Optimization Theory and Methods

2025 Autumn

同济经管 TONGJI SEM

魏可佶 kejiwei@tongji.edu.cn
https://kejiwei.github.io/

CAMEA 中国高质量MBA教育认证 AACSB ACCREDITED EQUIS ACCREDITED

- Arc Covering Problem (Chinese Postman Problem – CPP)
  - Eulerian Cycle
  - CPP Algorithm
- Node Covering Problem (Traveling Salesperson Problem – TSP)
  - Insertion Heuristics
  - Improvement Heuristics
  - MST Heuristic
  - Christofides Heuristic

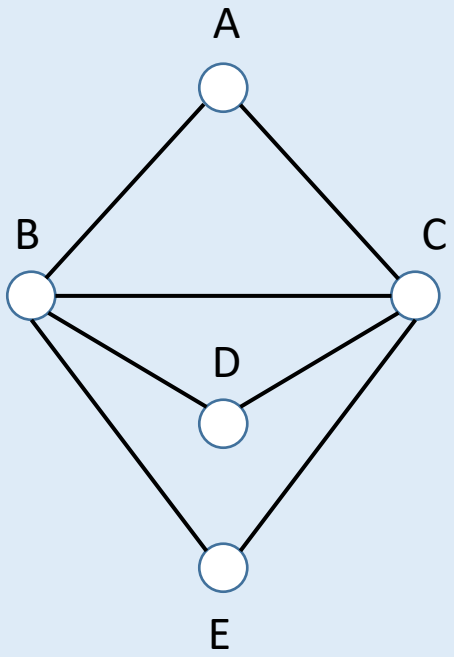
↳ Chinese Postman Problem (CPP)

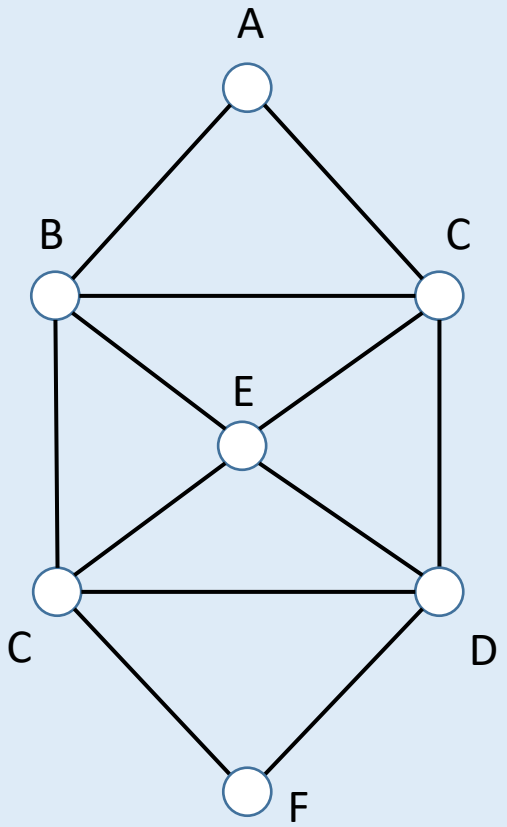
- Postman travels from door to door covering each street at least once to deliver mail.
- Find a least cost walk that starts from a node, traverses each arc at least once, and returns to the starting node.
- Many applications and variations
  - Snow plowing
  - Street sweeping
  - Mail delivery
  - CPP with time windows
  - Rural CPP
  - etc.

↳ Background: Eulerian Cycle

- Defined for an undirected network.
- **Closed Walk:** A walk that ends at the same node where it starts.
- **Eulerian Cycle**: A closed walk that passes along each arc exactly once.
- **Degree of a node**: Number of arcs incident to the node.
- Theorem: An undirected network has an Eulerian cycle if and only if the network is connected and every node has an even degree.
  - Proof: The degree of a node is twice the number of times it appears on the walk (except for the starting and ending node).

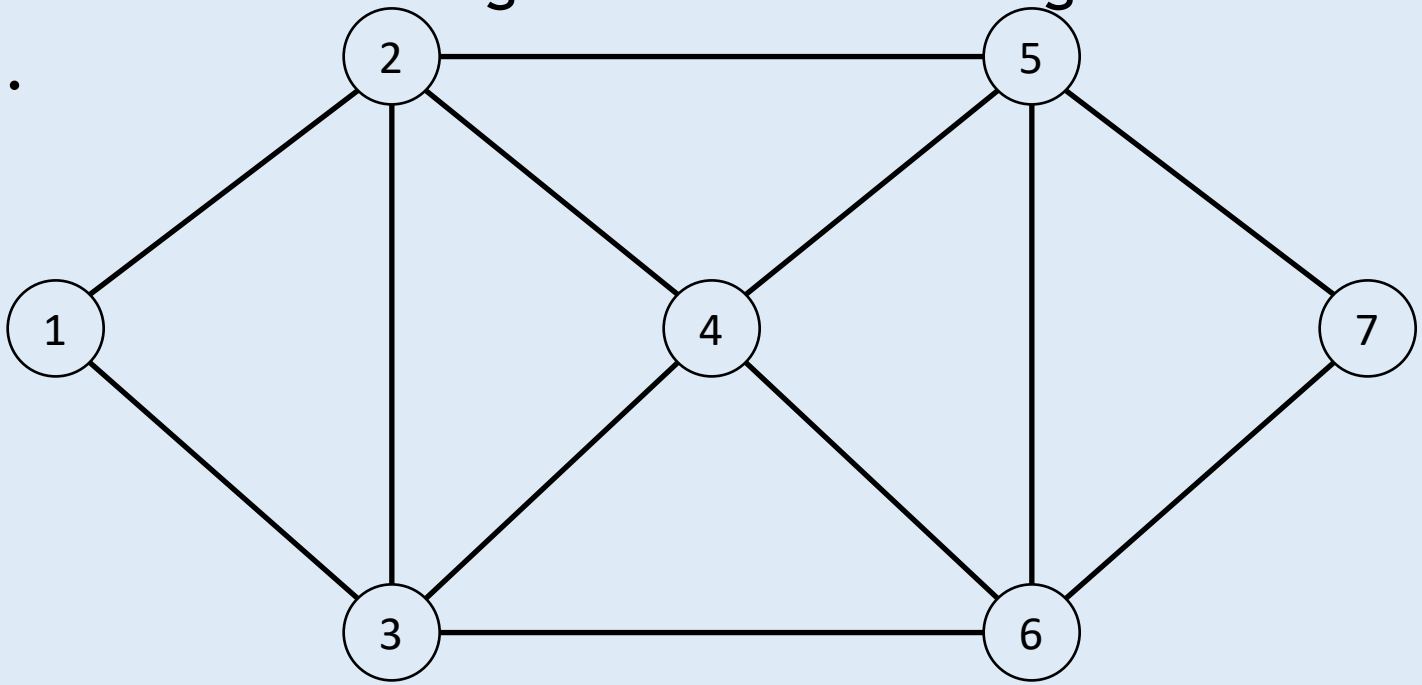
↳ Example Networks with Eulerian Cycle





- An important fact: The number of odd-degree nodes in any undirected network is always even.
  - Proof:
    (1) Each arc contributes 2 degrees, one to each end node. So total number of degrees across all nodes is even.
    (2) Total number of degrees contributed by even-degree nodes is obviously even (since sum of even numbers is even).
    (3) So total number of degrees contributed by odd-degree nodes is also even.
    (4) If number of odd-degree nodes was odd then the sum of their degrees cannot be even.
    (5) So number of odd-degree nodes must be even.

- If all nodes have even degree, then an Eulerian Cycle exists.
- For such a network, Eulerian Cycle is the optimal solution to the Arc Covering Problem (CPP) (Why?)
- Finding an Eulerian Cycle: Start from any node and continue along previously uncovered arcs ensuring that the remaining network is not disconnected.



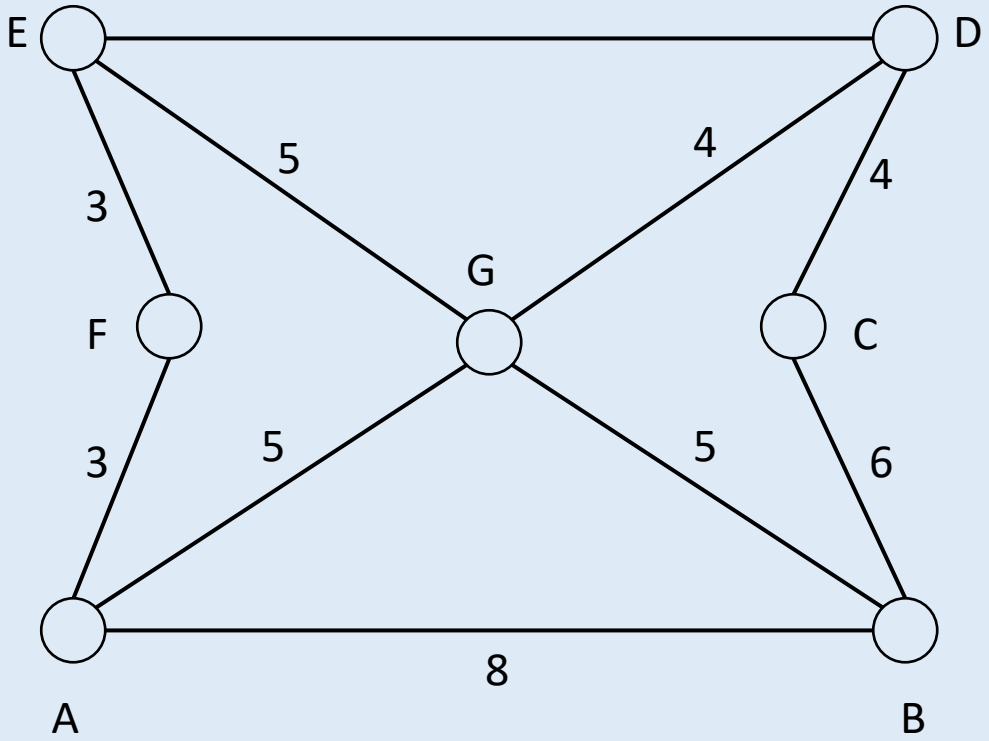
- Basic idea: Given a network $G(N, A)$, add dummy arcs between odd-degree nodes, so that all nodes become even degree nodes. Ensure minimum total length of additional arcs. Then find an Eulerian Cycle in this modified network.
- Details:

(1) Find all odd-degree nodes. Let m be the number of such nodes. Note: We have already shown that m is an even number.

(2) Find the shortest paths between each pair of odd-degree nodes. (As we learnt in class, this can be done in polynomial time.)

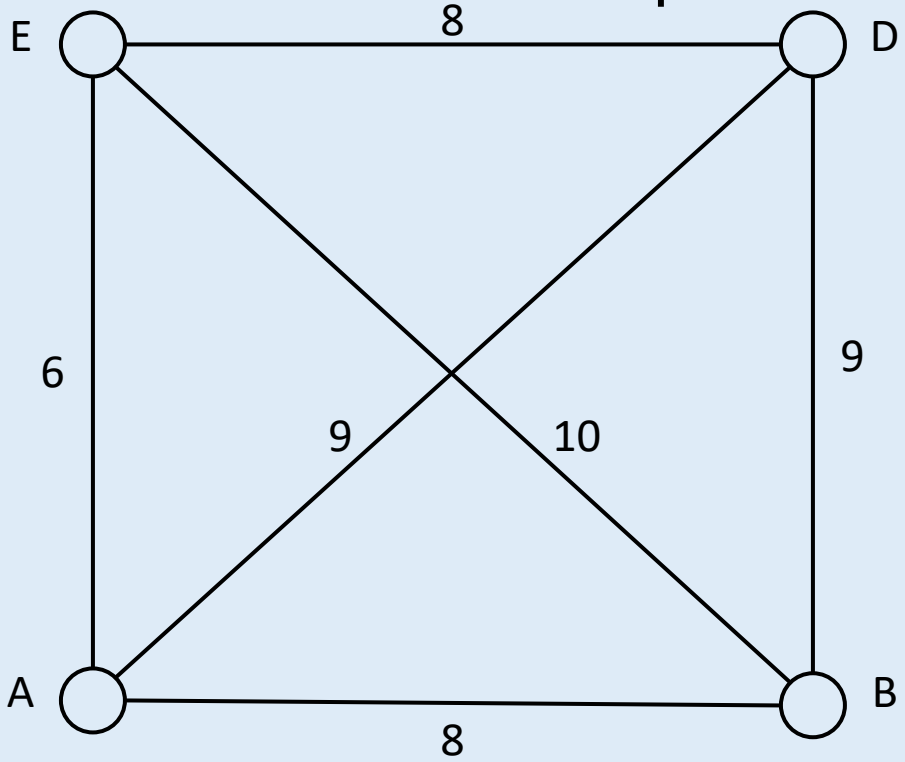
(3) Using the shortest path costs, find the minimum cost pair-wise matching of odd-degree nodes. (This can be done in polynomial time too!) Let it be denoted by arcs A'.

(4) Define augmented network $G'(N, A \cup A')$ and find an Eulerian Cycle on G'. This is the solution to CPP.



- There are four odd-degree nodes: A, B, D, and E.
- Find shortest paths between all 6 combinations: AB, AD, AE, BD, BE, and DE.

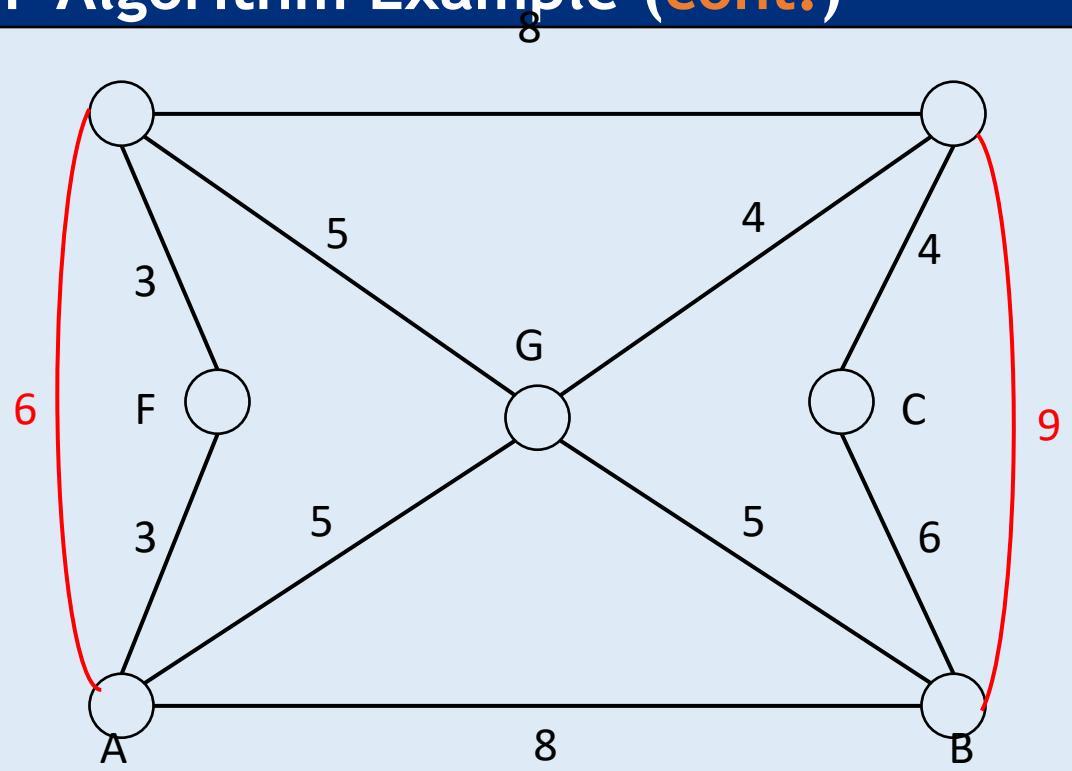
- Find minimum cost pair-wise matching of odd-degree nodes.



- 3 Pair-wise Matches:
  - $\{AB, ED\}$: Cost = 16
  - $\{AD, EB\}$: Cost = 19
  - $\{AE, BD\}$: Cost = 15
- Select $\{AE, BD\}$ since it is the optimal matching.

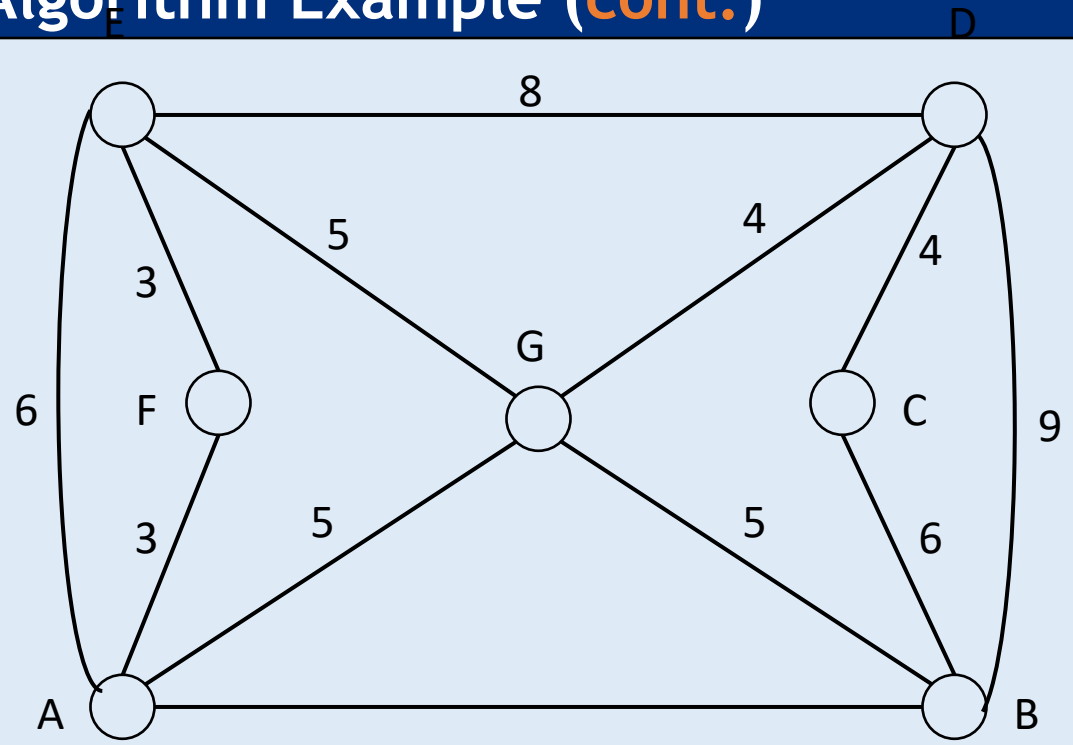
- Key Observation: No two shortest paths in an optimal matching can have common arcs.
- So manual matching often works well.
- Local neighborhood searches are often optimal or near-optimal.

↳ CPP Algorithm Example (cont.)

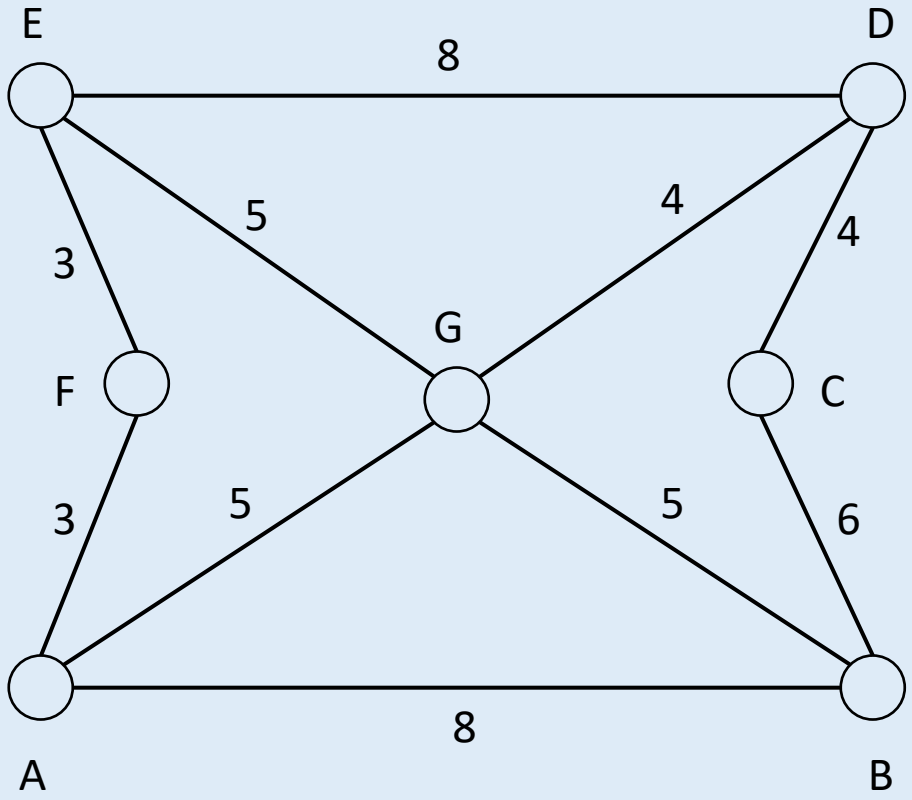


- Now find a Eulerian Cycle on this augmented network.

↳ CPP Algorithm Example (cont.)



- Optimal CPP tour length for original network
  - = Eulerian cycle length for the augmented network
  - = Sum of lengths of all arcs of the augmented network
  - = 8+9+4+5+3+3+6+8+4+6+5+5 = 66.



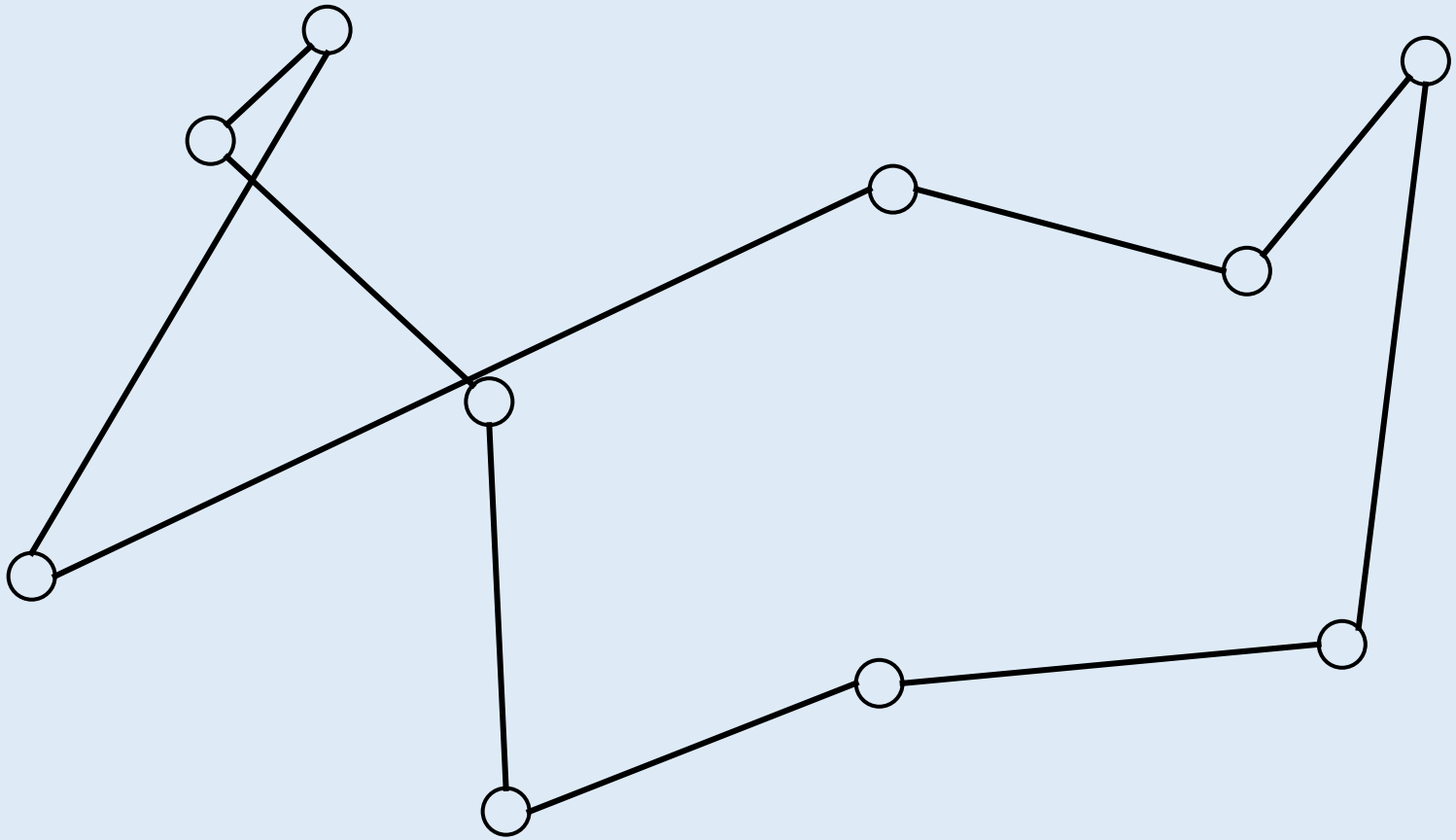
- Optimal CPP Tour: A-B-G-D-G-E-F-A-F-E-D-C-B-G-A

↳ Traveling Salesperson Problem (TSP)

- A salesperson travels from city-to-city.
- Find a least cost tour that starts from a city (node), visits every other city (node) exactly once and returns to the starting city (node).
- Very common problem in many fields.
  - Optimizing vehicle routing.
  - Determining the landing sequence of aircraft on a runway.
  - Machine scheduling in a machine shop.
  - Genome sequencing.
  - Imaging of celestial objects.
  - Testing in semi-conductor manufacturing.
  - Optimizing the design of a fiber optics network, etc.

- It is an "*NP-Hard*" problem. No known method to solve it exactly (to optimality) and efficiently (polynomial run times).
- If we find a polynomial time algorithm to solve this problem, then we would have automatically found a polynomial time algorithm to solve several other known problems that are equally *hard* (set of these problems is called "NP-Complete").
- Two solution ideas:
  - Use simple rules to gradually build a tour – Construction Heuristic
  - Use simple rules to gradually improve an existing tour – Improvement Heuristic
- Construction Heuristics: Start from nothing, or start from a tour on a small subset of nodes.

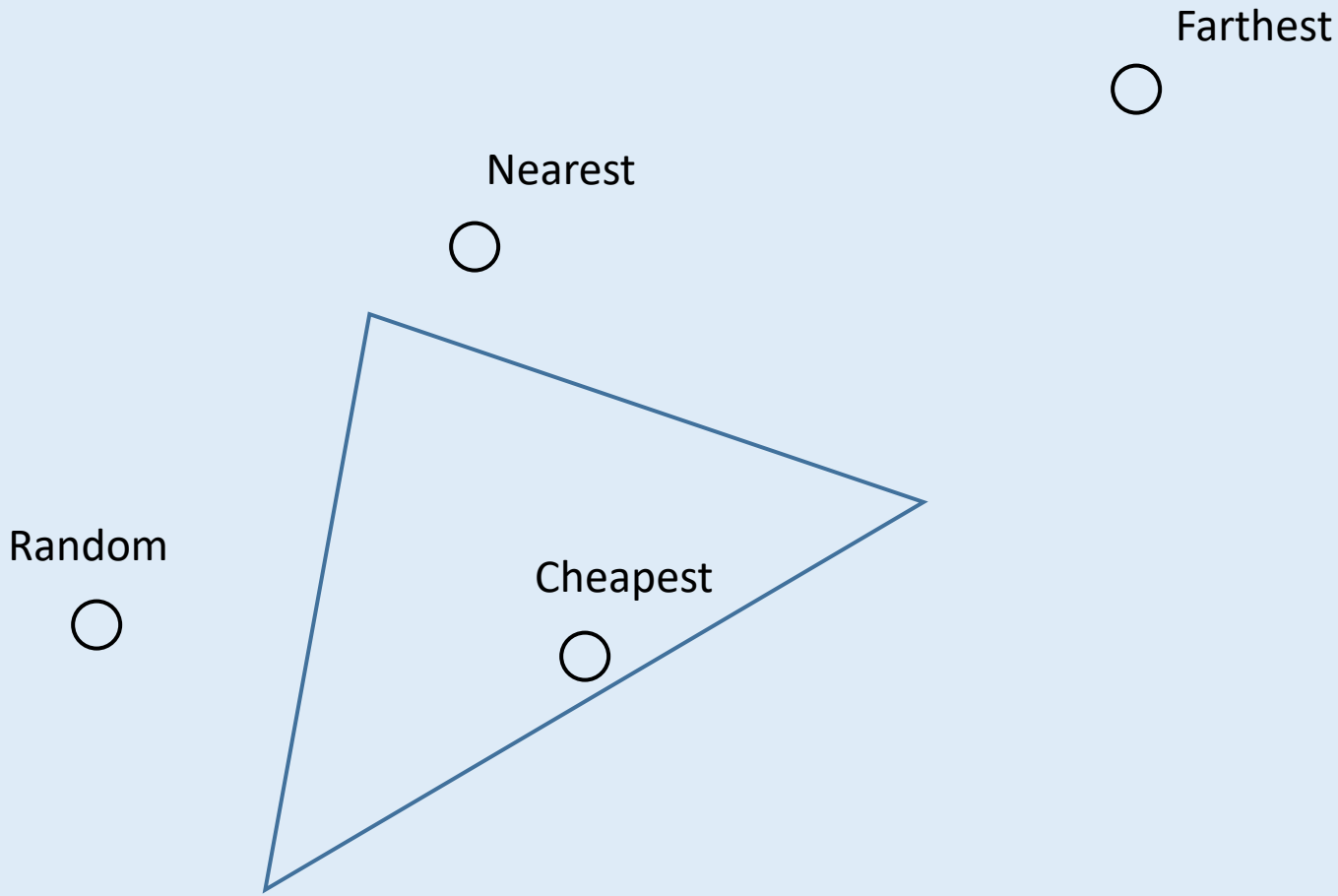
- Construct Starting from Nothing
- Nearest Unvisited Neighbor Heuristic



↳ Construction Heuristics (cont.)

- **Insertion Heuristics**: Construct Starting From a Smaller Tour on a Subset of Nodes.
- Initialization: Create a tour on a small subset of nodes.
  - E.g. 3 node tour.
  - E.g. Outer nodes (nodes on the convex hull).
- Iterations: Grow the tour progressively by inserting one node at a time.
- Termination: Stop when the tour covers all the nodes.

- Insertion Strategies
- Nearest insertion
- Farthest insertion
- Cheapest insertion
- Random insertion



↳ Euclidean TSP: Performance Bounds

- In many problems, distances are Euclidean.
- Performance of the heuristics is defined in terms of the worst-case ratio of the length of the heuristic solution tour to the length of the true optimal tour.
- E.g.:

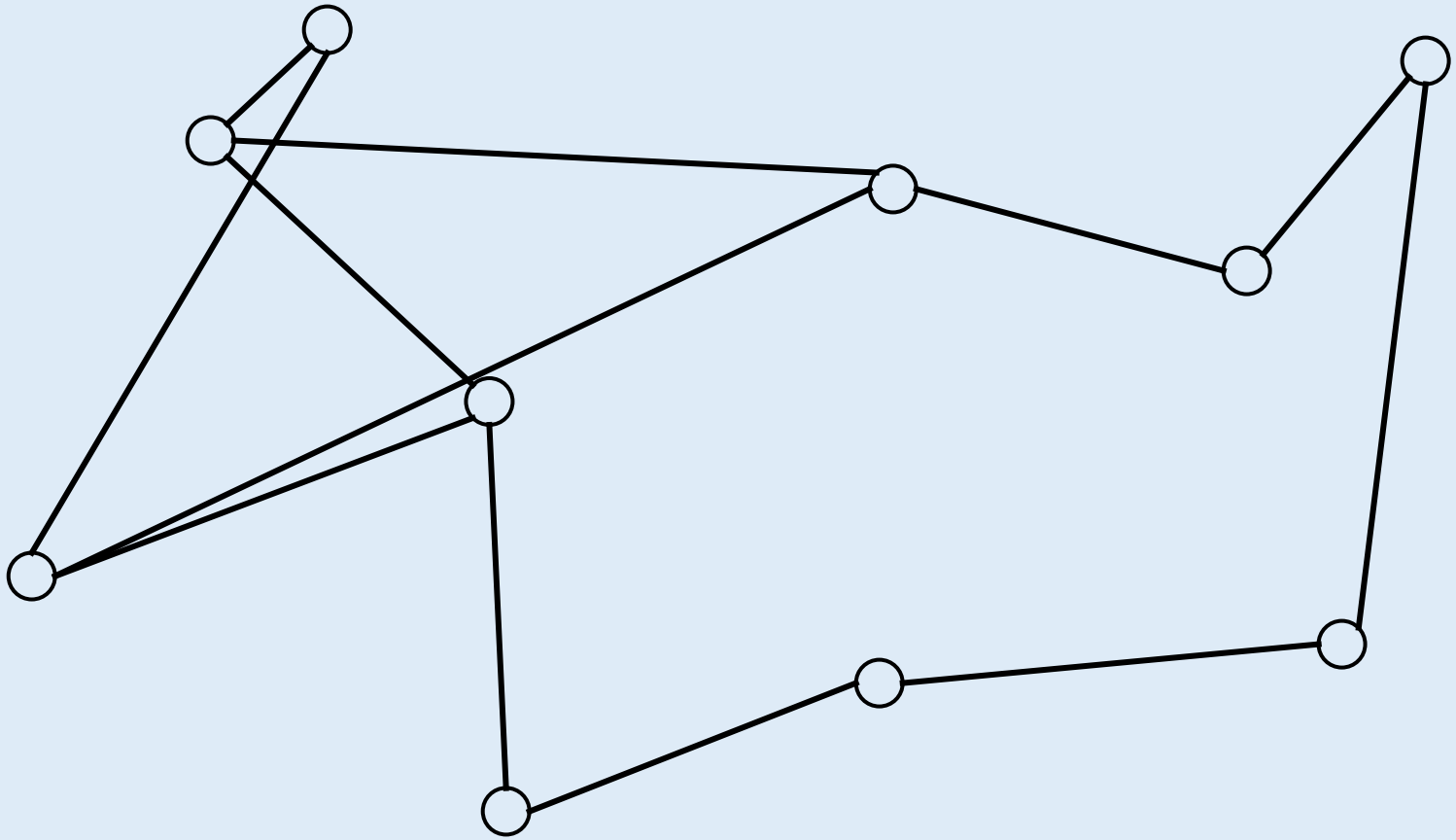
Heuristic	Ratio Bound
Nearest Unvisited Neighbor Heuristic	$\frac{1}{2}\lceil \log_2 n \rceil + \frac{1}{2}$
Nearest Insertion Heuristic	2
Farthest Insertion Heuristic	Unknown
Cheapest Insertion Heuristic	2
Random Insertion Heuristic	$\lceil \log_2 n \rceil + 1$

where n = number of nodes.

- $Nh(T)$: Neighborhood of tour T.
  - $Nh(T)$ can be defined in many different ways.
- For example,
  - All tours obtained by deleting 2 arcs and adding 2 new arcs to T.
  - All tours obtained by deleting 3 arcs and adding 3 new arcs to T.
  - All tours obtained by switching the positions of 2 points in T, etc.
- General structure of an improvement heuristic is as follows:
  - Start with a tour T (could be obtained by nearest unvisited neighbor method, or randomization, or through any other way).
  - While there is a tour T' in $Nh(T)$ such that length of T' is less than that of T, then replace T with T'.
  - Stop when no such T' can be found.

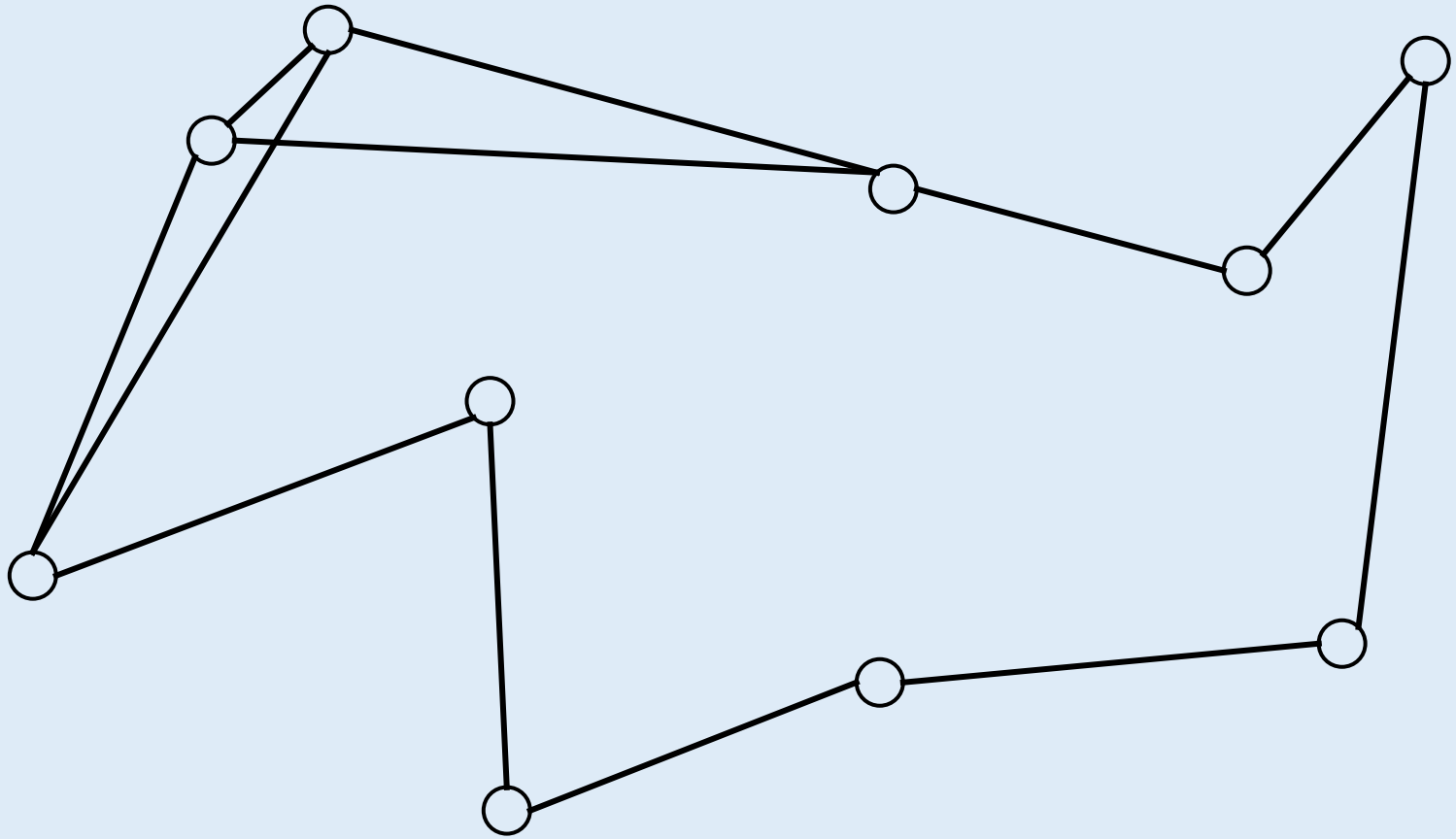
↳ Improvement Heuristics : Two Exchange

- Let the neighborhood consist of all tours obtained by deleting 2 arcs and adding 2 new arcs.

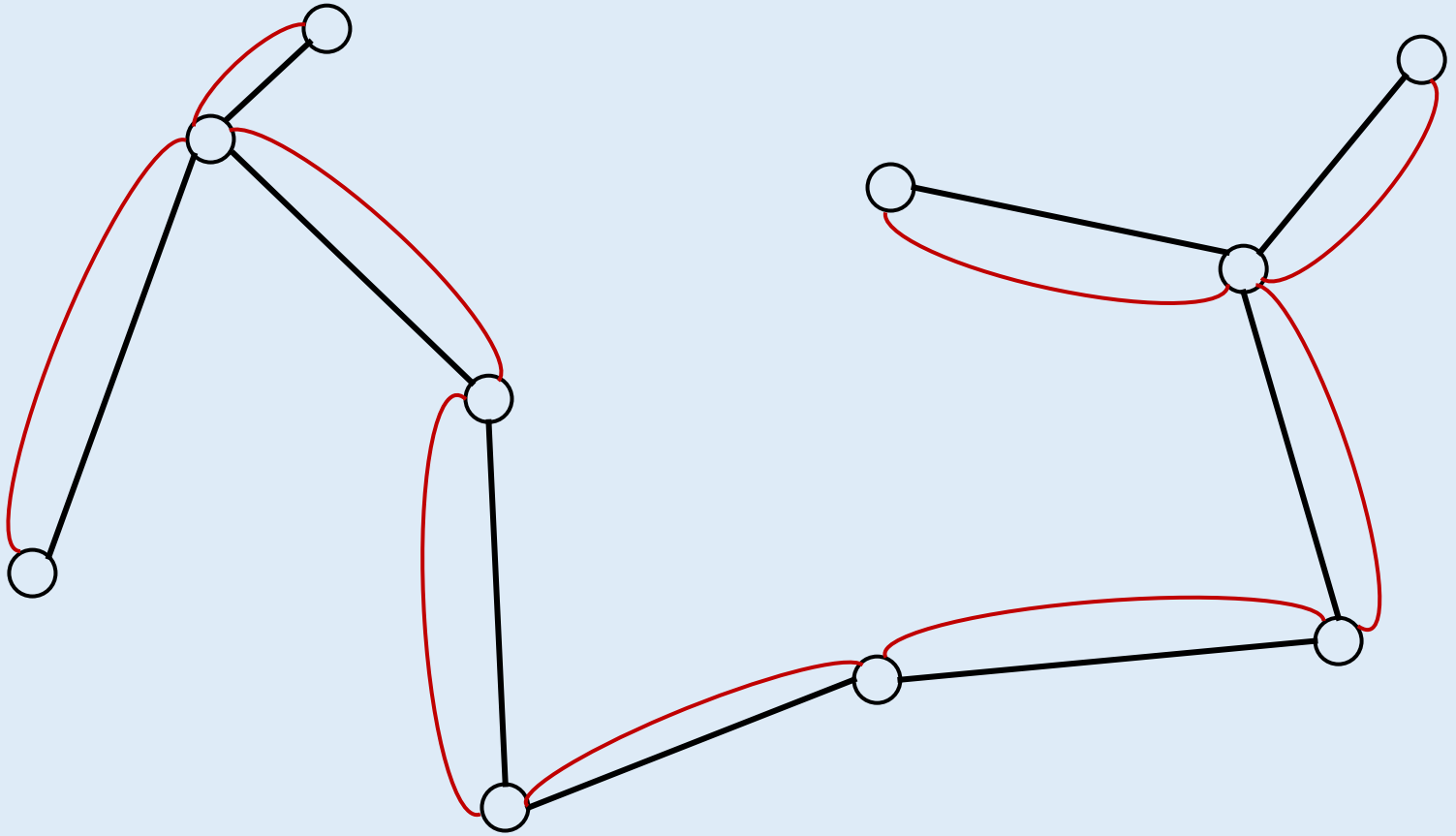


↳ Improvement Heuristics : Two Exchange (cont.)

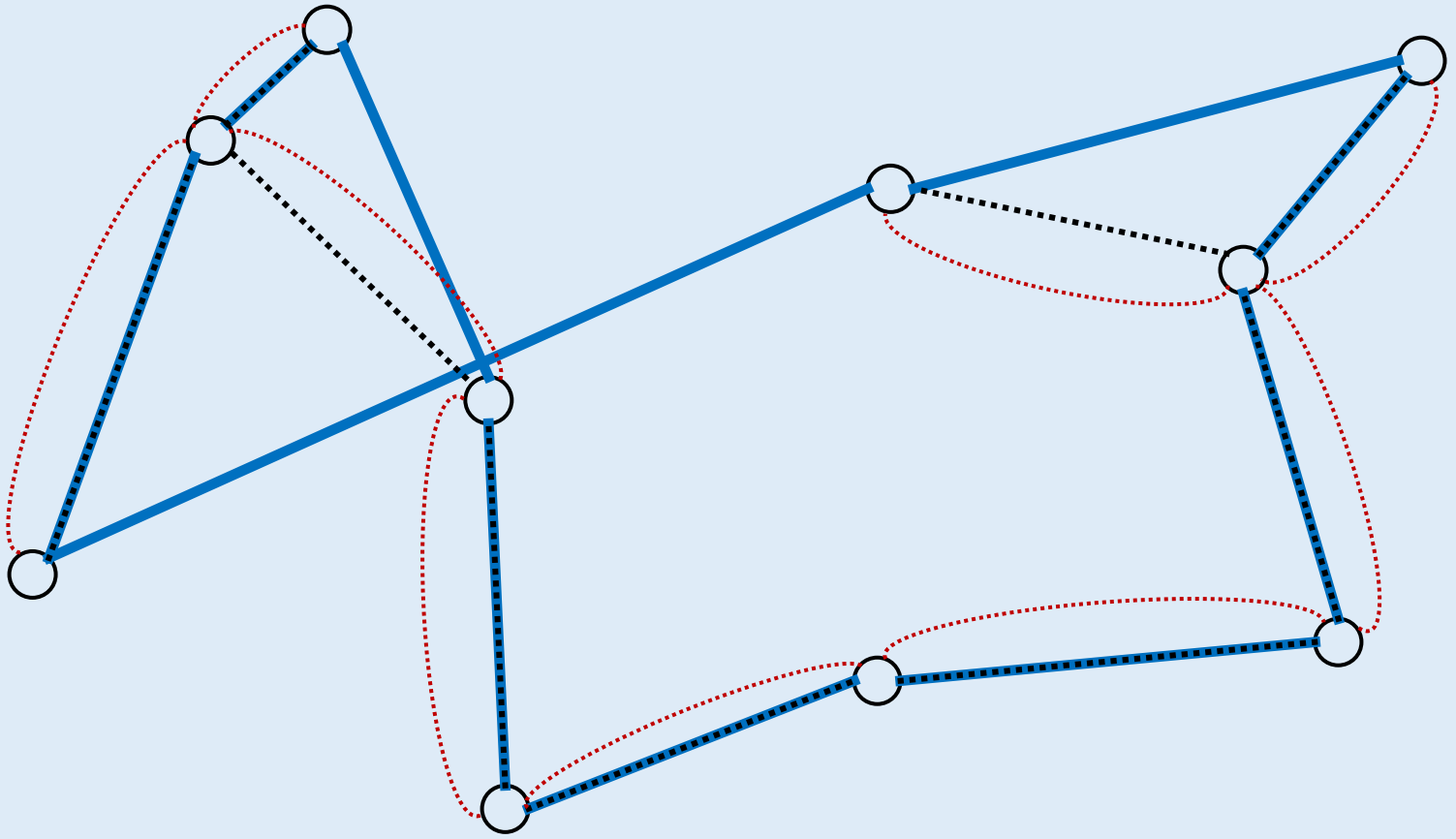
- After 2 iterations, convergence is reached in this example.



1) First, find the minimum spanning tree (MST).
2) Next, merge with a second copy of the MST.



3) Improve by skipping points already visited.



Worst-Case Performance can be calculated as follows:

- Removal of one arc of any TSP tour gives a spanning tree.
- Length of Optimal MST

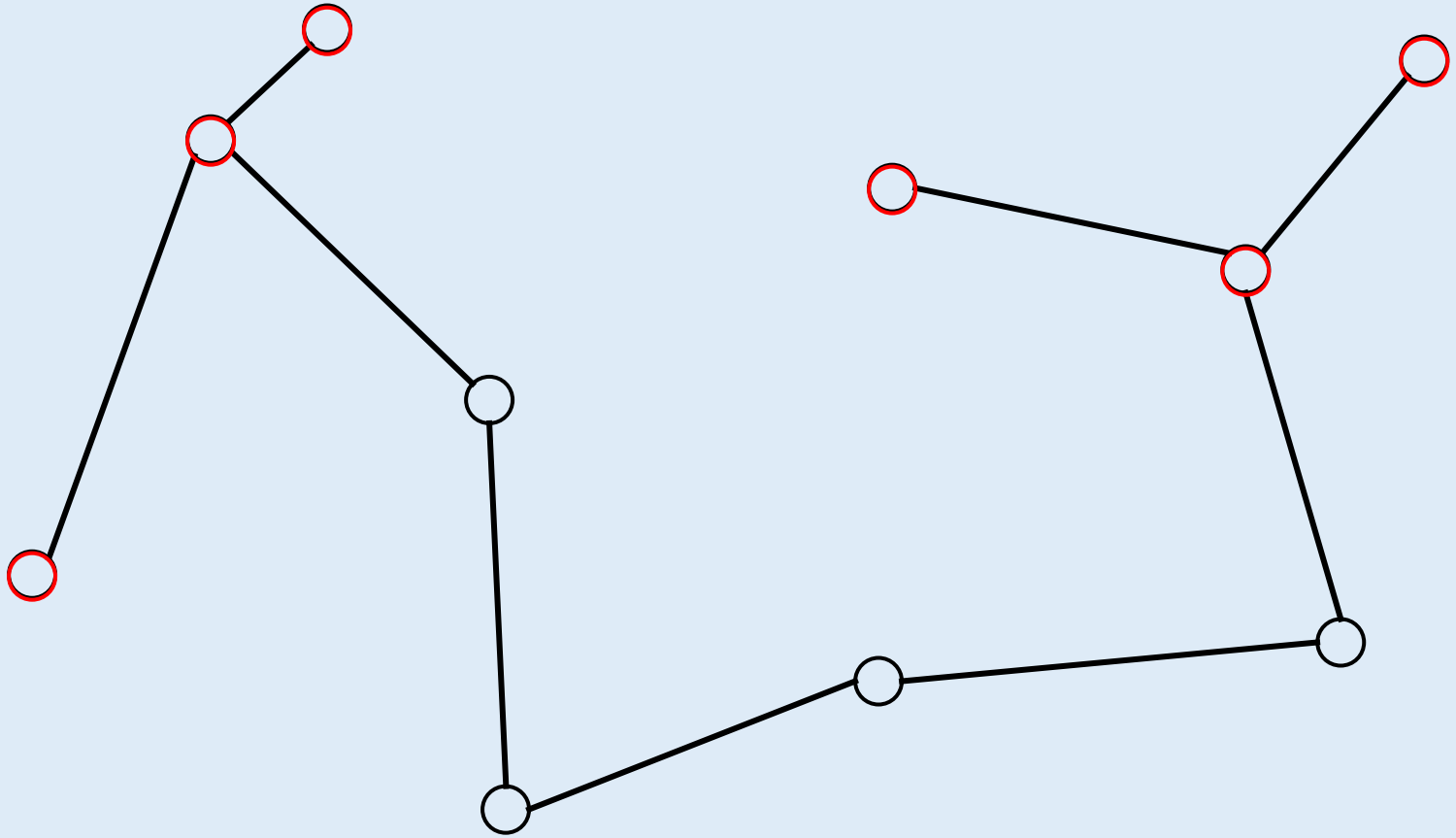
  ≤ Length of Optimal TSP – Length of Any Arc of the Optimal TSP

  < Length of Optimal TSP.
- So, length of tour obtained by MST heuristic

  ≤ 2*Length of Optimal MST (Why?)

  < 2*Length of Optimal TSP
- Thus, worst-case bound for the MST heuristic for TSP is 2.
- This can be improved using Christofides Heuristic.

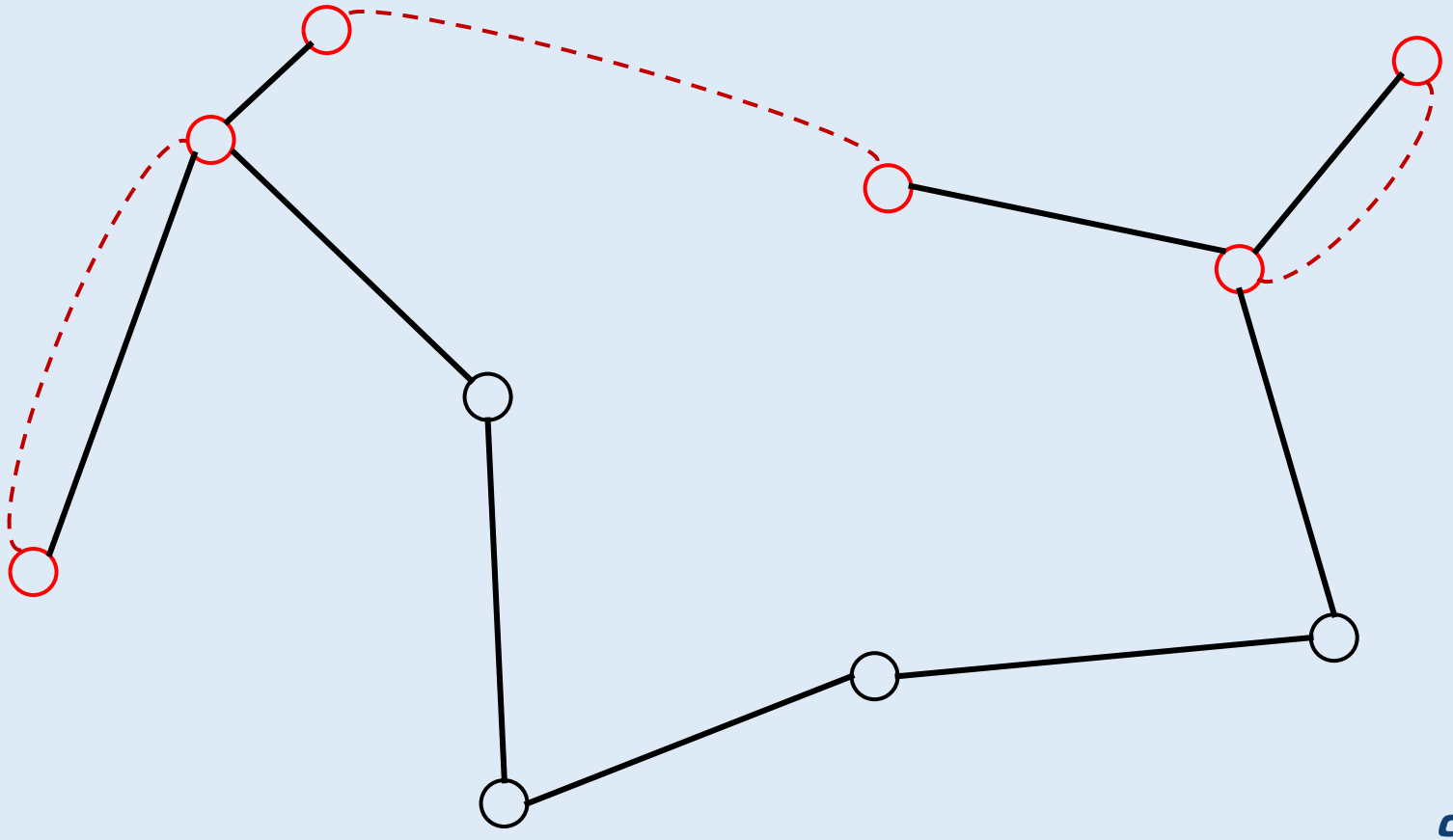
1) First, find the minimum spanning tree. Denote it by MST.



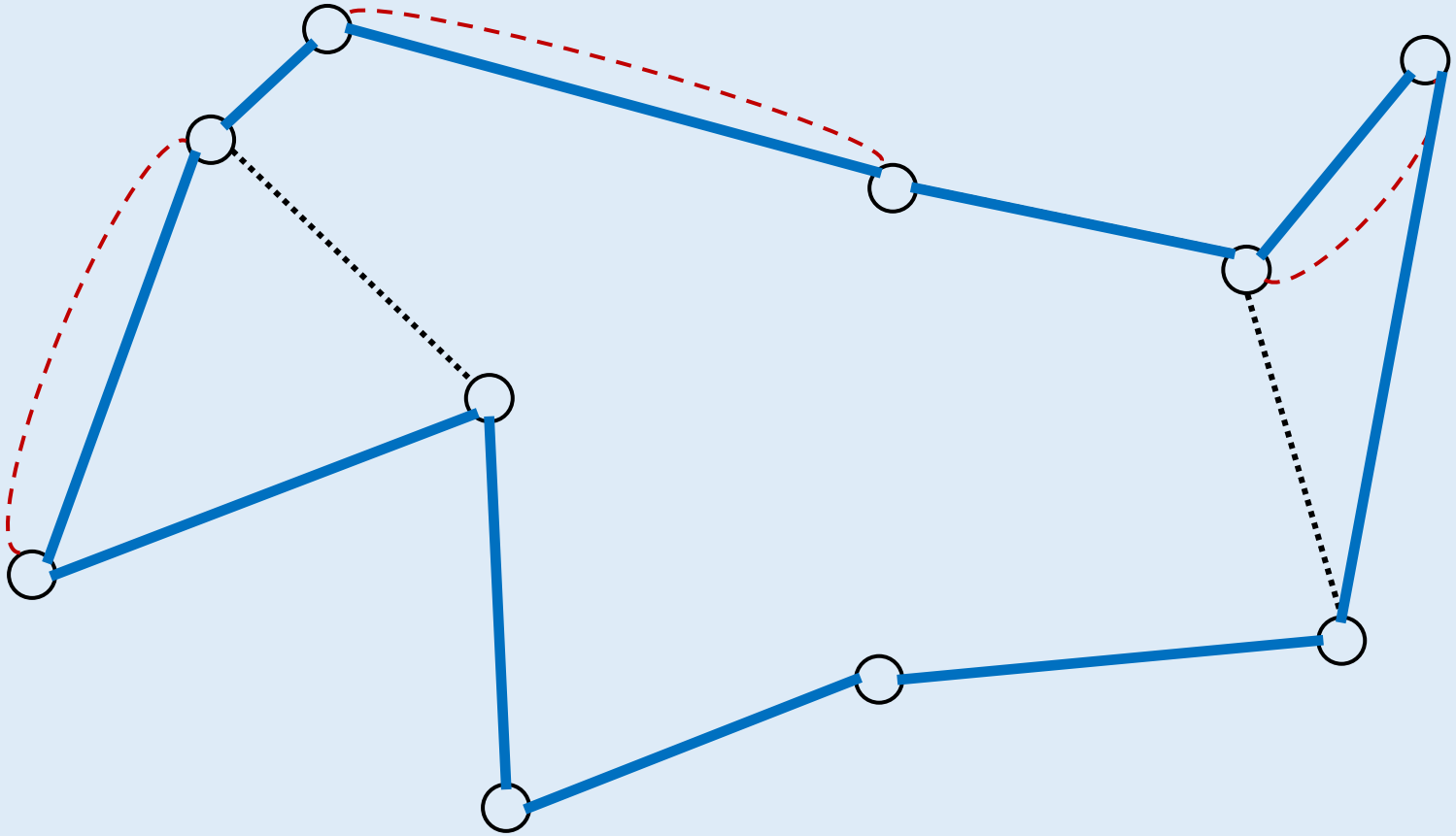
↳ Christofides Heuristics (cont.)

2) Find all odd-degree nodes in the MST and find minimum-cost pairwise matching of these nodes. Denote it by M.

3) Merge MST and M



4) Improve solution by skipping points already visited.



↳ Performance of Christofides Heuristics

1) Length of Christofides solution

   ≤ Length of MST + Length of M

2) Length of MST < Length of TSP.

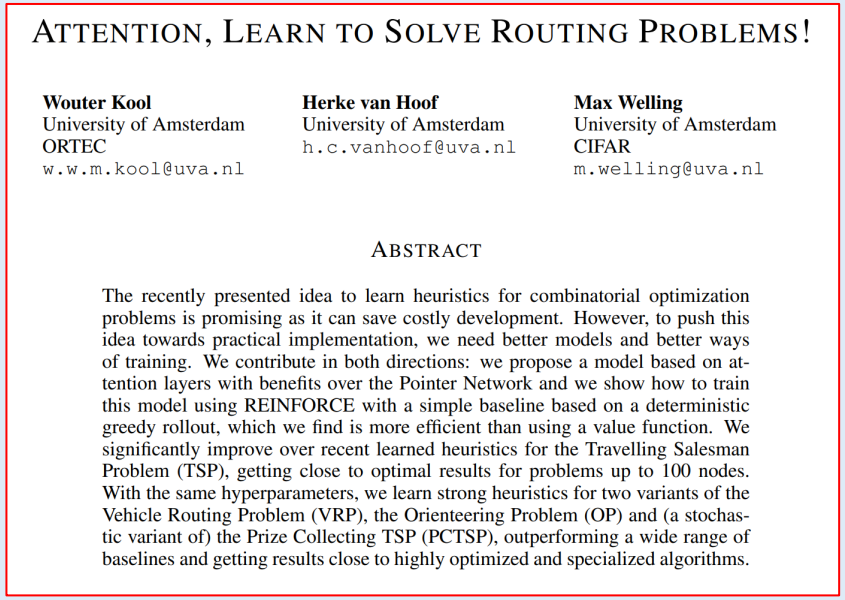
3) Length of M ≤ Length of M' ≤ Length of TSP/2.

   where M'=Minimum cost pairwise matching of odd-degree nodes of MST using only arcs that are part of the TSP solution.

- So, length of Christofides solution < (3/2) Length of TSP.
- Thus, worst-case bound for the Christofides heuristic is 3/2

https://research.googleblog.com/2016/09/the-280-year-old-algorithm-inside.html

Can we approximately solve TSP rapidly?



ATTENTION, LEARN TO SOLVE ROUTING PROBLEMS!

Wouter Kool
University of Amsterdam
ORTEC
w.w.m.kool@uva.nl

Herke van Hoof
University of Amsterdam
h.c.vanhoof@uva.nl

Max Welling
University of Amsterdam
CIFAR
m.welling@uva.nl

ABSTRACT

The recently presented idea to learn heuristics for combinatorial optimization problems is promising as it can save costly development. However, to push this idea towards practical implementation, we need better models and better ways of training. We contribute in both directions: we propose a model based on attention layers with benefits over the Pointer Network and we show how to train this model using REINFORCE with a simple baseline based on a deterministic greedy rollout, which we find is more efficient than using a value function. We significantly improve over recent learned heuristics for the Travelling Salesman Problem (TSP), getting close to optimal results for problems up to 100 nodes. With the same hyperparameters, we learn strong heuristics for two variants of the Vehicle Routing Problem (VRP), the Orienteering Problem (OP) and (a stochastic variant of) the Prize Collecting TSP (PCTSP), outperforming a wide range of baselines and getting results close to highly optimized and specialized algorithms.

- Recently, other combinatorial problems such as TSP, Vehicle Routing Problem (VRP), Orienteering Problem (OP) and Prize Collecting TSP (PCTSP) have been solved successfully using Graph Attention Network models.
- Define a solution as a tour $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ or permutation of nodes, so that $\pi_t \in \{1, \dots, n\}$ where, $\pi_t \neq \pi_{t'} \; \forall t \neq t'$.
- Then a solution for problem instance s is obtained using a greedy stochastic policy: $p(\pi|s) = \Pi_{t=1}^{n} p(\pi_t|s, \pi_{1:t-1})$.
- This policy is easy to implement rapidly in real-time. But how to train it?

Graph Attention Networks

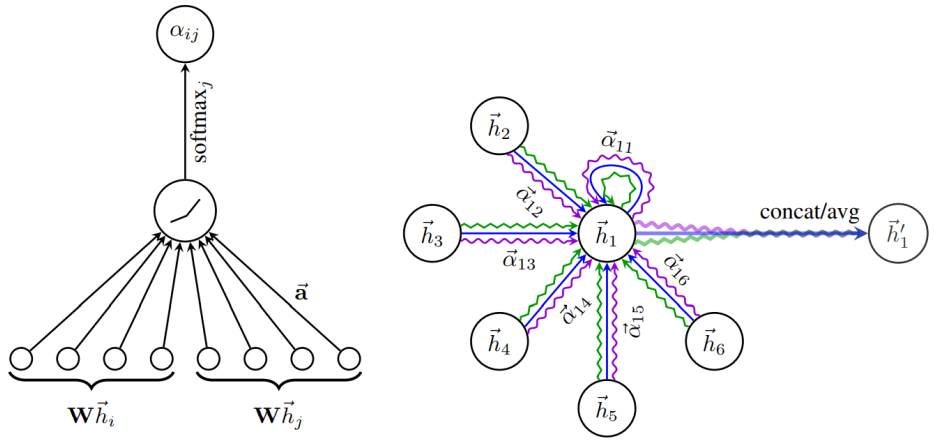


Figure 1: **Left:** The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by our model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

Published as a conference paper at ICLR 2018

GRAPH ATTENTION NETWORKS

Petar Veličković*
Department of Computer Science and Technology
University of Cambridge
petar.velickovic@cst.cam.ac.uk

Guillem Cucurull*
Centre de Visió per Computador, UAB
gcucurull@gmail.com

Arantxa Casanova*
Centre de Visió per Computador, UAB
ar.casanova.8@gmail.com

Adriana Romero
Montréal Institute for Learning Algorithms
adriana.romero.soriano@umontreal.ca

Pietro Liò
Department of Computer Science and Technology
University of Cambridge
pietro.lio@cst.cam.ac.uk

Yoshua Bengio
Montréal Institute for Learning Algorithms
yoshua.umontreal@gmail.com

ABSTRACT

We present graph attention networks (GATs), novel neural network architectures that operate on graph-structured data, leveraging masked self-attentional layers to address the shortcomings of prior methods based on graph convolutions or their approximations. By stacking layers in which nodes are able to attend over their neighborhoods' features, we enable (implicitly) specifying different weights to different nodes in a neighborhood, without requiring any kind of costly matrix operation (such as inversion) or depending on knowing the graph structure upfront. In this way, we address several key challenges of spectral-based graph neural networks simultaneously, and make our model readily applicable to inductive as well as transductive problems. Our GAT models have achieved or matched state-of-the-art results across four established transductive and inductive graph benchmarks: the *Cora*, *Citeseer* and *Pubmed* citation network datasets, as well as a *protein-protein interaction* dataset (wherein test graphs remain unseen during training).

- $\mathbf{h} = \{\vec{h_1}, \vec{h_2}, \ldots, \vec{h_{N_i}}\}$: Node features
- $\mathbf{W}$: Weight matrix
- $\vec{\boldsymbol{a}}$: Single layer feedforward neural network
- σ: Nonlinearity

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

where $\cdot^T$ represents transposition and $\|$ is the concatenation operation.

- Encoder: 3 main layers each with 2 sublayers
  - A multi-head attention layer (MHA)
  - Node-wise fully connected feed-forward (FF) layer
- Decoder:

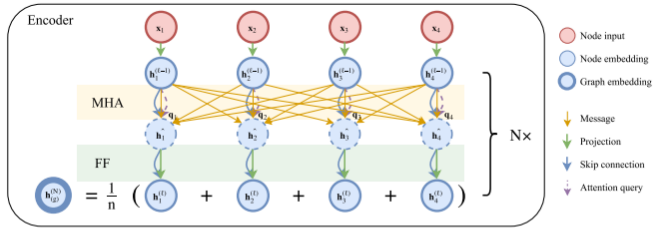


Figure 1: Attention based encoder. Input nodes are embedded and processed by N sequential layers, each consisting of a multi-head attention HA) and node-wise feed-forward (FF) sub- er. The graph embedding is computed as the an of node embeddings. Best viewed in color.

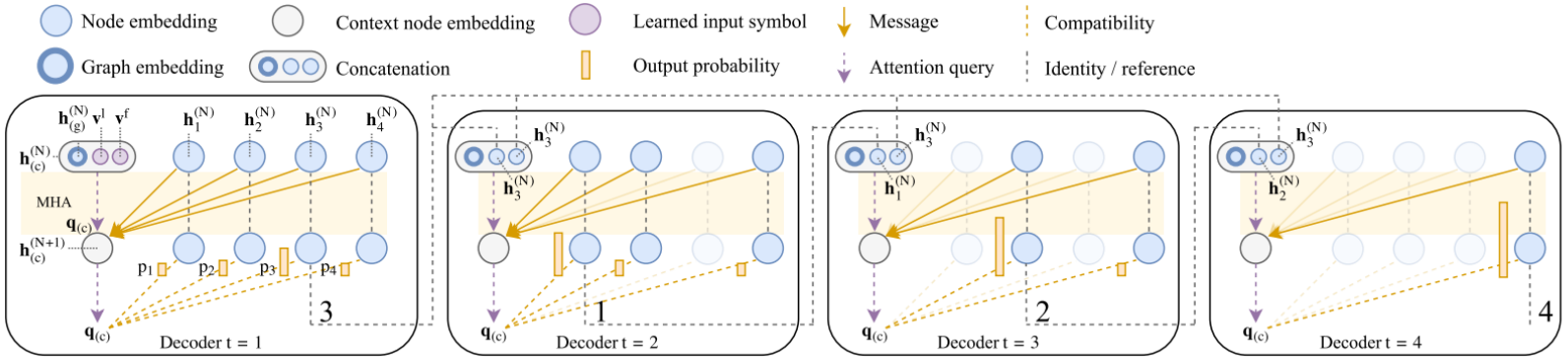


Figure 2: Attention based decoder for the TSP problem. The decoder takes as input the graph embedding and node embeddings. At each time step t, the context consist of the graph embedding and the embeddings of the first and last (previously output) node of the partial tour, where learned placeholders are used if $t = 1$. Nodes that cannot be visited (since they are already visited) are masked. The example shows how a tour $\boldsymbol{\pi} = (3, 1, 2, 4)$ is constructed. Best viewed in color.

- Trained using REINFORCE with greedy rollout baseline (Williams, 1992)

Objective：

Key Concepts：