CS 466/666 **Spring 2013**

Approximate Solutions to NP-Hard Problems

What do you do if you don't see a reasonable solution to a problem?
Given a problem: Look for a solution by mapping to a known problem or type of problem.

Don't see a fast solution; perhaps it is NP hard.

Reduce NP hard problem to new one – but be we careful

In doing so, make sure your problem with all its special constraints is NP Hard….and not a potentially easier special case (e.g. 2-SAT).

Now you are stuck - try a "heuristic" of some sort, may get (optimal) solution or may not.

One notion is to get an approximation algorithm with a guaranteed approximation ratio.
$\rho(n) \geq \max\left(\frac{c}{c^*}, \frac{c}{c^*}\right)$ where c=our solution cost, c* = optimal (unknown)
The idea of the maximum is that the ratio is greater than 1 whether we have a maximization or minimization problem.

Ideal Situation:

Approximation scheme: taking into account ε, and getting a (1+ ε) ratio.

Polynomial time approximation scheme: a method that for any fixed ε >o, scheme runs in polynomial time (e.g. $n^{1/\varepsilon}$ is ok here)

But really we want
Fully polynomial time approximation scheme: runtime bounded by a polynomial in both $1/\varepsilon$ and n,
e.g. $O((1/\varepsilon)^2 n^3 \lg n)$.

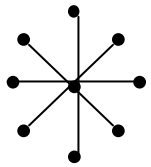
We will see a variety of problems/solutions.

Generally methods are simple. Bounds are pessimistic in terms of typical behaviour. Once we have an approximate solution, we may be able to find an even better solution and so a better guarantee for the particular case.

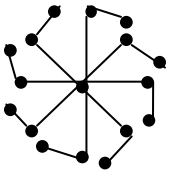
Vertex cover: Given undirected graph G=(V,E) and subset of vertices $V' \subseteq V$ such that $\forall$ edge (u,v) $u \varepsilon V'$ $v \varepsilon V'$ or both. Size of vertex cover (# nodes) is issue.
Note that minimum vertex cover is NP-Complete as we can reduce the clique problem to it.

It is natural to first try a greedy approach. In this case one might choose first the node of highest degree to be a member of the solution set. While this may often be a good idea, it is possible that a node of very high degree is not necessarily in the optimal solution.

Consider first a simple “star graph” with a centre and edges from the centre node to n-1 others, as below.



Choosing the centre node (which has the maximum degree) as a member of the vertex cover solves the problem as it covers all edges. However is we add an extra edge and node to each of the rays, we get a “windmill” in which the centre node is no longer in the optimal vertex cover.



Returning to our main interest of finding a fast technique guaranteed to give a solution within a “reasonable” factor of the cost of the optimal, consider the following method for finding a vertex cover:

```
Approx_VC(G)
V' ←ϕ
E'←E
While E'≠ ϕ
do      { choose arbitrary (u,v) in E'
        V'←V'  ∪ {u,v}   {i.e. take both!!!)
        Remove for E' every edge incident with u or v.
        }

Return V'
```

Ok, it’s a silly method and with care can be made linear, but ...

Theorem: Approx VC is a polynomial time 2-approximation algorithm.

Proof: Let A denote set of edges chosen
Every vertex cover, including any optimal one (C*), must contain at least one endpoint of each edge. In particular for any edge (u,v) as noted above, either u or v must be in the cover. So our V' has at most twice as many nodes as the optimal solution.

So we have a 2-approximation. Interesting idea, we don’t know C*, but can still argue about it. And note the method can be as bad as the bound states.
If you are trying to come up with “as good a solution as you can” to the vertex cover problem, it wold seem reasonable to take an extra scan over the nodes in V', and remove any node that (still) has all its edges covered by other nodes. One might even try this by trying to get rid of low degree noes first. There are no guarantees that this will help on any given instance, but it might, and it certainly won’t hurt. Actually there are no known polynomial time methods guaranteed to do significantly better than a 2-approximation.

Traveling Salesman Problem:

We now turn to the well know Traveling Salesman Problem: Given complete undirected graph G with nonnegative costs on edges, c(u,v). Find the Hamiltonian cycle of minimum length.
{c(A) will denote cost of our approximation A, c(A) = $\sum_{(u,v)\in A} c(u,v)$
The problems are both NP-hard: To prove Hamiltonian cycle can be shown to be NP-complete by a reduction from VC. This is an interesting proof using an innovative construction (see CLRS). Showing TSP is NP-hard is easy. Reduce Hamiltonian cycle to TSP {HC on unweighted graph → make complete graph by connecting edges weight 1 "nonedges" weight 2. The original graph has an HC iff the new weighted graph has a tour of length n}

Interesting observation: In reduction, give edges weight 0, non edges weight 1. Then HC iff tour of length 0. So there is no polynomial time approximations algorithms unless P=NP. If you feel weights of length 0 are a "cheat", give edges weight 1 and non edges weight 2^n. This makes the encoding still polynomial in the original size, indeed $O(n^3)$ bits where n is the number of nodes in the graph. (n bits for each of the non edges.) Now any deviation from a Hamiltonian circuit gives a path length exponential in n.

In many situations, however, we can add a constraint, the *triangle inequality*: c(u,w) ≤ c(u,v +c(v,w)

Fact: TSP NP-hard even with triangle inequality {as in original reduction above from HC}

Approx_TSP_2MST:
Find a minimum spanning tree of the graph. The basic tour is given by doing a depth first search order of the tree (and returning to the start node); this involves following each edge of the MST in each direction so the cost is twice the cost of the MST. This can be modified to a tour that inspects each node only once and may be shorter than basic form. Suppose we are at node u and the next two edges of the DFS are (u,v) and (v,w). If v has already been visited we simply replace these two edges with (u,w) which has cost at most c(u,v +c(v,w). (The edge (u,w) would be eliminated in the same way if we have already visited w)

Why is this a 2-Approximation?

1) Let C(T) denote sum of weights of tree edges, C(H*) the cost of the optimal TSP, and C(A) the cost of our approximation

Note H* with any edge removed is a tree so:
C(H*) ≥C(T)

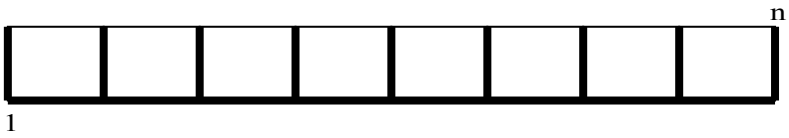
2) Consider a "full walk" of the MST, i.e. follow each edge in both directions
- as we go to a new node

and
- as we return after recursive call.

Then cost of full walk = 2C(T)

But over approximations cost is ≤ cost of full walk as we back up we may shorten the path by triangle inequality, hence we have a 2 approx.

Fact: The method can be that bad, consider the following example:



All edges indicated have length 1. All missing edges have length inferred by the triangle inequality.
The optimal tour clearly has length n.

Choose an unfortunate MST (as per the heavy lines). The tour, starting at 1, covers the bottom then top of each vertical line, then moves down and right (at a cost of 2) to the bottom of the next vertical line. This process ends at node n. This has cost 3n/2 – 2. The return to 1 costs n/2, for a total of 2n-2.

It turn out we can do better with a similar, though more sophisticated approach due to Christofides.

The key idea of 2-MST was to get a "cheap" subset of the edges on which you can do a tour.

New Idea: Eulerian tour (recall Bridges of Könnigsberg). Let G be any connected multigraph (maybe more than one edge (u,v)) in which every node is of even degree. Then it is possible to start at an arbitrary point and take a walk traversing every edge exactly once, returning to the start point. Indeed the algorithm to do this is very easy.

Algorithm Euler-Tour: Start anywhere, at each point go to a new node if possible, otherwise take a new edge to an old node. This must return to start point, but may miss some edges; "splice" them in. E.g. if there is an unused edge at v (and v has been visited) after first visit to v take a formerly unused edge from v, this walk uses formerly unused edges and returns to v. Pick up old tour from that point.

Observe: Our "2 copies" of MST was such a multigraph.

How do we get a cheaper Eulerian graph?

1) Find MST. This is the cheapest connected graph.

2) Consider nodes of <u>odd</u> degree in MST (they can use 1 more edge each).

3) Find Minimum Weighted Matching on nodes of odd degree.
   [Pair these nodes so some of the chosen edges are minimized. This can be done in $O(n^{2.5})$ time, though the method is sophisticated]

Do Eulerian tour on multigraph of MST plus MWM. Take "shortcuts"?

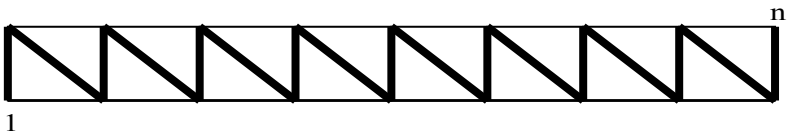
Cost: 1) $C(MST) \leq C(H^*)$

2) Consider the odd degree nodes we used for matching. Let H_o* denotes optimal TSP tour on these nodes and M_o is the matching cost. Consider every second edge of H_o* (there are 2 choices). M_o is at most the cost of either of these (as they are matching) so $M_o \leq C(H_o^*)/2$

But the TST on a subset of nodes costs at most as much as the TSP on all (due to the triangle inequality).
So from 1) and 2) and our Eulerian tour with shortcuts

$$C(A) \leq C(MST) + M_o$$
$$\leq C(H^*) + C(H^*)/2$$
$$\leq 3C(H^*)/2$$

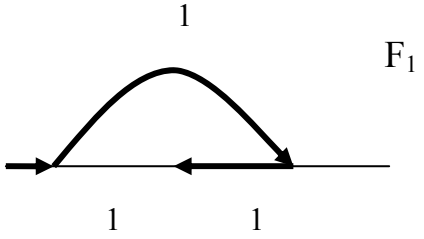
Fact: Our approximation can be this bad



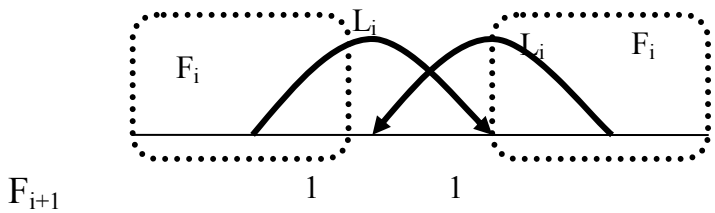
Consider essentially the same "ladder" example, except edges (2i,2i+1) also have length 1
Choose the (unfortunate) MST as the path from node 1 to node n (length n-1). Then the only nodes of odd degree are 1 and n (distance n/2). Our approximation method gives a tour of length 3n/2 - 1 instead of n

Other approaches to TSP
Nearest Neighbour approach to extending path. That is, at each step extend the path to the nearest unvisited node. At the end, we must return to the start node,
This works reasonably in "practice", but can give an approximation ratio, ρ, of (lg n)/3. The bad case occurs when the optimal is to simply go around a circle in unit steps. Unfortunately, we again make bad choices when given several edges of the same length.

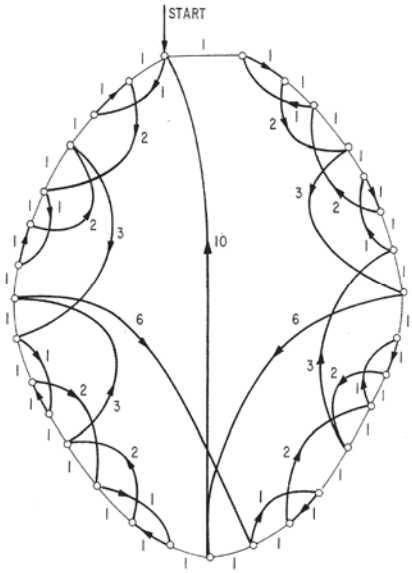
Construction: The base case, F_1:



In general, F_{i+1}:



Choosing the L_i to be the largest permitted by the triangle inequality so that we take these edges, we can force $L_1 = 2$ and $L_i = (6i2^i + 8\ 2^i + (-1)^i - 9)/9$. This gives the desired ratio. It can be shown that the method can produce no worse result.



Another approach is Nearest Insert, that is to start with a subtour and add in the cheapest new node. This gives $\rho \leq 2$.

Often the TSP to be solved is actually Euclidean distance in the plane. In such a case one can start with ant tour (even nearest neighbour) and improve it by detecting whether any edges cross. So if a pair of edges, say AD and BC crossed, them by opposing pairs of side edges. It can be shown that one of the choices will produce a single shorter cycle tour, while the other will give two disjoint cycles.