## COT 6936: Topics in Algorithms

Giri Narasimhan
ECS 254A / EC 2443; Phone: x3748
giri@cs.fiu.edu

http://www.cs.fiu.edu/~giri/teach/COT6936_S10.html
https://online.cis.fiu.edu/portal/course/view.php?id=427



## 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 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
  - $R_A = \inf \{r \geq 1 \mid R_A(I) \leq r$, 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

1/12/10 COT 6936 7

## 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?

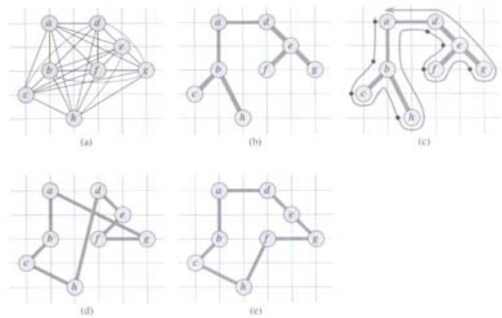
1/12/10 COT 6936 8

## 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:
    - $L(T) \le L(T_1) = 2L(M) \le 2L(T_{OPT})$
  - Is the analysis tight?

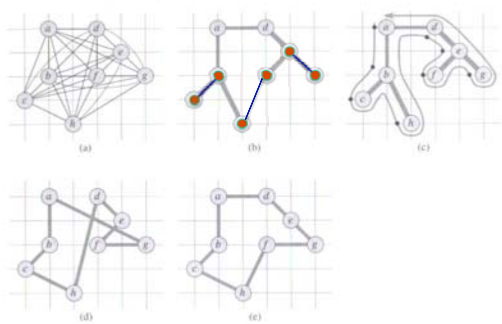
1/12/10 COT 6936 9

## 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 MST doubling algorithm is a 2-approximation algorithm for inputs from any metric space.



## 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?]



## Greedy Vertex Cover

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

## Greedy Vertex Cover: Analysis

- Let $C$ be optimal vertex cover and $K = |C|$
- Iteration $i$: vertex of maximum degree $d_i$ is processed resulting in graph $G_i$
- Let $e(G)$ = # edges in $G$. So $e(G_i) = e(G_{i-1}) - d_i$
- Observation: Sum of degrees of vertices in any cover is $\geq e(G)$. Thus their average degree is $\geq e(G_{i-1})/K$. And, $\boxed{d_i \geq e(G_{i-1})/K}$.
- $\Sigma_K d_i \geq \Sigma_K e(G_{i-1})/K \geq e(G) - \Sigma_K d_i$
- Thus $\boxed{\Sigma_K d_i \geq e(G)/2}$



## Greedy Vertex Cover: Analysis

- After K vertices are removed, half the edges of G are covered. After K logn vertices are removed, all edges of G will be covered.
- 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.



## 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 $\Delta+1$ colors, where $\Delta$ is the maximum degree of the graph.
- Theorem: No graph can be edge colored with less than $\Delta$ colors.
- Theorem: It is NP-complete to decide whether a graph can be edge colored with $\Delta$ 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? SUBSET-SUM
  - Example: { −7, −3, −2, 5, 8}
- 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)? SUBSET-SUM
  - Example

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

## 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



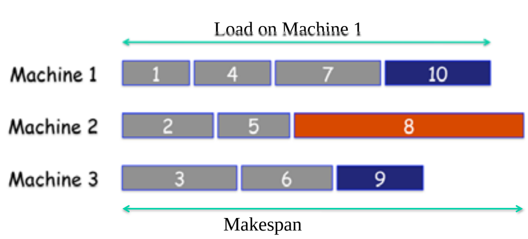
## Load Balancing

- 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



## 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. $L_{OPT} \geq AVG(t)$
  3. If $n > m$, then $L_{OPT} \geq 2t_{small}$

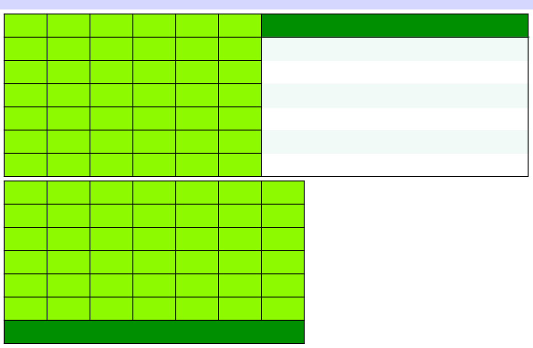


## Analysis

- 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 $L_i - t_j \leq L_k$, for all k in [1..m]
  - $t_j \leq L_{OPT}$
  - $L_i \leq 2L_{OPT}$
- 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 |

1/12/10 COT 6936 31

## 0-1 Knapsack Problem

- Subproblems?
  - $V[j, L]$ = Optimal solution for knapsack problem assuming truck weight limit $L$ & choice of items from set $\{1,2,\ldots, j\}$.
  - $V[n, B]$ = 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

10/30/08 COT 5407 32

## 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

1/12/10 COT 6936 33

## 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?



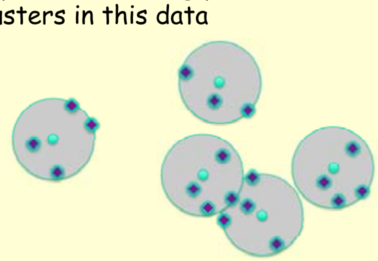
## Knapsack: Approximations

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



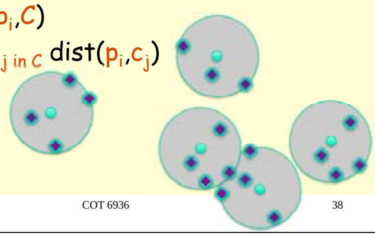
## Clustering

- 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.
- $\mathrm{Min}_C \, \mathrm{Max}_i \, d(p_i, C)$
- $d(p_i, C) = \mathrm{Min}_{c_j \text{ in } C} \, \mathrm{dist}(p_i, c_j)$



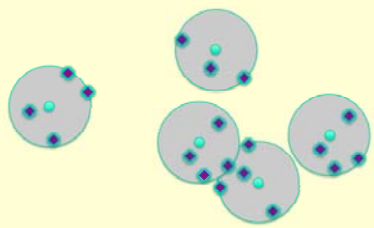


## Well-known clustering techniques

- 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 next center to be site farthest from any existing center. Choose first center is 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 $\leq r_{OPT}$, then distance from any site to some optimal center is $\leq r_{OPT}$. 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 next center to be site farthest from any existing center
- Analysis:
  - Let $r$ = distance between last 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!



## 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
    - $FFD(I) = 11/9\ OPT(I) + 4$

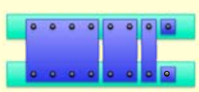
## Bin Packing: Approx Algorithm

- 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 |
| $\log^2 n$ | |
| $\sqrt{n}$ | |
| $n^{\varepsilon}$ | Independent Set, Clique |
| n | General TSP |