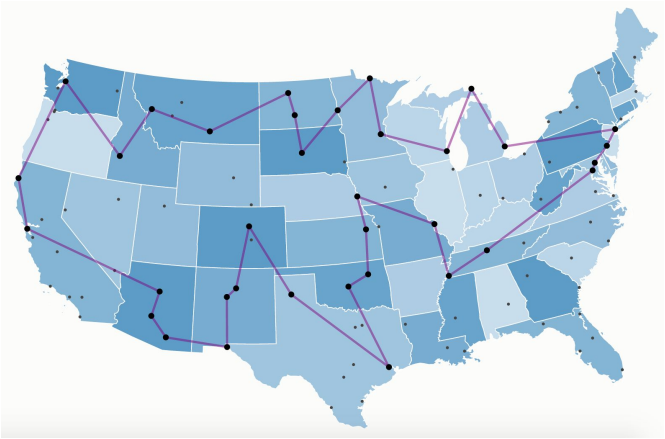
CIS 1921

Lecture 10: TSP Techniques

Ishaan Lal ilal@seas.upenn.edu

Traveling Salesman Problem

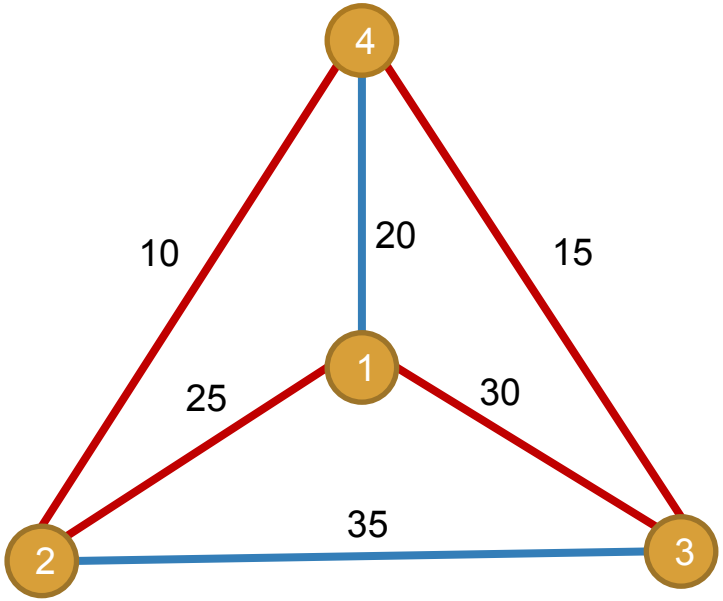
- **Problem:** in weighted complete graph, find a tour of minimum total cost that visits every vertex exactly once and returns to starting vertex
  - Graph can be directed or undirected
- Applications in routing, logistics, etc.
- NP-complete!



Preliminary Notation

- We'll look at complete directed graphs (parallel edges, but no self-loops) with n nodes, m edges
  - Undirected graphs are often a special case
- Directed edge $(i, j) = i \to j$ has weight $w(i, j)$
- We'll denote a **tour** as a permutation $v_1, v_2 \ldots, v_n$ of the vertices, which represents $v_1 \to v_2 \to \cdots \to v_n \to v_1$

Example



- For simplicity, examples will generally be drawn undirected
- Imagine each edge (i, j) is really two parallel edges with same cost
- **Optimal tour cost:**

$$10 + 25 + 30 + 15 = 80$$

Attempt: Solving TSP with CP?

- Define 0/1 variables x_{ij} indicating if edge (i, j) is in the TSP tour
- Each vertex is visited exactly once:

$$\sum_{j \neq i} x_{ij} = \sum_{j \neq i} x_{ji} = 1, \qquad \forall 1 \leq i \leq n$$

- Want to minimize total cost:

$$C = \sum w(i, j) \cdot x_{ij}$$

An issue

- This CP formulation allows “subtours” rather than forcing one contiguous tour!

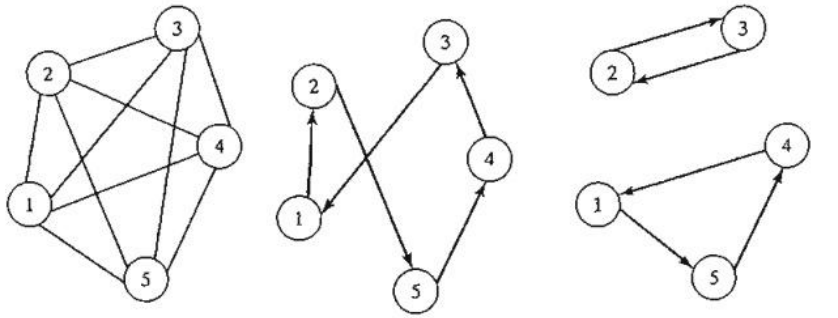


FIGURE 9.11

A 5-city TSP example with a tour and subtour solutions of the associated assignment model

- How to fix this?

Disallowing subtours

- For each possible subtour of vertices S, make sure that we take less than $|S|$ edges between them
- As a constraint:

$$\sum_{i \neq j \in S} x_{ij} < |S|, \qquad \forall S \subset V, |S| > 1$$

- Problem: there are exponentially many subtours!
  - Ways to fix this or add constraints lazily...
  - But in general CP is not state-of-the-art for TSP

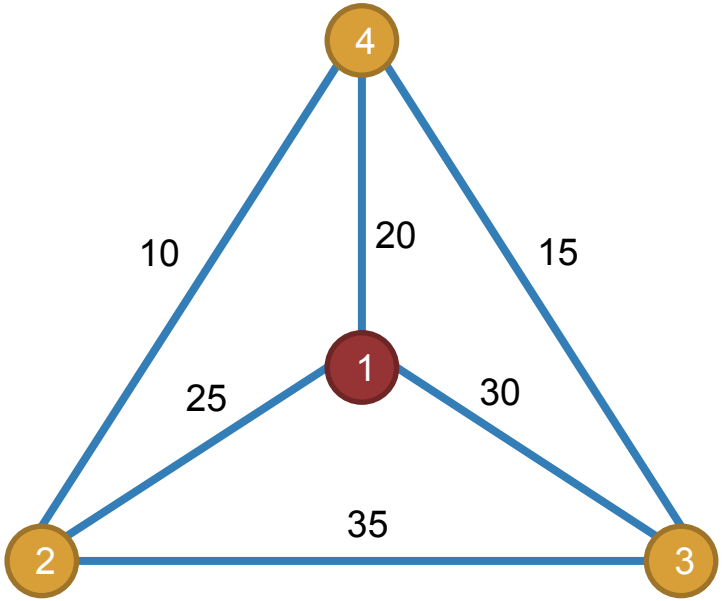
Traveling Salesman Problem

- Observation: TSP is an approximation-friendly problem
  - In practice, “good enough” usually is good enough!
- **Goal:** design efficient heuristics that give an empirically cheap tour (possibly not quite cheap*est*)
- Today: **constructive heuristics**
  - Start from nothing and iteratively build up partial solution

Nearest-Neighbor (NN)

- Start at any vertex u. Pick nearest unseen out-neighbor v of u and add it to end of tour, then repeat starting from v. Continue until all vertices added.
- Pros:
  - Simple, intuitive, and relatively efficient
  - Empirically OK, esp. on Euclidean TSP
- Cons:
  - Greedy: can easily miss shortcut paths

Nearest-Neighbor (NN)



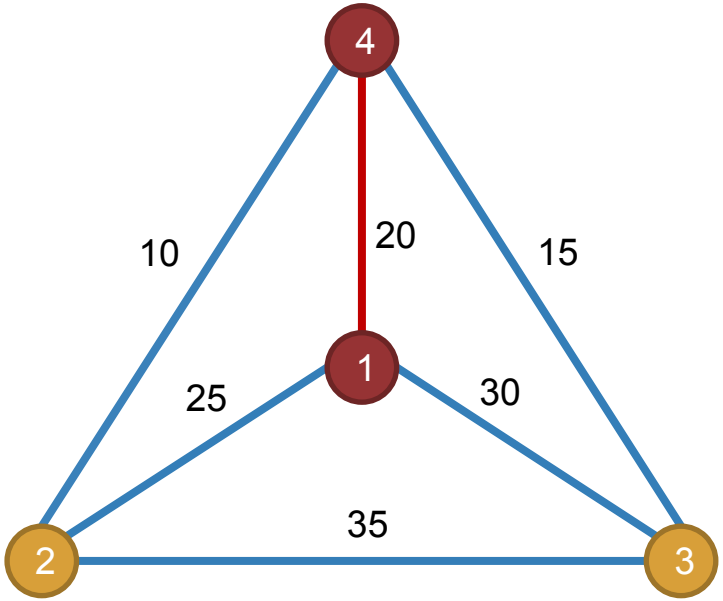
- Current tour:

  1

- Current cost:

  0

Nearest-Neighbor (NN)



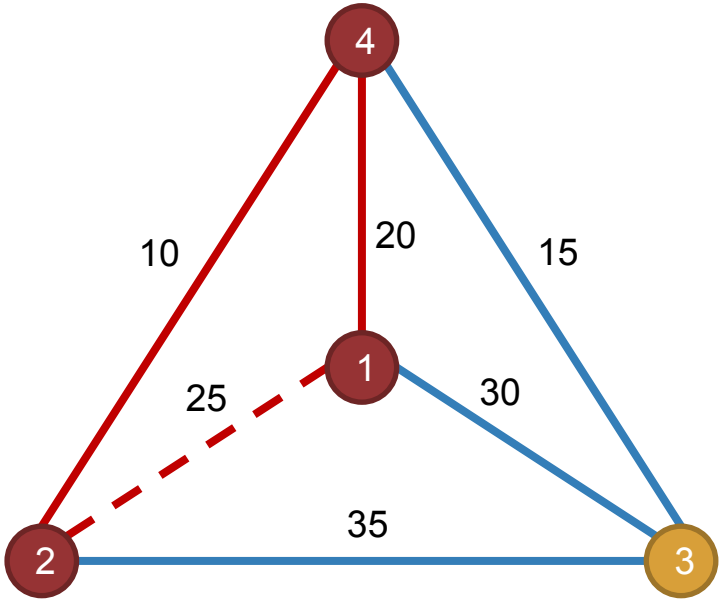
- Current tour:

  1, 4

- Current cost:

  20 + 20 = 40

Nearest-Neighbor (NN)



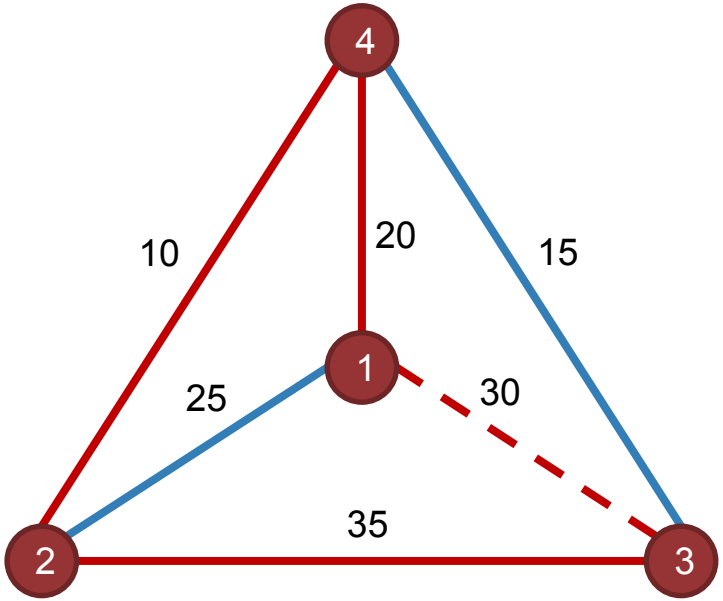
- Current tour:

  1, 4, 2

- Current cost:

  20 + 10 + 25 = 55

Nearest-Neighbor (NN)



- Current tour:

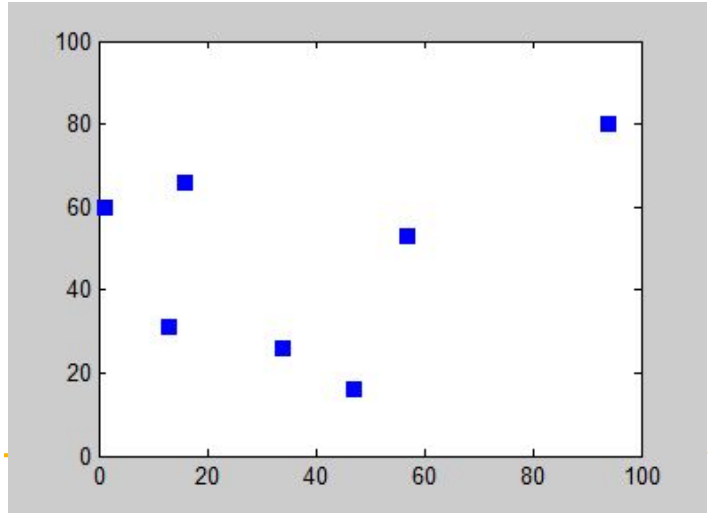
  1, 4, 2, 3

- Current cost:

  20 + 10 + 35 + 30 = 95

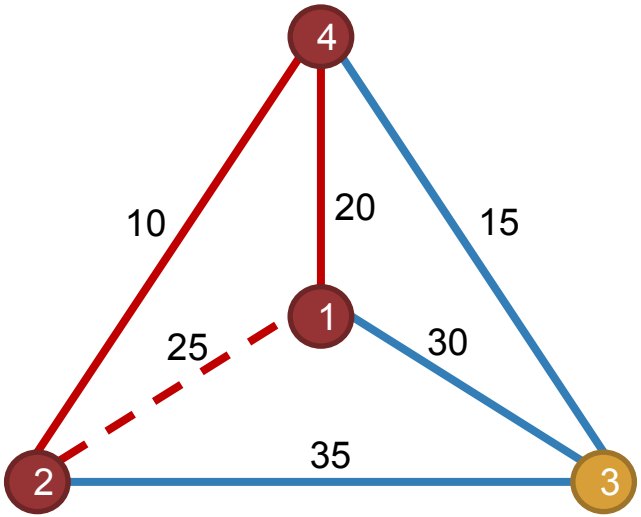
Nearest-Neighbor (NN)

- NN yields a path that is *on average* 25% longer than the true shortest path
- NN does not guarantee that the solution it comes up with will be good – there are instances of TSP where NN would return the worst path
- Solution changes depending on starting point



From NN to NI

- NN commits early – this might leave us with the situation that later cities are far apart.



- Current tour:

  1, 4, 2

- Current cost:

  20 + 10 + 25 = 55

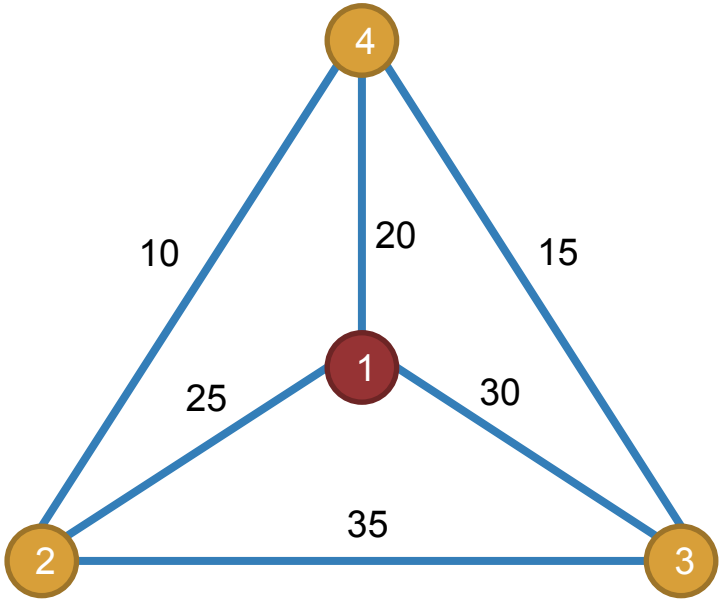
- **IDEA:** “Optimally” insert the nearest city.

Nearest-Insertion (NI)

- Start the tour T at any vertex
- Pick the nearest unseen out-neighbor v of **any** vertex in the tour
- Insert it into the tour $T = t_1, \ldots, t_k$ so that the total tour distance is minimized
  - i.e., find i s.t. $w(t_i, v) + w(v, t_{i+1}) - w(t_i, t_{i+1})$ is minimized
- Repeat until all vertices added to tour

- **Intuition:** still greedy, but not as greedy as NN – allow the partial tour to be modified

Nearest-Insertion (NI)



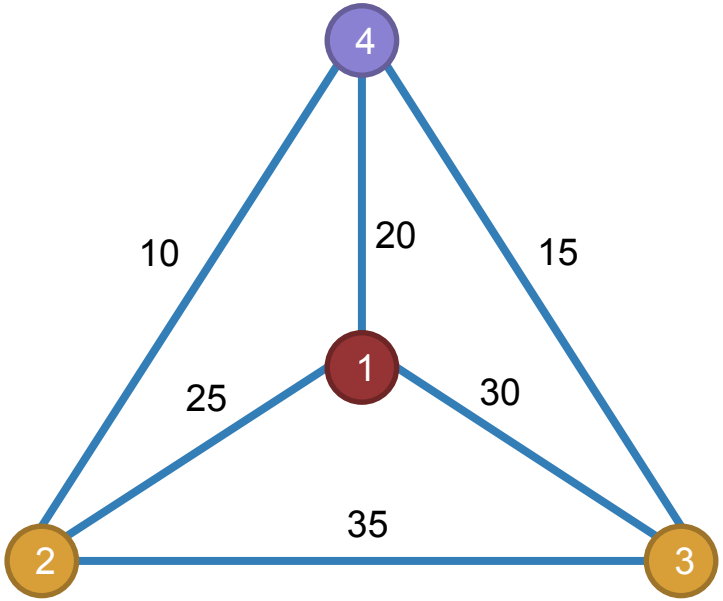
- Current tour:

  1

- Current cost:

  0

Nearest-Insertion (NI)



- Current tour:

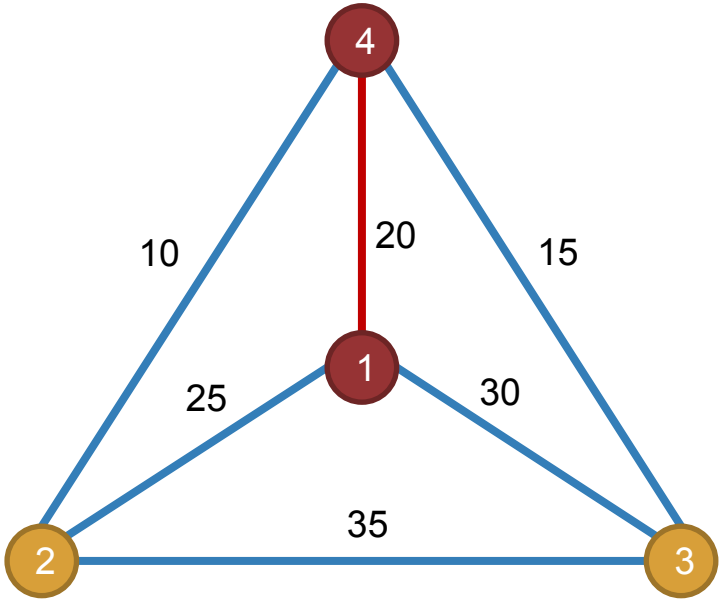
  1

- Current cost:

  0

- Next vertex: 4
  - Only one place to insert (up to rotation)

Nearest-Insertion (NI)



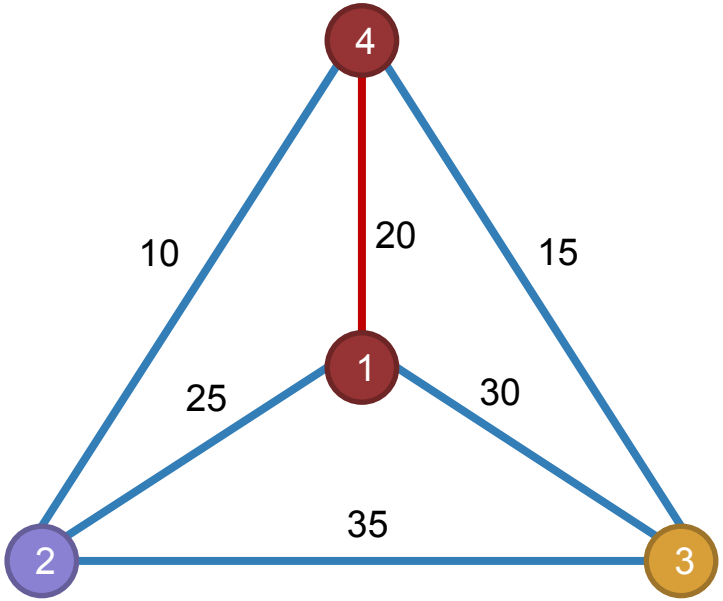
- Current tour:

  1, 4

- Current cost:

  20 + 20 = 40

Nearest-Insertion (NI)



- Current tour:

  1, 4

- Current cost:

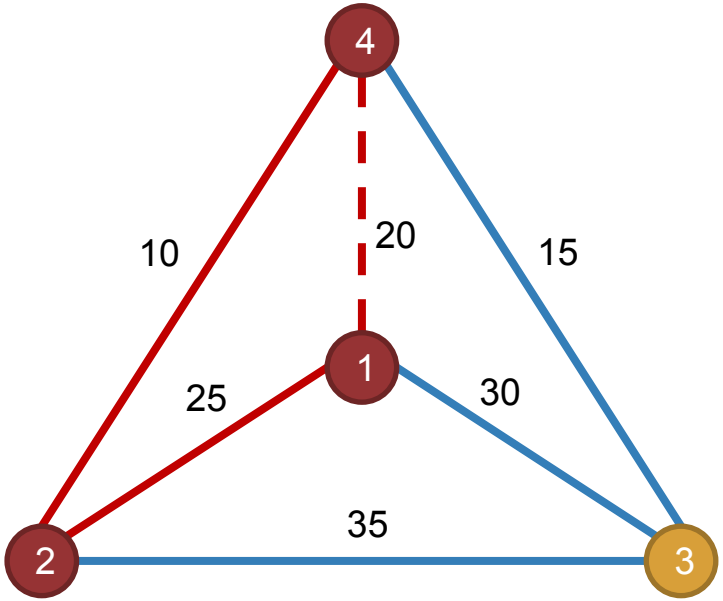
  20 + 20 = 40

- Next vertex: 2

After 1: $w(1, 2) + w(2, 4) - w(1,4) = 25 + 10 - 20 = 15$

After 4: $w(4, 2) + w(2, 1) - w(1,4) = 10 + 25 - 20 = 15$

Nearest-Insertion (NI)



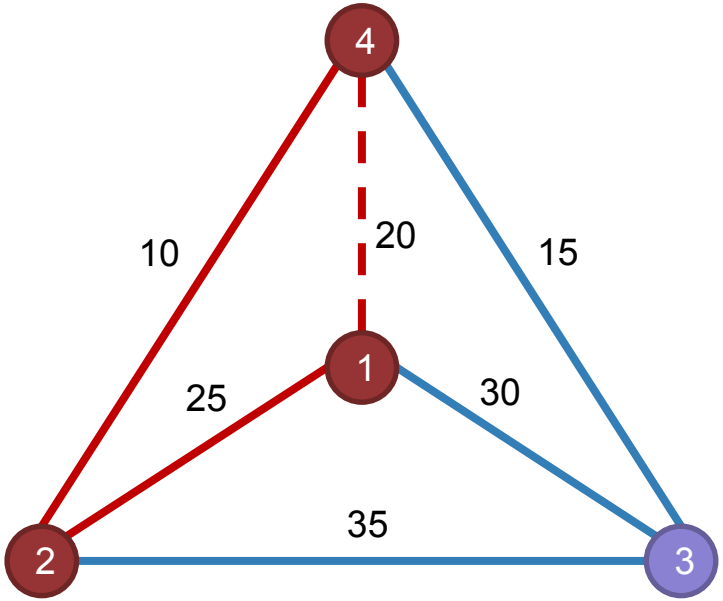
- Current tour:

  1, 2, 4

- Current cost:

  25 + 10 + 20 = 55

Nearest-Insertion (NI)



- Current tour:

  1, 2, 4

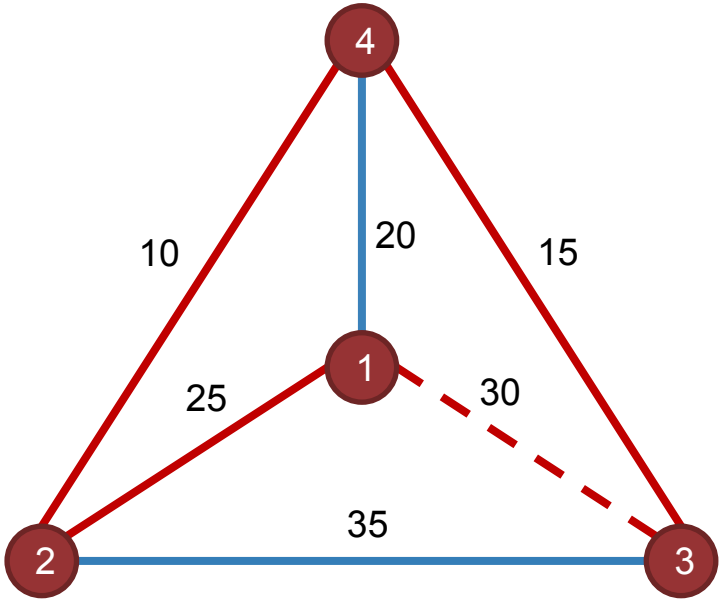
- Current cost:

  25 + 10 + 20 = 55

- Next vertex: 3

After 1: $w(1, 3) + w(3, 2) - w(1, 2) = 30 + 35 - 25 = 40$
After 2: $w(2, 3) + w(3, 4) - w(2, 4) = 35 + 15 - 10 = 40$
After 4: $w(4, 3) + w(3, 1) - w(4, 1) = 15 + 30 - 20 = 25$

Nearest-Insertion (NI)



- Current tour:

  1, 2, 4, 3

- Current cost:

  25 + 10 + 15 + 30 = 80

Nearest-Insertion (NI)

Worst Case Behavior:

$$\frac{length\_of\_nearest\_insertion\_tour}{length\_of\_optimal\_tour} \leq 2$$

Number of Computations:

The nearest insertion algorithm is algorithm is $O(n^2)$.

Farthest-Insertion (FI)

- Start the tour T at any vertex
- Pick the ~~nearest~~ farthest unseen out-neighbor v of **any** vertex in the tour
- Insert it into the tour $T = t_1, \ldots, t_k$ so that the total tour distance is minimized
  - i.e., find i s.t. $w(t_i, v) + w(v, t_{i+1}) - w(t_i, t_{i+1})$ is minimized
- Repeat until all vertices added to tour

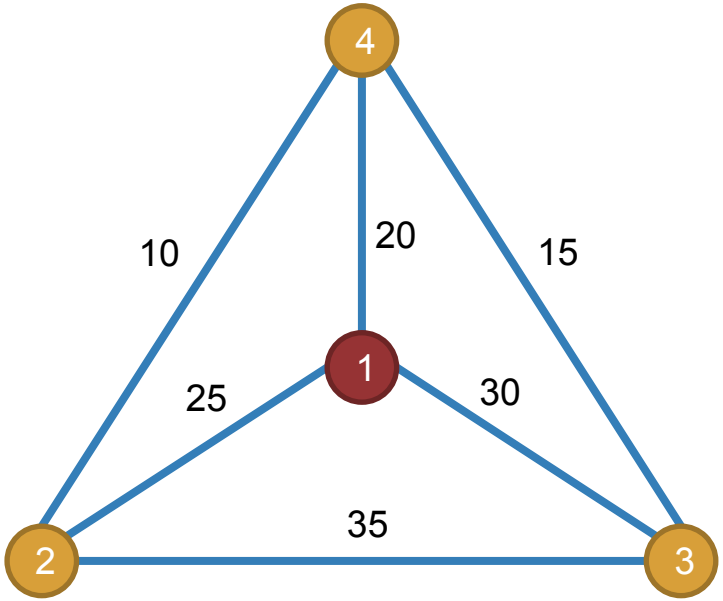
- **Intuition:** Start with the "difficult" vertices first to avoid getting into bad situations down the line.

Farthest-Insertion (FI)

where nearest insertion has a min. The intuitive appeal is that the method establishes the general outline of the approximate tour at the outset and then fills in the details. The early establishment of a general outline is appealing because we expect better performance when the number of nodes is small. Inserting nearby points late in the approximation is appealing because the short edges used late in the procedure are less likely to be accidentally deleted by some still later insertion.

(Rosenkrantz, Stearns, Lewis II, 1977)

Farthest-Insertion (FI)



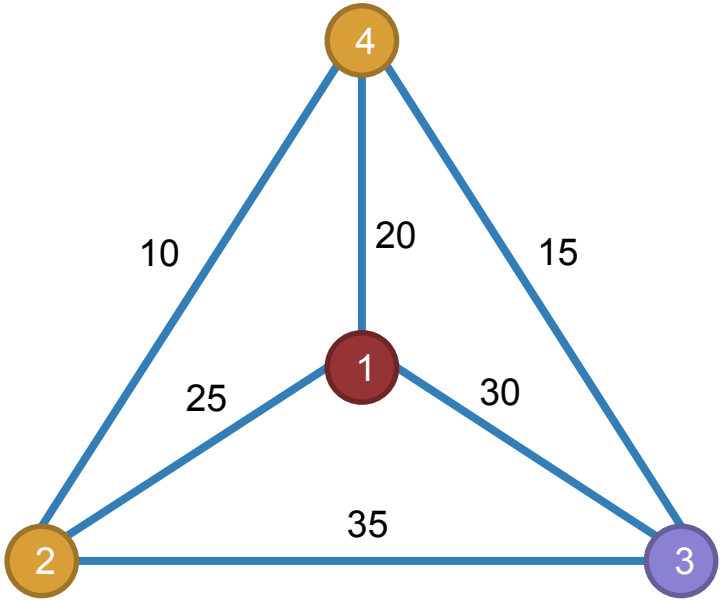
- Current tour:

  1

- Current cost:

  0

Farthest-Insertion (FI)



- Current tour:

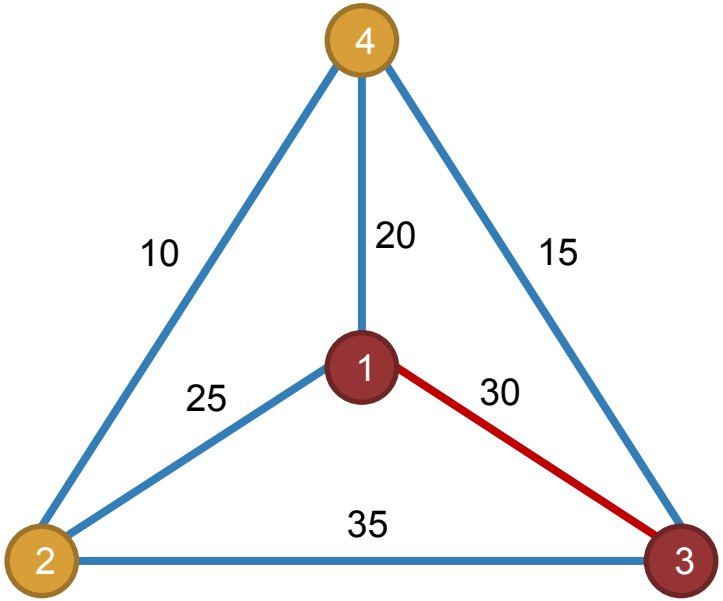
  1

- Current cost:

  0

- Next vertex: 3
  - Only one place to insert (up to rotation)

Farthest-Insertion (FI)



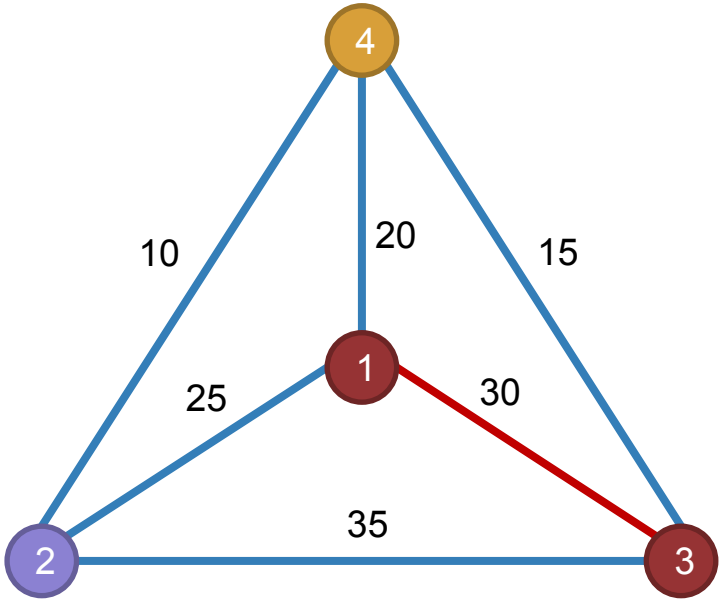
- Current tour:

  1, 3

- Current cost:

  30 + 30 = 60

Farthest-Insertion (FI)



- Current tour:

  1, 3

- Current cost:

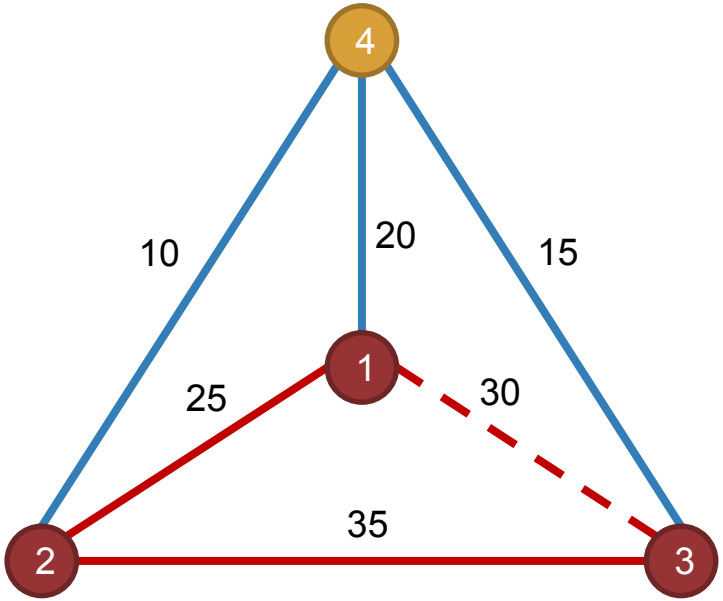
  30 + 30 = 60

- Next vertex: 2

After 1: $w(1, 2) + w(2, 3) - w(1, 3) = 25 + 35 - 30 = 30$

After 3: $w(3, 2) + w(2, 1) - w(1, 3) = 35 + 25 - 30 = 30$

Farthest-Insertion (FI)



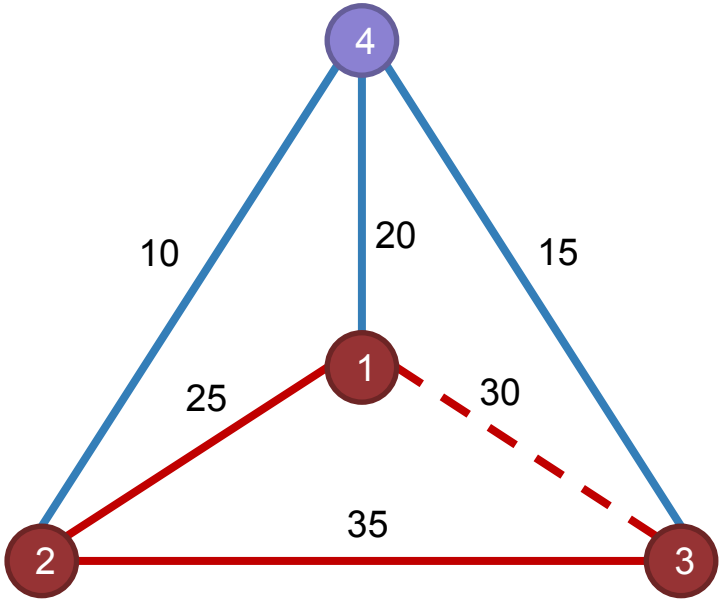
- Current tour:

  1, 2, 3

- Current cost:

  25 + 35 + 30 = 90

Farthest-Insertion (FI)



- Current tour:

  1, 2, 3

- Current cost:

  25 + 35 + 30 = 90

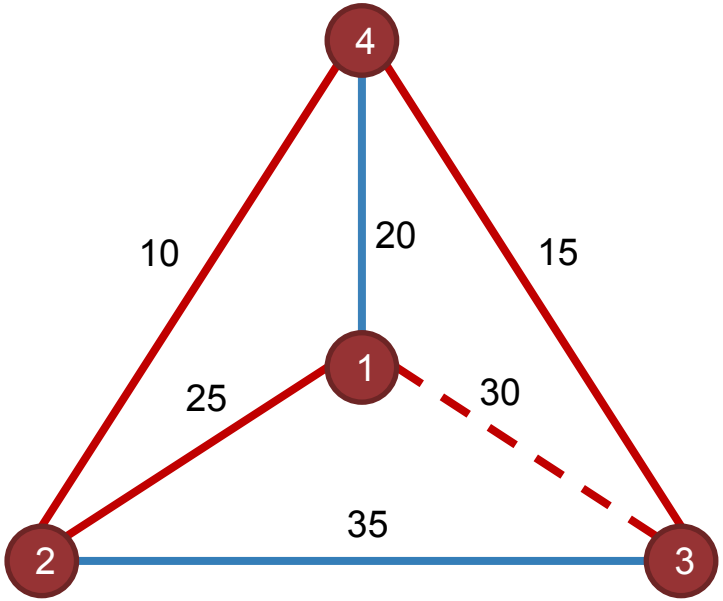
- Next vertex: 4

After 1: $w(1, 4) + w(4, 2) - w(1, 2) = 20 + 10 - 25 = 5$

After 2: $w(2, 4) + w(4, 2) - w(2, 3) = 10 + 15 - 35 = -10$

After 3: $w(3, 4) + w(4, 1) - w(3, 1) = 15 + 20 - 30 = 5$

Farthest-Insertion (FI)



- Current tour:

  1, 2, 4, 3

- Current cost:

  25 + 10 + 15 + 30 = 80

Farthest-Insertion (NI)

Worst Case Behavior:

$$\frac{length\_of\_farthest\_insertion\_tour}{length\_of\_optimal\_tour} \leq 2\ln(n) + 0.16$$

Number of Computations:

The farthest insertion algorithm is $O(n^2)$.

Insertion Heuristics

- Aims to be less naively greedy than NN
  - Unlike NN, can modify partial tour
- Somewhat more expensive than NN heuristic
- FI works pretty well in practice...
- ...but NI not so much.

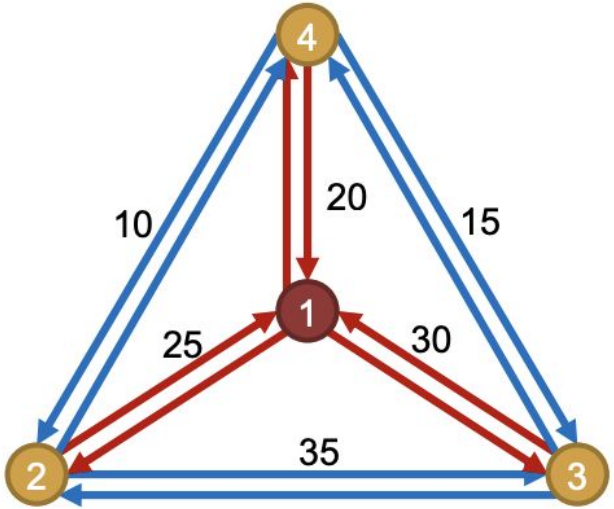
Farthest-Insertion (NI)

and the nearest neighbor. For example, when tried on problems obtained by placing 50 nodes randomly on a unit square, nearest insertion was from 7 to 22% worse than farthest insertion, nearest neighbor was from 0 to 38% worse, and cheapest insertion ranged from 7% better to 12% worse. The usual ranking was thus farthest insertion first, cheapest insertion second, nearest insertion third, and nearest neighbor last.

Savings Heuristic

- Pick any vertex x to be the "central vertex"
- Start with $n-1$ subtours: $x \to v \to x$ for all $v \in V - x$
- For each edge (i, j), where $i, j \in V - x$, compute its **savings** $s(i, j)$
  - $s(i, j) = w(i, x) + w(x, j) - w(i, j)$
- Sort edges in decreasing order of savings
- Repeat until only one tour remains:
- Let (i, j) be the next edge in sorted order
- If edges (i, x) and (x, j) are in our subtours, and i, j are not already in the same tour: replace (i, x) and (x, j) by (i, j)

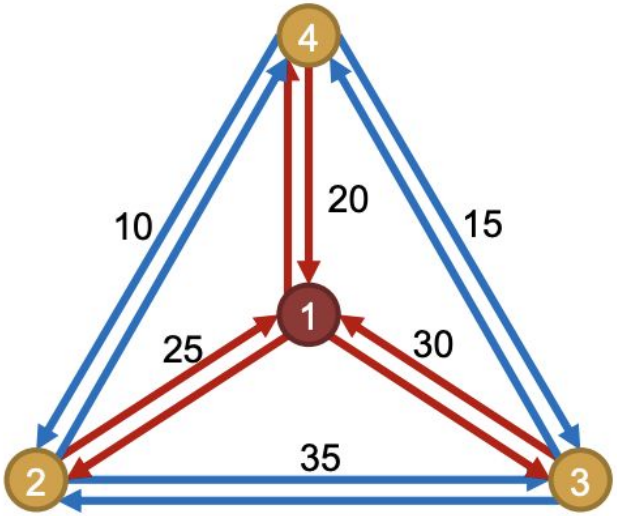
Savings Heuristic



- Current cost:

25 + 25 + 30 + 30 + 20 + 20 = 150

Savings Heuristic

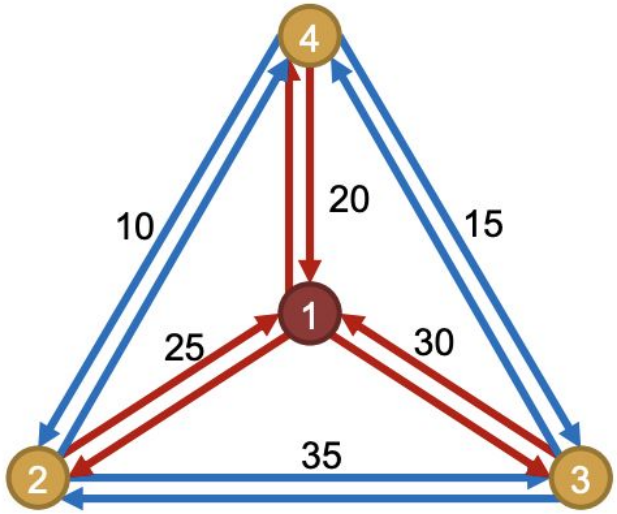


- Current cost:

25 + 25 + 30 + 30 + 20 + 20 = 150

(i,j)	Savings $s(i,j)$
$(2,3)$	$w(2,1)+w(1,3)-w(2,3)$ $=25+30-35=20$
$(3,2)$	$w(3,1)+w(1,2)-w(3,2)$ $=30+25-35=20$
$(2,4)$	$w(2,1)+w(1,4)-w(2,4)$ $=25+20-10=35$
$(4,2)$	$w(4,1)+w(1,2)-w(4,2)$ $=20+25-10=35$
$(3,4)$	$w(3,1)+w(1,4)-w(3,4)$ $=30+20-15=35$
$(4,3)$	$w(4,1)+w(1,3)-w(4,3)$ $=20+30-15=35$

Savings Heuristic

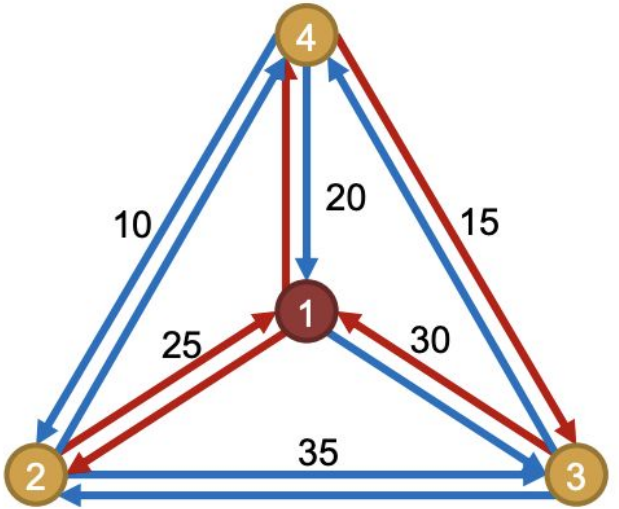


- Current cost:

25 + 25 + 30 + 30 + 20 + 20 = 150

(i,j)	Savings $s(i,j)$
$(2,3)$	$w(2,1)+w(1,3)-w(2,3)$ $=25+30-35=20$
$(3,2)$	$w(3,1)+w(1,2)-w(3,2)$ $=30+25-35=20$
$(2,4)$	$w(2,1)+w(1,4)-w(2,4)$ $=25+20-10=35$
$(4,2)$	$w(4,1)+w(1,2)-w(4,2)$ $=20+25-10=35$
$(3,4)$	$w(3,1)+w(1,4)-w(3,4)$ $=30+20-15=35$
$(4,3)$	$w(4,1)+w(1,3)-w(4,3)$ $=20+30-15=35$

Savings Heuristic

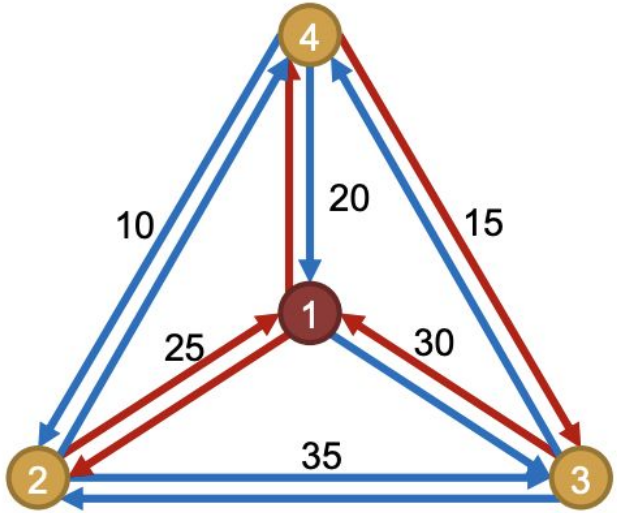


- Current cost:

25 + 25 + 20 + 15 + 30 = 115

(i,j)	Savings $s(i,j)$
$(2,3)$	$w(2,1)+w(1,3)-w(2,3)$ $=25+30-35=20$
$(3,2)$	$w(3,1)+w(1,2)-w(3,2)$ $=30+25-35=20$
$(2,4)$	$w(2,1)+w(1,4)-w(2,4)$ $=25+20-10=35$
$(4,2)$	$w(4,1)+w(1,2)-w(4,2)$ $=20+25-10=35$
$(3,4)$	$w(3,1)+w(1,4)-w(3,4)$ $=30+20-15=35$
$(4,3)$	$w(4,1)+w(1,3)-w(4,3)$ $=20+30-15=35$

Savings Heuristic

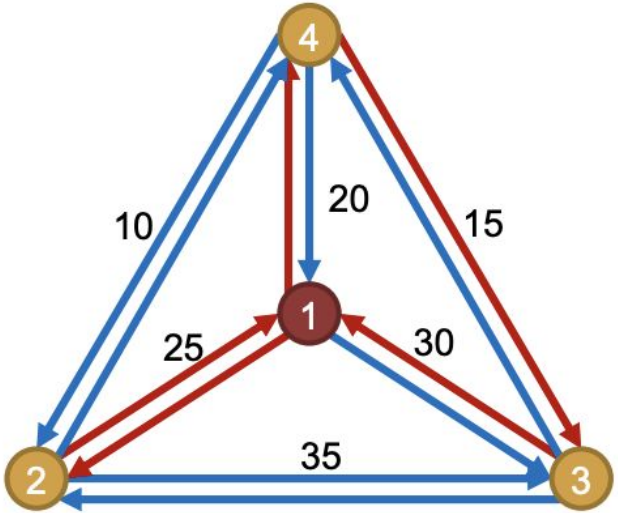


- Current cost:

25 + 25 + 20 + 15 + 30 = 115

(i,j)	Savings $s(i,j)$
$(2,3)$	$w(2,1)+w(1,3)-w(2,3)$ $=25+30-35=20$
$(3,2)$	$w(3,1)+w(1,2)-w(3,2)$ $=30+25-35=20$
$(2,4)$	$w(2,1)+w(1,4)-w(2,4)$ $=25+20-10=35$
$(4,2)$	$w(4,1)+w(1,2)-w(4,2)$ $=20+25-10=35$
$(3,4)$	$w(3,1)+w(1,4)-w(3,4)$ $=30+20-15=35$
$(4,3)$	$w(4,1)+w(1,3)-w(4,3)$ $=20+30-15=35$

Savings Heuristic

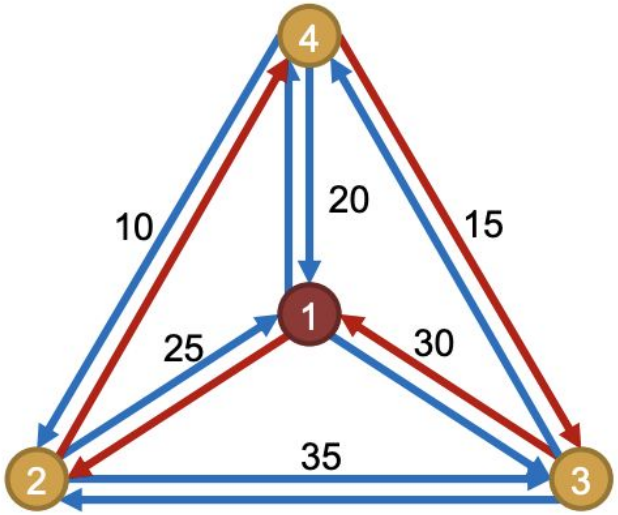


- Current cost:

25 + 25 + 20 + 15 + 30 = 115

(i,j)	Savings $s(i,j)$
$(2,3)$	$w(2,1)+w(1,3)-w(2,3)$ $=25+30-35=20$
$(3,2)$	$w(3,1)+w(1,2)-w(3,2)$ $=30+25-35=20$
$(2,4)$	$w(2,1)+w(1,4)-w(2,4)$ $=25+20-10=35$
$(4,2)$	$w(4,1)+w(1,2)-w(4,2)$ $=20+25-10=35$
$(3,4)$	$w(3,1)+w(1,4)-w(3,4)$ $=30+20-15=35$
$(4,3)$	$w(4,1)+w(1,3)-w(4,3)$ $=20+30-15=35$

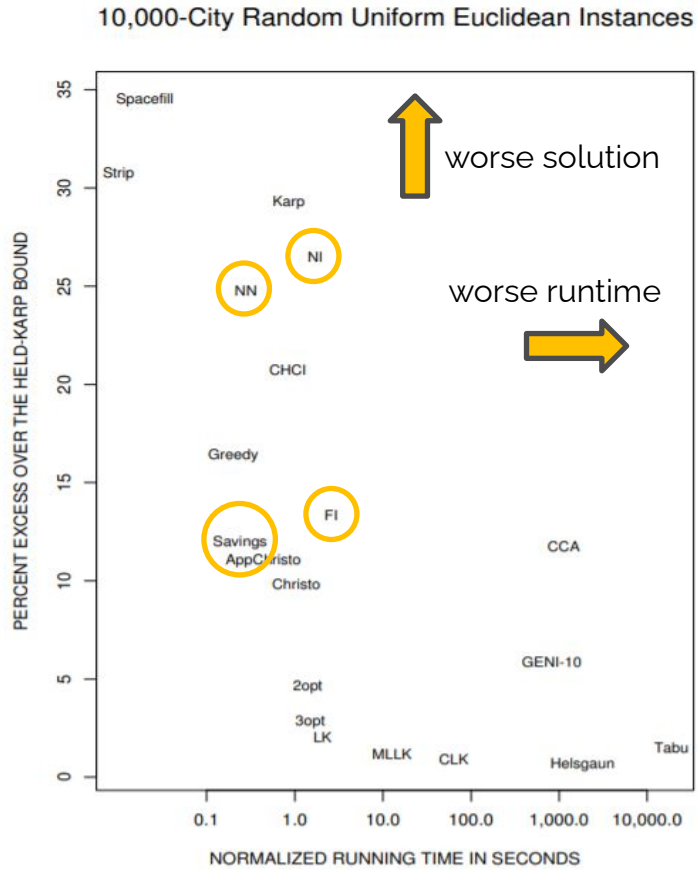
Savings Heuristic

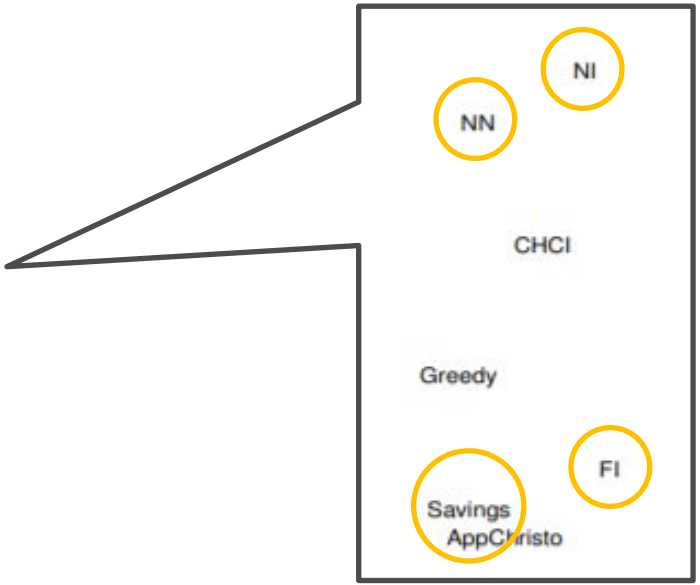


- Current cost:

25 + 10 + 15 + 30 = 80

(i, j)	Savings $s(i, j)$
$(2, 3)$	$w(2,1) + w(1,3) - w(2,3)$ $= 25 + 30 - 35 = 20$
$(3, 2)$	$w(3,1) + w(1,2) - w(3,2)$ $= 30 + 25 - 35 = 20$
$(2, 4)$	$w(2,1) + w(1,4) - w(2,4)$ $= 25 + 20 - 10 = 35$
$(4, 2)$	$w(4,1) + w(1,2) - w(4,2)$ $= 20 + 25 - 10 = 35$
$(3, 4)$	$w(3,1) + w(1,4) - w(3,4)$ $= 30 + 20 - 15 = 35$
$(4, 3)$	$w(4,1) + w(1,3) - w(4,3)$ $= 20 + 30 - 15 = 35$





https://pubsonline.informs.org/doi/abs/10.1287/ijoc.4.4.387
http://www.atgc-montpellier.fr/permutmatrix/manual/SeriationPPI.htm

Vehicle Routing Problem

- Actually, the Savings heuristic was created to solve a generalization of the TSP:
- The **Vehicle Routing Problem (VRP)** also takes place in a weighted, complete graph
- Instead of one salesman, we have a fleet of vehicles which are all parked at a central vertex (the **depot**)
  - May or may not be a limit on the number of vehicles
- **Goal**: find routes starting and ending at the depot for each vehicle with minimum total weight so that each vertex is visited once by some vehicle

Constrained VRP

- In real life: why use a fleet of vehicles when you could have one vehicle that travels all the routes?
- There may be additional constraints for vehicles, e.g.:
  - Maximum distance a vehicle can travel
  - Carrying capacity of a vehicle, where each node has some volume to be delivered

Savings Heuristic for VRP

- Let x denote the depot
- Start with $n-1$ subtours: $x \to v \to x$ for all $v \in V - x$
- For each edge (i, j), where $i, j \in V - x$, compute its **savings** $s(i, j)$
  - $s(i, j) = w(i, x) + w(x, j) - w(i, j)$
- Sort edges in decreasing order of savings
- Repeat until only one tour remains or we reach negative savings:
- Let (i, j) be the next edge in sorted order
- If edges (i, x) and (x, j) are in our subtours, and i, j are not already in the same tour: replace (i, x) and (x, j) by (i, j)...
  - ...unless it would violate our constraints

Solving TSP with OR-Tools

- OR-Tools comes with a **routing solver** that can solve the TSP and VRP with much more complex constraints!
  - Pickups and drop-offs, time windows, penalties...
- The guide is pretty good: https://developers.google.com/optimization/routing
- Comes with many heuristics including NN, Savings, etc...
  - By default, solver automatically chooses a heuristic to use based on the problem at hand
- Note: the routing solver is optimized for getting a "good enough" solution to constrained problems, not exact solving huge TSPs

Scaling and Shifting

- **Warning:** the OR-Tools routing solver may not work correctly with fractional/negative edge weights
  - Even worse, it might not throw an error!
- Can fix negative weights by **shifting**:
  - Add large constant K to all weights to make them positive
  - Preserves TSP structure since all tours increase by $K \cdot n$
  - May not necessarily preserve VRP structure ¯_(ツ)_/¯
- Can fix fractional weights by **scaling**:
  - Multiply all weights by a large constant M to make them integers (or minimize rounding error)
  - If no rounding, preserves TSP and VRP structure

The OR-Tools TSP Solver doesn't always produce an optimal solution.

How well does it do in practice ?

Let's test it on instances from the National TSP Collection, a set of real-world instances ranging in size from 29 to 71,000+ nodes.

Benchmarking the TSP

Country	# Cities	Output Cost	Optimal Cost	Percent Error	*Runtime (s)
W. Sahara	29	27749	27603	0.53%	0.0320
Djibouti	38	7078	6656	6.3%	0.0657
Qatar	194	10064	9352	7.6%	2.61
Uruguay	734	83476	79114	5.5%	37.9
Zimbabwe	929	101100	95345	6.0%	91.4
Oman	1979	92250	86891	6.2%	668

*Running on a Dell XPS laptop with 16GB of RAM, in a Jupyter notebook.