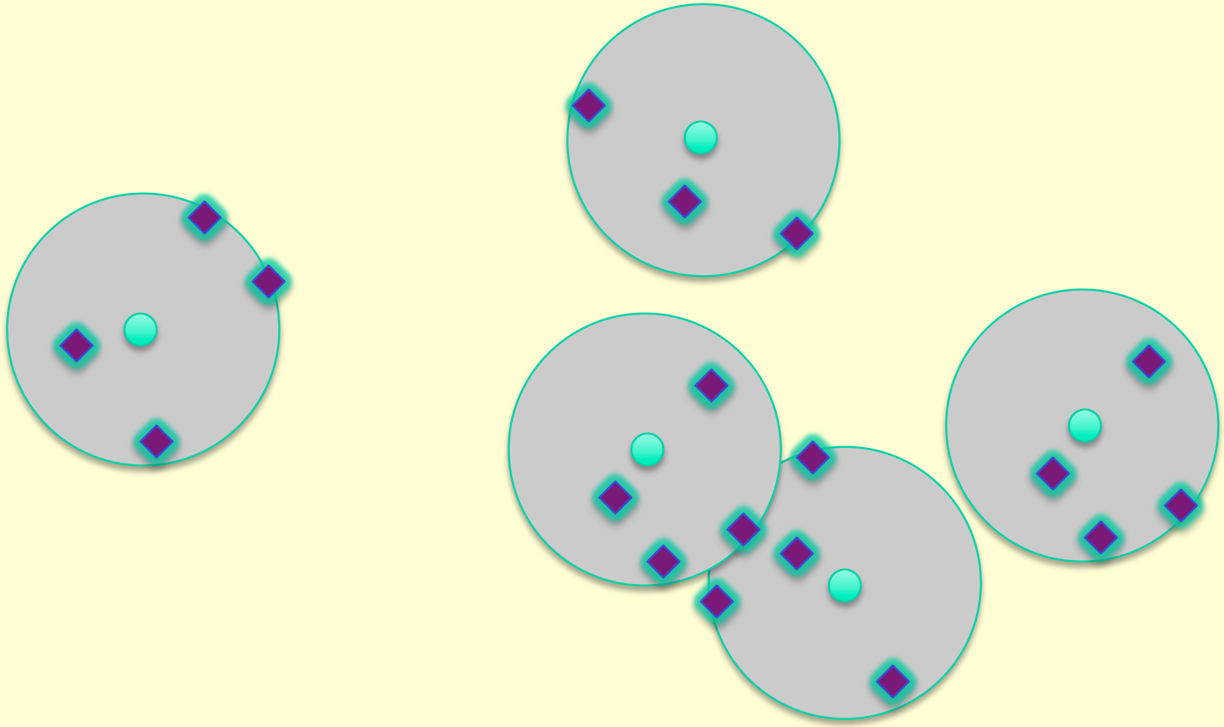


Well-known clustering techniques

- Algorithms
  - K-Means
  - Hierarchical clustering
  - Clustering using MSTs
  - Greedy algorithm
    - Put first center at best possible location for single center; then keep adding centers to reduce covering radius each time by as much as possible.
- Disadvantages
  - All three are heuristic algorithms (solutions not optimal, no provable approximation factor)

Clustering: Approximation Algorithm

- Improved Greedy algorithm:
  - Repeatedly choose (k vertices selected) next center to be site farthest from any existing center. Choose first center arbitrarily.



Clustering: Approximation Analysis

- Analysis:
  - Let r = radius of largest greedy cluster
  - Let r_{OPT} = radius of largest optimal cluster
  - If distance from optimal center to every site is ≤ r_{OPT}, then distance from any site to some optimal center is ≤ r_{OPT}. Take ball of radius r_{OPT} around every greedy center. All optimal centers are covered;
  - Ball of radius $2r_{OPT}$ around each greedy center will cover every site.
  - Thus $r \leq 2\, r_{OPT}$.

Alternative (Corrected) Proof

- Improved Greedy algorithm:
  - Repeatedly choose (k vertices selected) next center to be site farthest from any existing center
- Analysis:
  - Let r = min distance between 2 greedy centers & r_{OPT} = radius of largest cluster in optimal clustering
  - Let $r > 2r_{OPT}$. Take ball of radius $\frac{1}{2}r$ around every greedy center. Exactly one optimal center in each ball (?);
  - Pair optimal and greedy centers (c_i, c_i^*).
  - Let s be any site and c_i^* be its nearest optimal center
  - $d(s, C) \leq d(s, c_i) \leq d(s, c_i^*) + d(c_i^*, c_i) \leq 2r(C^*)$.
  - Thus $r(C) \leq 2r(C^*)$, i.e., $r < 2r_{OPT}$

Observation

- Analysis compared r with r_{OPT} without knowing what the optimal clustering looked like!

Yet Another Proof!

- Improved Greedy algorithm:
  - Repeatedly choose (k vertices selected) next center to be site farthest from any existing center
- Analysis:
  - Let r = min distance between 2 greedy centers & r_{OPT} = radius of largest cluster in optimal clustering
  - Let $r > 2r_{OPT}$. Take ball of radius $\frac{1}{2}r$ around every greedy center. Exactly one optimal center in each ball (?);
  - Ball of radius r_{OPT} around each greedy center will cover every optimal center. Ball of radius $2r_{OPT}$ around each greedy center will cover every site.
  - Thus $r \leq 2\, r_{OPT}$. CONTRADICTION!

Bin Packing

- Given an infinite number of unit capacity bins
- Given finite set of items with rational sizes
- Place items into minimum number of bins such that each bin is never filled beyond capacity
- BIN-PACKING is NP-Complete
  - Reduction from 3-PARTITION

Bin Packing: Approx Algorithm

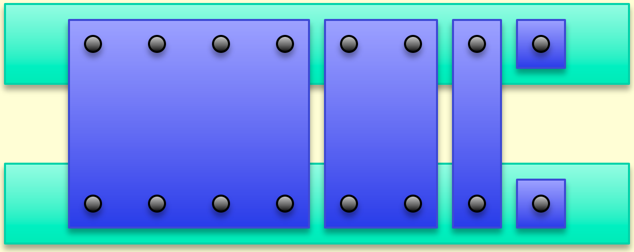
- First-Fit:
  - place item in lowest numbered bin that can accommodate item
    - $FF(I) < 2\ OPT(I)$
    - $FF(I) \leq 17/10\ OPT(I) + 2$
- First-Fit Decreasing:
  - Sort items in decreasing size and then do first-fit placement
    - $FFD(I) = 11/9\ OPT(I) + 4$

Bin Packing: Approx Algorithm

- Connection to Partition
  - Hard even when you have only 2 bins
  - Cannot approximate to within $(3/2)-\varepsilon$ unless P = NP
  - Can get $(1+\varepsilon)$approximation if OPT > $2/\varepsilon$

Set Cover

- Greedy Algorithm
  - While there are uncovered items
    - Find set with most uncovered items and add to cover
- Analysis
  - Approximation Ratio = log n
  - It is tight. In example below, it will pick 5 sets instead of 2.



Approximability of NP-Hard Problems

Approximation Factor	Problem/Algorithm
$1+\varepsilon$	Euclidean TSP (Arora)
1.5	Euclidean TSP (Christofides)
2	Vertex Cover
c	Coloring
log n	Set Cover
$\log^2 n$	
$\sqrt{n}$	
n^{ε}	Independent Set, Clique
n	General TSP

Reading Assignment

Required Reading for Feb 6

- Network Flow
  - Ford Fulkerson Algorithm
- Linear Programming
  - Standard LP
  - Dual LP
  - Feasibility and feasible region