CS 163 & CS 265: Graph Algorithms

Week 4: More paths

Lecture 13: Approximate traveling salesperson tours

David Eppstein
University of California, Irvine

Fall Quarter, 2022

The traveling salesperson problem

For distances:

Input is a matrix of distances between n points, satisfying

- Symmetric: $D[i,j] = D[j,i]$
- Positive: $D[i,j] \geq 0$ $(0 \Rightarrow i = j)$
- Triangle inequality: $D[i,j] + D[j,k] \geq D[i,k]$

Goal: find cyclic order using each point exactly once, minimizing sum of distances between consecutive points

For graphs:

Input: connected undirected graph with edge lengths > 0

Goal: find a walk that visits each vertex at least once, returning to start vertex, minimizing sum of edge lengths

Converting between matrices and graphs

To convert a distance matrix D into an equivalent graph problem:

- Use a complete graph: an undirected graph with an edge between each pair of vertices
- Weight each edge by the distance between endpoints
- If you get a solution with repeated vertices, omit them
- Triangle inequality implies that omitting repetitions cannot increase tour length

To convert a graph into an equivalent distance matrix:

- Compute matrix of all-pairs shortest path distances
- Turn any tour of the resulting matrix into a walk in the graph by concatenating together shortest paths between consecutive vertices in the tour

Some applications

- Plan the travel of actual traveling salespeople from city to city within their sales area, visiting each city with as little travel time or cost as possible
- Plan other physical movements that must visit each of a system of points (trucks and delivery points; 3d printers and points where they must deposit material)
- Order the rows of a product comparison table so more-similar products are next to each other
- Reconstruct long DNA sequences by reading shorter subsequences and finding a way to order them so they overlap as much as possible

Hardness

TSP is NP-hard. Even for graphs with all weights $= 1$, finding a walk with total length $\leq n$ is NP-complete.

- NP: yes-no problems that can be solved by a brute force search over solutions of polynomial size. For weight-1 TSP, try all length-n walks, checking whether any walk visits all vertices.
- NP-complete: In NP and all other NP problems can be translated into it. E.g., we could find an input to a Boolean circuit that makes the circuit output true by translating into an equivalent TSP problem and then using a TSP solver.
- TSP itself is not in NP because it's not a yes-no problem. That's why it's called NP-hard rather than NP-complete.

Implications of hardness

All NP-complete problems including TSP can be solved by exponential-time brute force search algorithms

Some of them have faster (still exponential time!) algorithms based on other methods; we will see one for TSP next time

None have known polynomial-time algorithms; if you found one, it would give you a general method for automatically translating all brute force searches into much more efficient algorithms.

We think a polynomial-time algorithm does not exist but we don't know how to prove that.

Finding a polynomial-time algorithm or proving its nonexistence would win a $1,000,000 prize

Approximation algorithms

If we can't solve it quickly and exactly, what can we do?

- Solve it slowly and exactly (next time)
- Solve it quickly and less exactly, but still with some guarantees on solution quality (this time)

Approximation algorithm:

- Algorithm for producing inexact solutions
- Should run in polynomial time
- Should produce solution of guaranteed quality

How to measure approximation quality?

Approximation ratio of an algorithm: The largest possible value

algorithm's solution quality/optimal solution quality

that any worst-case input to the algorithm can cause it to produce (for minimization problems like TSP where bigger ratios are bad)

Goal: Find an approximation algorithm whose approximation ratio is as close to 1 as possible

Overview of TSP approximation

- Approximation ratio 2 is easy using minimum spanning trees
- Approximation ratio $\frac{3}{2}$ has been known since 1976, using minimum spanning trees + matching
  [??]
- Recent breakthrough: $\frac{3}{2} - \frac{1}{10^{36}}$ [?]

  Very rough sketch: random spanning trees + matching
- Several special cases have better approximations, e.g. for unweighted graphs (or all edge lengths 1) there is an algorithm with ratio $\frac{7}{5}$ [?]
- For the general problem, NP-hard to approximate with ratio better than $\frac{123}{122}$ [?]

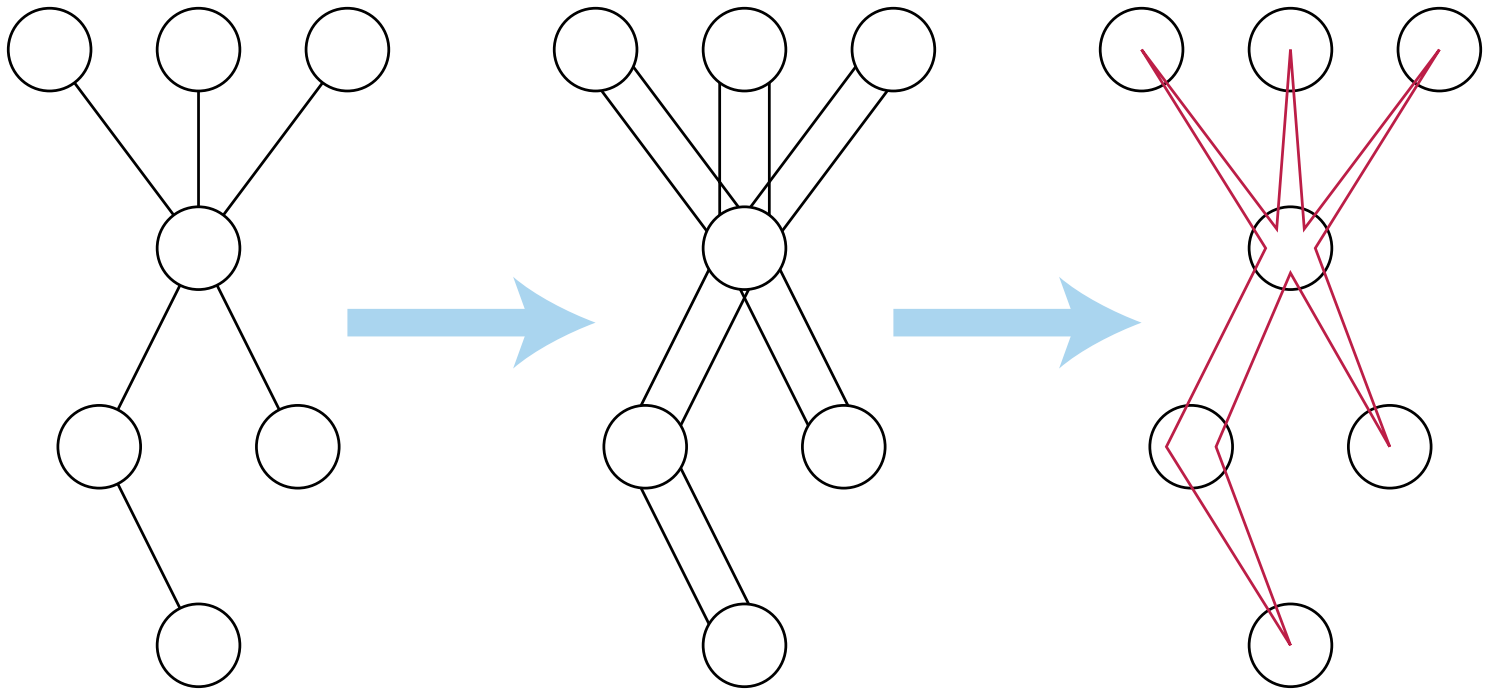
  Still a big gap between $\frac{123}{122}$ and $\frac{3}{2} - \frac{1}{10^{36}}$ open for research

2-approximation

Use graph version of TSP, so repeated vertices are allowed

Find a minimum spanning tree

Make two copies of each tree edge $\Rightarrow$ multigraph, all degrees even



Find an Euler tour and return it as the approximate TSP tour

Why is this a 2-approximation?

Approximate tour $= 2$(minimum spanning tree)

Optimal tour $\geq$ path formed by removing any one of its edges

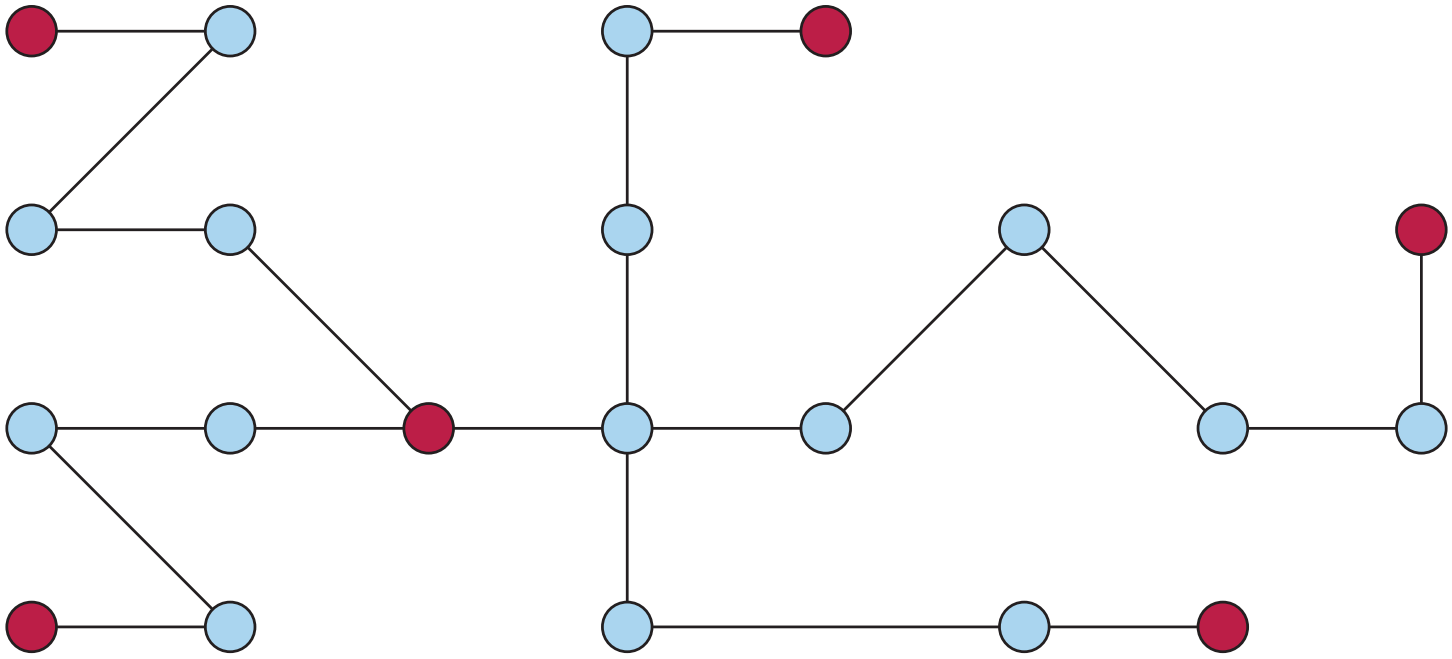
But this path is a spanning tree!

So length of minimum spanning tree $\leq$ length of optimal tour

Put it together: approximate tour ≤ 2 optimal tour

3/2-approximation, step 1

Construct a minimum spanning tree

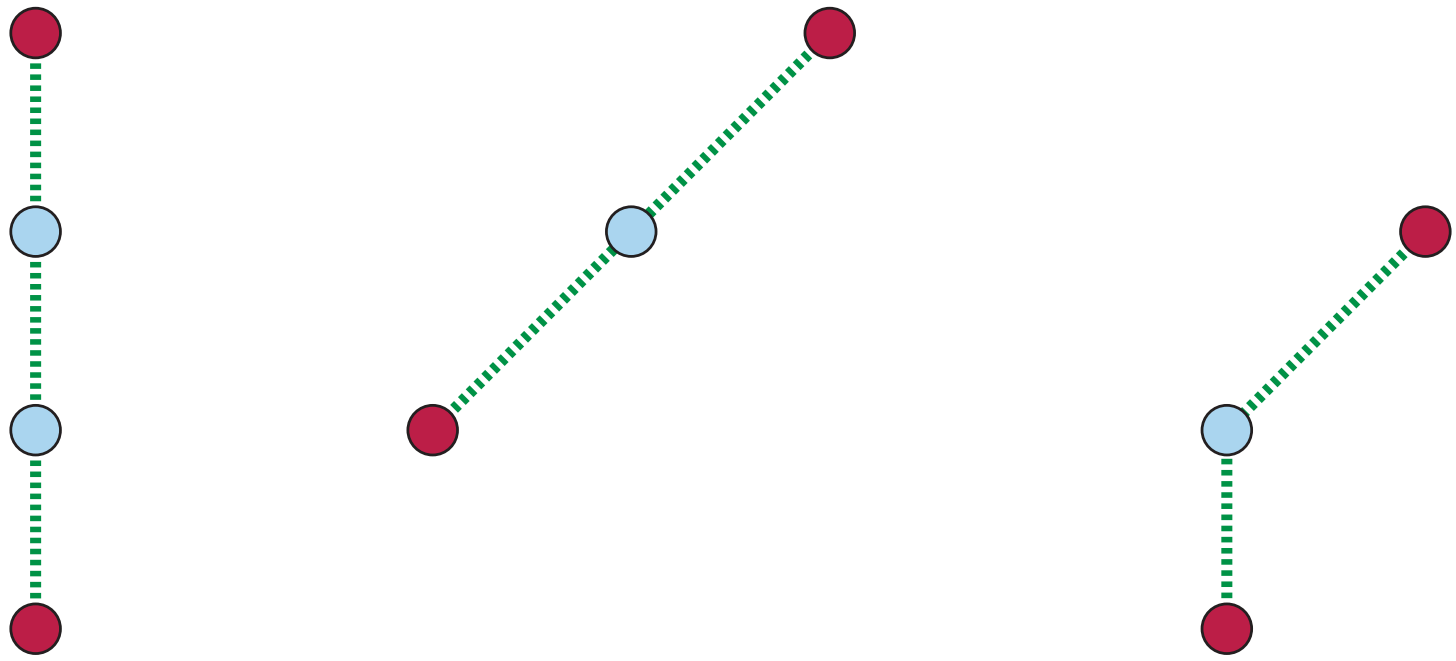


Identify its odd vertices
(odd by their degree in the spanning tree, not in the whole graph)

Before using an Euler tour, we need to make their degrees even,
more cheaply than doubling every edge in the whole tree

3/2-approximation, step 2

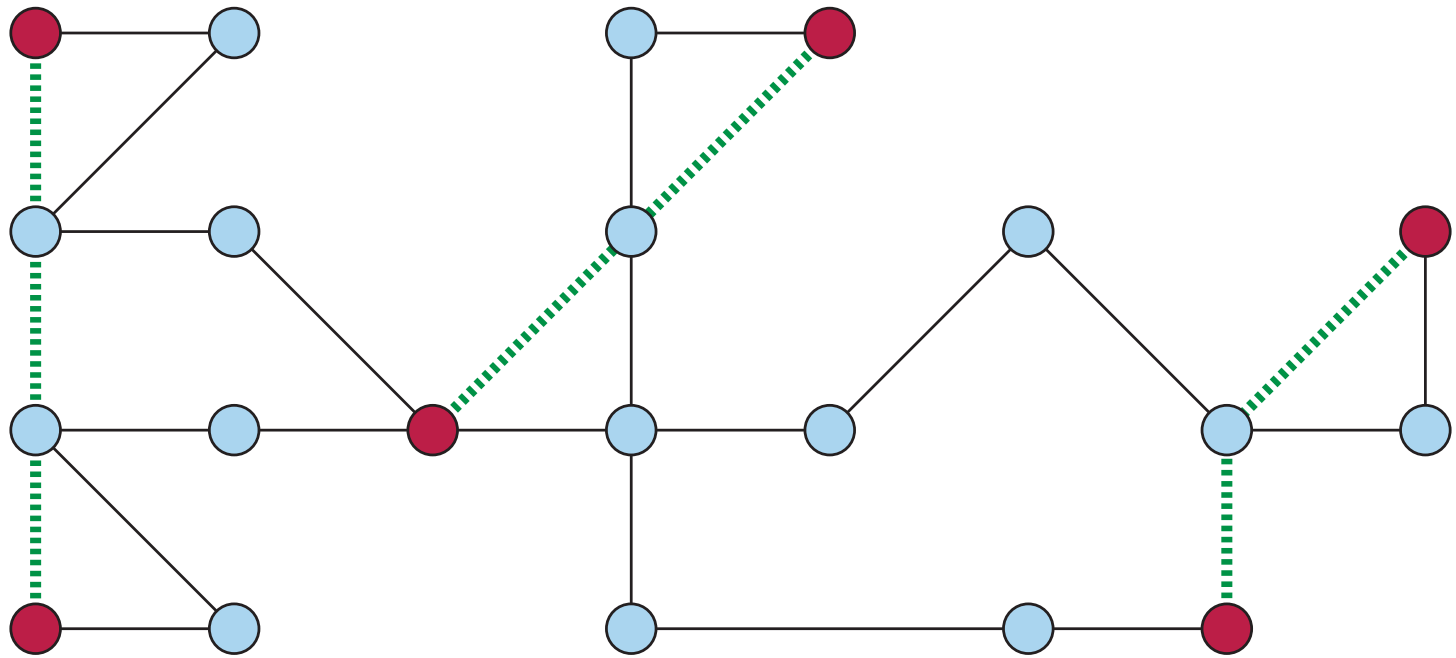
Handshaking lemma: The number of odd vertices we found is even



Connect them in pairs by shortest paths from the original graph choosing pairings in a way that minimizes the sum of path lengths (this is a problem of matching; we will discuss matching in week 8)

3/2-approximation, step 3

Put minimum spanning tree and paired paths together in one graph
(or multigraph if the paths and spanning tree have shared edges)

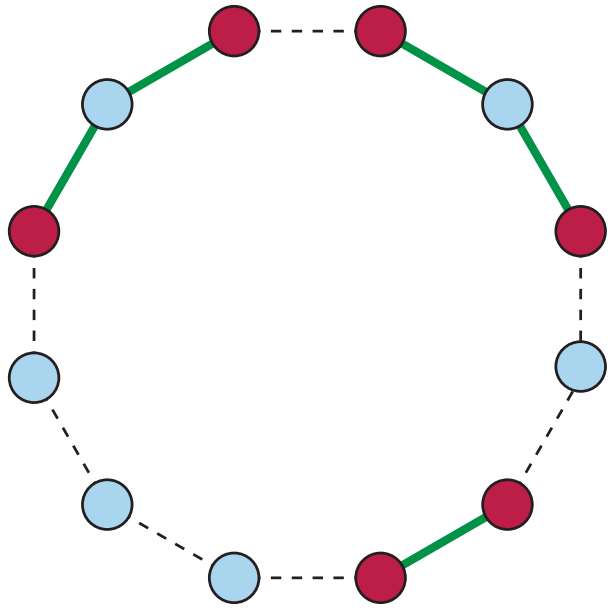


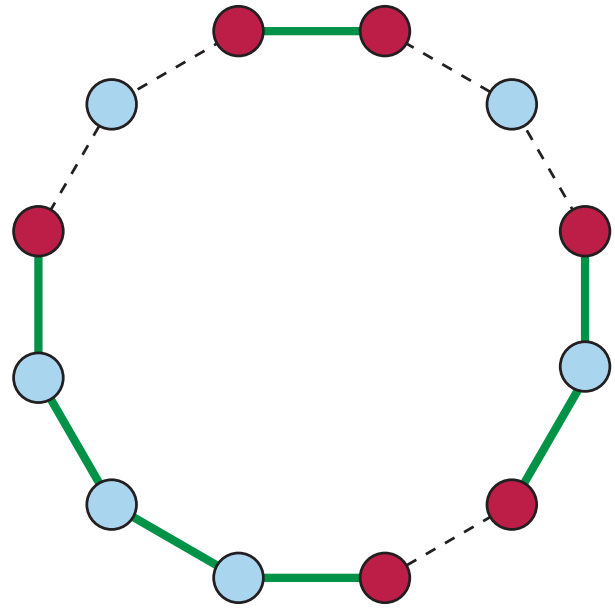
Find an Euler tour and return it as the approximate TSP tour

Why is it a 3/2-approximation?

Minimum spanning tree $\leq$ optimal tour as before

There are two ways to pair consecutive odd vertices in the optimal tour, by paths around the tour. Together they add to whole tour





The pairing by shortest paths that we find is at least as good as the best of these two ways $\Rightarrow \leq \frac{1}{2}$ optimal tour

Morals of the story

Equivalence between graph and distance versions of traveling salesperson

What it means for traveling salesperson to be NP-hard:

We don't know how to solve it in polynomial time

Although we think there is no polynomial-time solution, we also don't know how to prove it is impossible

The general concept of an approximation algorithm, and the 2- and 3/2-approximations for traveling salesperson

References I