Thursday, Nov. 17

1. Welcome!
2. Homework Inventory
   - ↳ get in touch about anything you haven't turned in
3. here out → project work time
4. watch out for pregentation schedule email.
5. Questions?
6. TSP
7. outro
   - ↳ schedule!

Today we're talking about the traveling salesperson problem: finding the minimum cost hamiltonian cycle in a complete graph. We saw last time that finding a hamiltonian cycle is *hard*, and that's (literally) just the starting point for TSP. Nevertheless, they come up in practice, and we gotta deal with them somehow. Let's see what we can do!

Often the TSP is stated as a decision problem, so let's do that first:

TSP Decision: Given an edge weighted K_n and integer M, does K_n have a hamiltonian cycle of weight at most M?

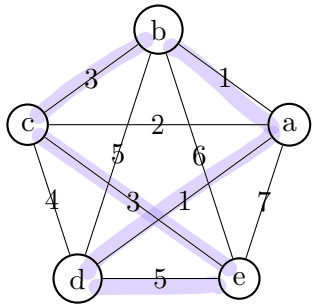
Example: Suppose we want to determine if a graph G has a hamiltonian cycle. Create an instance of the TSP decision problem to solve this question.

weight K_n as follows:

- 1 if $e \in G$
- 2 if $e \in \bar{G}$

does K_n have a ham. cycle of weight $\leq n$

Example: Find the minimum cost hamiltonian cycle in the graph below.



val = 13

Follow up: Suppose we wanted to brute force this, meaning we need to check every possibility. How many do we need to check? 12. What if our graph was a K_n? $\frac{(n-1)!}{2}$.

So this is a problem, but also allows us to talk about the complexity of TSP: NP-Complete. Without all the complexity details, it means that (1) given a potential solution, we can quickly verify that it satisfies (or doesn't) the decision problem, (2) there's a brute force algorithm to find all possible solutions and (3) it's in a group of problems that can all be reduced to each other. This is like what we did above, reducing HamCycle to TSP.

Example: What would this verification look like for TSP? Specifically, what would a potential solution be, and what would you need to check?

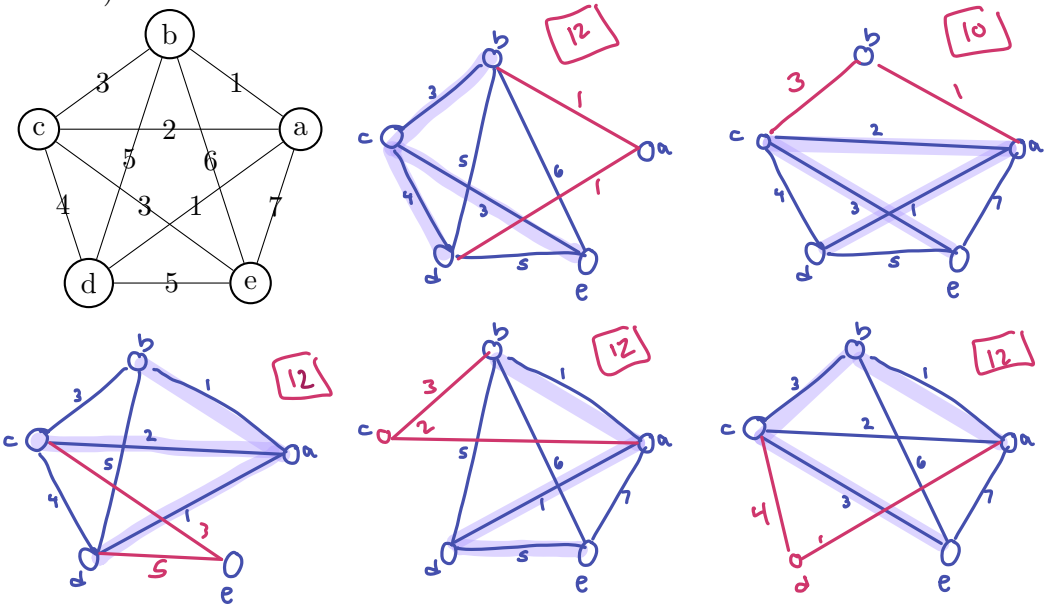
↳ [1] hamiltonian cycle [2] weight smaller than M

Our work on TSPs will fall into two camps: bounds on what the value of the minimum cycle is and heuristics on how to find an approximate solution. Let's try a bound first.

MST Bound: For a weighted K_n, let T_v^* be graph formed by constructing a minimum spanning tree of $K_n - v$ and adding in the two lowest cost edges incident to v. Then, if C is a optimum TSP cycle,

$$w(C) \geq \max\{w(T_v^*)\}$$

Example: Here's the K_5 from our first example. Calculate $w(T_v^*)$ (think back to Kruskals!) for each vertex and calculate the MST bound.



Claim. *The MST bound is correct.*

Pf: let C be an optimum TSP cycle. Then for any vertex v, C-v is a path, which is a spanning tree of $K_n - v$. Let T be an MST for $K_n - v$

then $w(C-v) \geq w(T)$

$w(C) = w(C-v) + w(e_1) + w(e_2)$ ← e_1, e_2 incident to v along cycle C.

$\geq \underbrace{w(T) + w(m_1) + w(m_2)}_{w(T_v^*)}$ ← m_1, m_2 smallest weight edges incident to v.

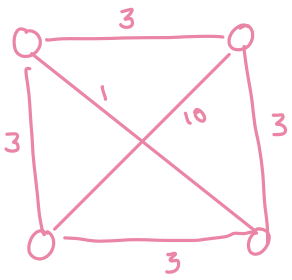
∎

A reasonable idea for making an optimum cycle may be to take the smallest edge you can at any stage, then connect the cycle once you've visited all the vertices. This is exactly the *nearest neighbor heuristic*.

Nearest Neighbor Heuristic: Starting at any vertex, greedly construct the shortest path, then connect the initial and final vertices.

Example: The optimum TSP cycle you constructed for our K_5 is the nearest neighbor heuristic applied to _____.

Example: Weight a K_4 so that no application of the nearest neighbor heuristic produces the minimum cycle. How "bad" might nearest neighbor be?



as bad as you want.

nn = 17 always

min cycle = 12.

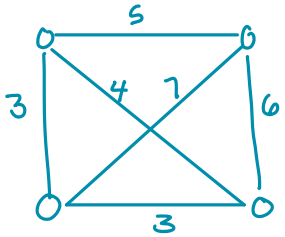
In practice, it's not actually all that bad, about 1.26 times the optimal value. We can even guarantee it provided we restrict to a *Euclidian TSP*. We'll get to that in a second, but we'll pause here to mention a couple more heuristics that build a cycles over time.

- Cheapest insertion: Add vertex that is cheapest to insert. At most 2 times optimum.
- Nearest insertion: Add vertex that is closest to any of vertices already in the tour. At most 2 times optimum.
- Furthest insertion: Add vertex that is furthest away from any of vertices already in the tour. At most $\log_2(n) + 1$ times optimum. Better in experiments than previous.

We'll transition here to talk about the Euclidian TSP, often denoted $\triangle$TSP because the Euclidian TSP problem is required to satisfy the triangle inequality on edge weights. Specifically,

$$c_{uw} + c_{wv} \geq c_{uv}$$

which makes some sense if we're talking about the original problem measuring traveling distances between cities. We'll build two approximation algorithms for problems of this type. In this weird open space between then and now, draw a quick $\triangle$TSP problem on 4 vertices.



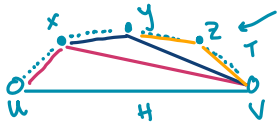
K approx : val ≤ K · opt

there is a circuit that visits every edge! ↕ iff all even degree.

Our first algorithm needs a theorem before we build it.

Claim. *Let K_n be an instance of $\triangle TSP$ and let G be any Eulerian spanning subgraph of K_n. If C is the optimum TSP cycle, then $w(C) \leq w(G)$.*

Proof: Let $T = \{v_0, v_1 \ldots, v_m = v_0\}$ be an Eulerian traversal of G. Build a cycle H based on how T traverses the vertices, ignoring repeated vertices. Note H may include edges not in T! However, since K_n satisfies the triangle inequality, $w(H) \leq w(G)$.



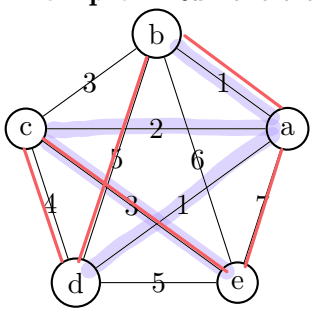
s/p x, y, z are already seen in H

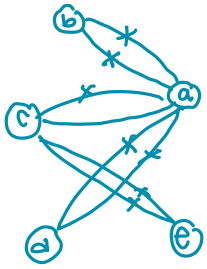
$w(uv) \leq w(ux) + w(xv) \leq w(ux) + w(xy) + w(yv) \leq w(ux) + w(xy) + w(yz) + w(zv)$

Since H is a cycle, $w(C) \leq w(H) \leq w(G)$.

Tree Algorithm: Construct a minimum spanning tree of K_n, double every edge to make an Eulerian multigraph G. Return the cycle H as we do in the proof.

Example: Run the tree algorithm on the graph below.





MST

T: abadaceca

H: abdcea

w(H) = 20

Claim. *This algorithm is a two approximation of $\triangle TSP$.*

Proof: let C be an optimum TSP cycle. let T be a min spanning tree. Then $w(T) \leq w(C)$. moreover, by construction,

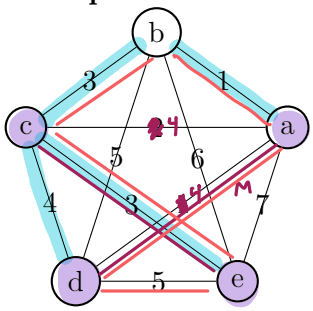
$$w(C) \leq w(H) \leq 2w(T) \leq 2w(C)$$

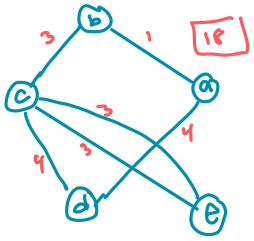
↑ cycle constructed from alg.

Our other $\triangle$TSP algorithm will also rely on Eulerian graphs, but will do even a bit better. It's also going to use a few of our optimization techniques!

Christofides' Algorithm: For an instance K_n of $\triangle$TSP, let T be a minimum spanning tree. Let $X = \{v \in V : \deg_T(v) \text{ is odd}\}$. Construct M to be a minimum cost perfect matching of the vertices of X. $T + M$ is an Eulerian (possibly multi)graph, so use its Eulerian tour to construct a cycle H as we've done before. The weight of H is at most $w(T) + w(M)$.

Example: Run Christofides' algorithm on the graph below.





T: a b c e c d a

H: a b c e d a

w(H) = 16

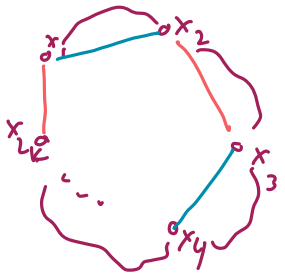
$w(C) \leq w(T) + w(M) \leq$?

Claim. *This algorithm is a 3/2 approximation of $\triangle$TSP.* $w(C) \leq w(T) + w(M) \leq \frac{3}{2} w(C)$

Proof: let H be produced by the alg. s/p $\{x_1, x_2 \cdots x_{2k}\}$ are the odd deg. vertices of T, listed in the order they appear in C (opt)

$M_1 = \{x_1x_2, x_3x_4, \cdots\}$

$M_2 = \{x_2x_3, x_4x_5 \cdots\}$



$w(M_1), w(M_2) \geq w(M)$ where M is smallest

$w(C) \geq w(M_1) + w(M_2) \geq 2w(M)$

$w(M) \leq \frac{1}{2} w(C)$

$w(T) \leq w(C)$

So $w(T) + w(M) \leq \frac{3}{2} w(C)$