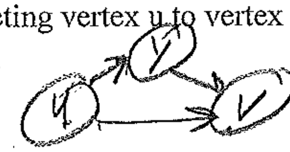
Data Structures (CS 1520) Lecture 29 Name:____________

Approximation Algorithm for TSP with Triangular Inequality

Restrictions on the weighted, undirected graph G=(V, E):

1. There is an edge connecting every two distinct vertices.
2. Triangular Inequality: If W(u, v) denotes the weight on the edge connecting vertex u to vertex v, then for every other vertex y,

$$W(u, v) \leq W(u, y) + W(y, v).$$



NOTES:

- These conditions satisfy automatically by a lot of natural graph problems, e.g., cities on a planar map with weights being as-the-crow-flys (Euclidean distances).
- Even with these restrictions, the problem is still NP-hard.

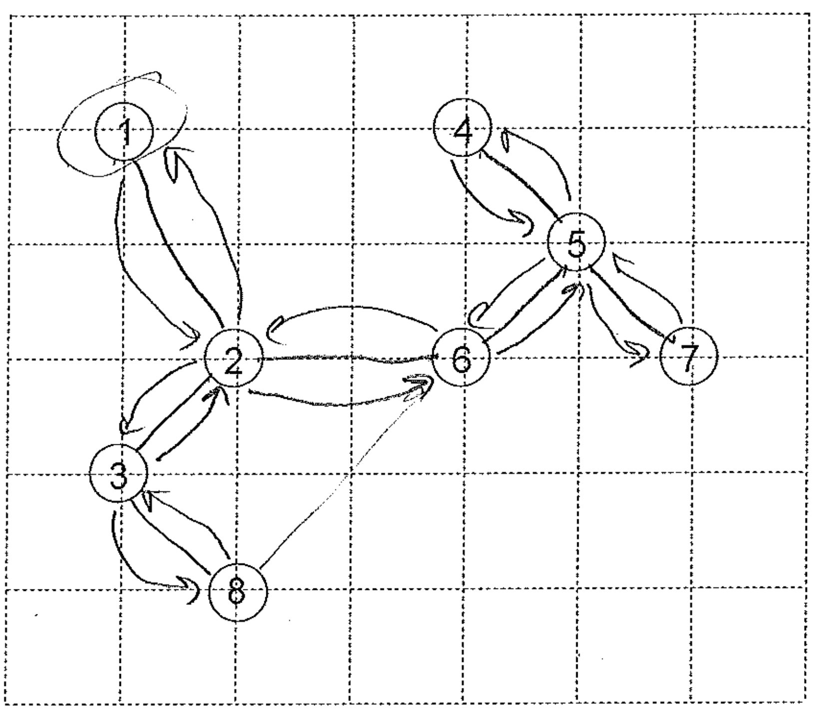
A simple TSP approximation algorithm:

Step 1. Determine a Minimum Spanning Tree (MST) for G (e.g., Prim's Algorithm section 4.1)

Step 2. Construct a path that visits every node by performing a preorder walk of the MST. (A *preorder walk* lists a tree node every time the node is encounter including when it is first visited and "backtracked" through.)

Step 3. Create a tour by removing vertices from the path in step 2 by taking shortcuts.

1) (Step 1) Determine a Minimum Spanning Tree (MST) for G (e.g., Prim's Algorithm) if we start with vertex 1 in the MST. (Assume edges connecting all vertices with their Euclidean distances)



Prim's algorithm is a greedy algorithm that performs the following:

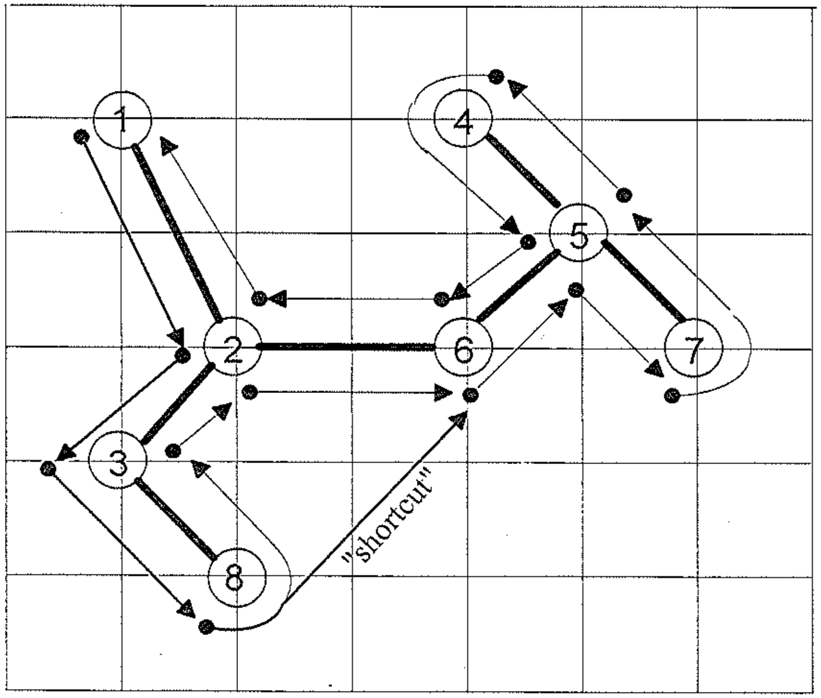
a) Select a vertex at random to be in the MST.
b) Until all the vertices are in the MST:
   - Find the closest vertex not in the MST, i.e., vertex closest to any vertex in the MST

Add this vertex using this edge to the MST

2) (Step 2) Determine the preorder walk of the MST.

1, 2, 3, 8, 3, 2, 6, 5, 7, 5, 4, 5, 6, 2, 1

1. Complete a tour by removing vertices from the path in step 2 by taking shortcuts.



a) Finish removing vertices from the preorder-walk path to create a tour by taking shortcuts:
[1 2 3 8 ~~3~~ ~~2~~ 6 5 7 ~~5~~ 4 ~~5~~ ~~6~~ ~~2~~ 1]

b) When scanning the above path, how did you know which vertices to eliminate to take a shortcut?

If a value has been seen before in the path, except for start vertex of 1.

c) If we take the optimal TSP tour and remove an edge, what do we have?

A spanning tree since it includes all the vertices and contains no cycles.

d) What is the relationship between the distance of the MST and the optimal TSP tour?

distance of the MST <= spanning tree by removing edge from optimal tour < the optimal TSP tour

distance of the MST < the optimal TSP tour

e) What is the relationship between the distance of the MST and the distance of the preorder-walk of the MST? Because every edge in the MST is traversed twice in the preorder-walk, then: distance of the MST * 2 = distance of the preorder-walk of the MST, or

0.5 * distance of the preorder-walk of the MST = distance of the MST

f) What is the relationship between the distance of the preorder-walk of the MST and the tour obtained from the preorder-walk of the MST?

tour obtained from the preorder-walk of the MST <= distance of the preorder-walk of the MST

0.5 * tour obtained from the preorder-walk of the MST <= 0.5 * distance of the preorder-walk of the MST

g) What is the relationship between the tour obtained from the preorder-walk of the MST and the optimal TSP tour?

0.5 * tour obtained from the preorder-walk of the MST <= 0.5 * distance of the preorder-walk of the MST = distance of the MST < the optimal TSP tour

0.5 * tour obtained from the preorder-walk of the MST < the optimal TSP tour

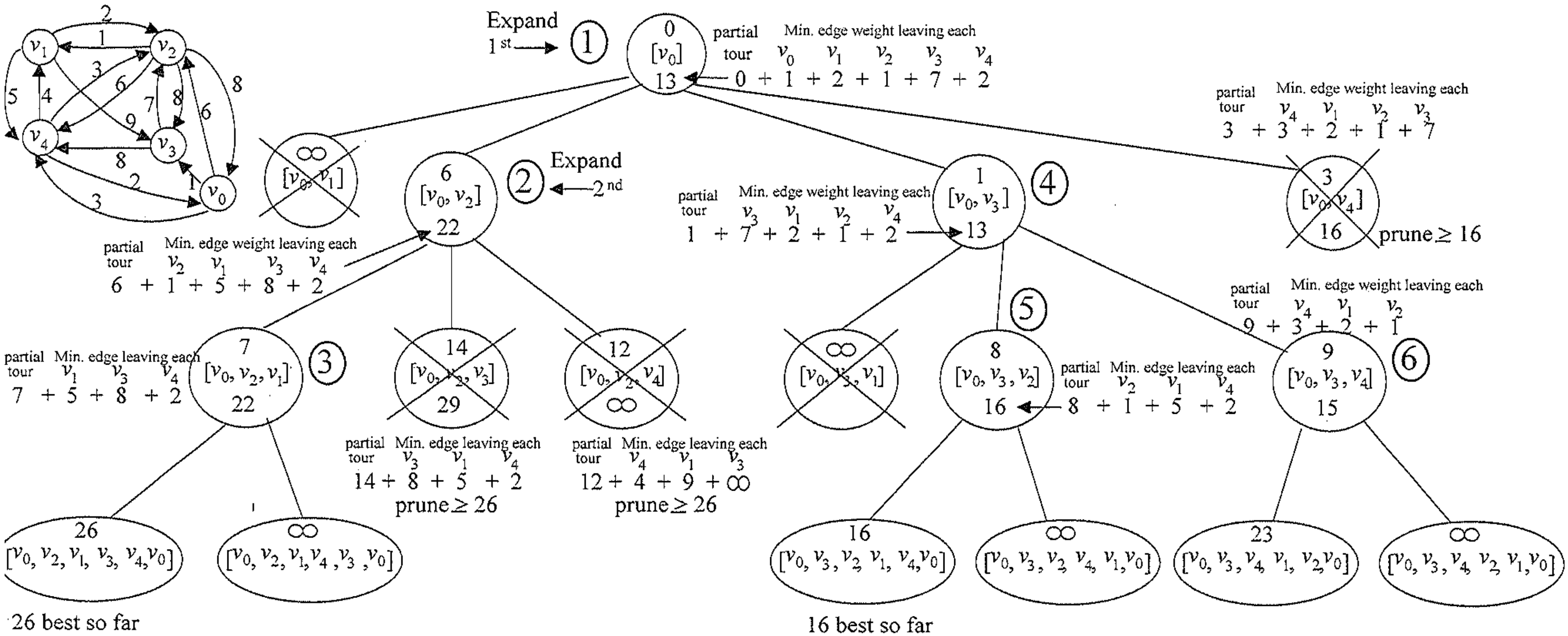
Thus, tour obtained from the preorder-walk of the MST < 2 * the optimal TSP tour

Handling "Hard" Problems: For many optimization problems (e.g., TSP, knapsack, job-scheduling), the best known algorithms have run-time's that grow exponentially ($O(2^n)$ or worse). Thus, you could wait centuries for the solution of all but the smallest problems!

Backtracking general idea: (Recall the coin-change problem from lectures 10 and 13)

- Search the "*state-space tree*" using depth-first search to find a suboptimal solution quickly
- Use the best solution found so far to prune partial solutions that are not "promising,", i.e., cannot lead to a better solution than one already found.

2. To prune a node in the search-tree, we need to be certain that it cannot lead to the best solution. We can calculate a "bound" on the best solution possible from a node (e.g., say node with partial tour: $[v_0, v_4, v_1]$) by summing the partial tour with the minimum edges leaving the remaining nodes. Complete the backtracking state-space tree with pruning.



3. In the *Best-First search with Branch-and-Bound* approach:

- does not limit us to any particular search pattern in the state-space tree
- calculates a "bound" estimate for each node that indicates the "best" possible solution that could be obtained from any node in the subtree rooted at that node, i.e., how "promising" following that node might be
- expands the most promising ("best") node first by visiting its children

a) What type of data structure would we use to find the most promising node to expand next? min. heap

b) Complete the best-first search with branch-and-bound state-space tree with pruning. Indicate the order of nodes expanded.