# Single Vehicle Arc Routing Problems

## 13442 Vehicle Routing and Distribution Planning - Lecture 10

Evelien van der Hurk

Department of Management Engineering, Technical University of Denmark (DTU)
based on previous slides by Roberto Roberti.

Friday 28th October, 2016

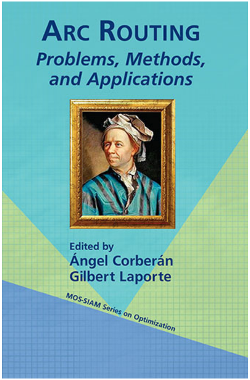
# Outline

# Learning Objectives

- Explain the three main single-vehicle arc routing problems (Covering Tour, Chinese Postman, Rural Postman)
- Solve the Covering Tour problem with the End-Pairing algorithm
- Formulate and solve the Chinese Postman problem to optimality
- Apply the Frederickson algorithm to find heuristic solutions of the Rural Postman problem

# Relevant Literature for This Lecture

- A. Corberan and G. Laporte
  *Arc Routing: Problems, Methods, and Applications*
  MOS-SIAM Series on Optimization, 2015



# Applications of Arc Routing Problems (ARP)

Arc Routing Problems: real-life routing problems where the goal is to traverse a given set of roads

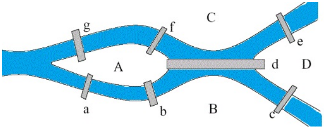
Street Sweeping

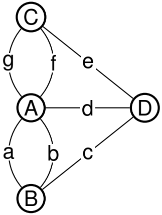

Waste Collection

Mail Delivery

# The Königsberg Bridges Problem I

"In Königsberg (Prussia), there is an island A; the river which surrounds it is divided in two branches, which are crossed by seven bridges *a*, *b*, *c*, *d*, *e*, *f*, and *g*. It was asked whether anyone could arrange a route that crosses each bridge once and only once." [Euler 1736]

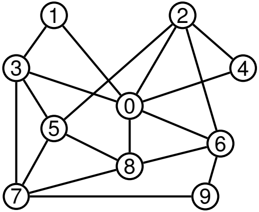


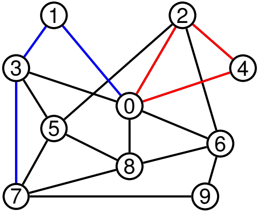

Leonhard Euler (1707-1783)

# The Königsberg Bridges Problem II

- A *graph* $G$ is a pair $(V, E)$ of a set $V$ of *vertices* and a set $E$ of *edges*
- A *path* from $u \in V$ to $w \in V$ in $G$ is a sequence $(u = v_0, v_1, v_2, \ldots, v_\ell = w)$ of vertices in $V$ such that $\{v_i, v_{i+1}\} \in E$ for all $i = 0, 1, \ldots, \ell - 1$
- A *cycle* or *tour* is a path whose first and last vertices are identical (i.e., $v_0 = v_\ell$)
- An *Eulerian tour* in $G$ is a tour visiting each edge exactly once
- A graph $G$ containing an Eulerian tour is said *Eulerian graph*

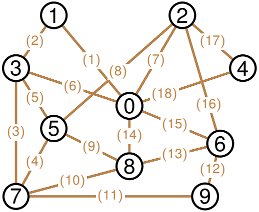



$V = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
$E = \{\{0, 1\}, \{0, 2\}, \{0, 3\}, \{0, 4\}, \{0, 6\}, \{0, 8\}, \{1, 3\}, \{2, 4\}, \{2, 5\}, \{2, 6\}, \{3, 5\}, \{3, 7\}, \{5, 7\}, \{5, 8\}, \{6, 8\}, \{6, 9\}, \{7, 8\}, \{7, 9\}\}$




Path from 0 to 7
(0, 1, 3, 7)

Tour
(0, 2, 4, 0)

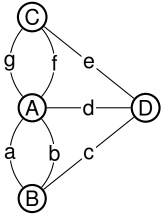



Eulerian tour
(0, 1, 3, 7, 5, 3, 0, 2, 5, 8, 7, 9, 6, 8, 0, 6, 2, 4, 0)

# The Königsberg Bridges Problem III

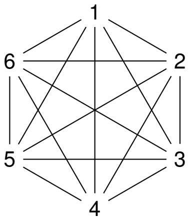
... Problem to Address

Is the Königsberg Bridges graph Eulerian?

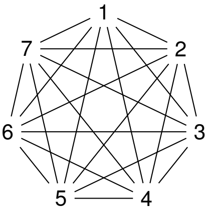


# Drawing Figures I

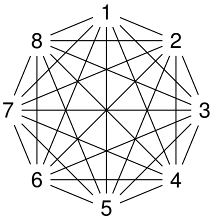
Given a figure with $n$ points, where each point is joined to every other point, when can it be drawn in one stroke without drawing each link more than once? [Poinsot 1810]



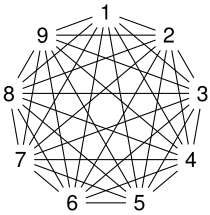

$n = 6$




$n = 7$




$n = 8$




$n = 9$

# Drawing Figures II

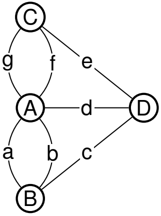
- A graph $G = (V, E)$ is *complete* if $\{i, j\} \in E$ for each pair $i, j \in V$ such that $i < j$

... Problem to Address

Is a complete graph with $|V| = n$ Eulerian?

# Eulerian Graphs Property

- A vertex $v \in V$ is *adjacent* to $u \in V$ if $\{v, u\} \in E$
- The *degree* $\delta(v)$ of $v \in V$ is the number of vertices adjacent to $v$ in $G$

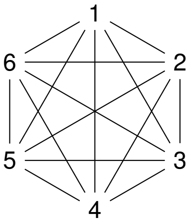



Not Eulerian

## Theorem [Hierholzer 1873]

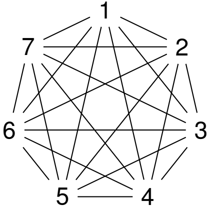
Necessary and sufficient condition for an undirected connected graph $G$ to be Eulerian:

*Every vertex has even degree*



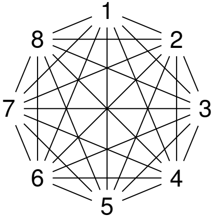

$n = 6$

Not Eulerian



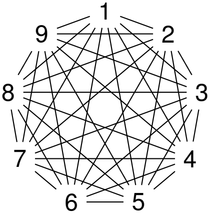

$n = 7$

Eulerian




$n = 8$

Not Eulerian




$n = 9$

Eulerian

# End-Pairing Algorithm

Given an Eulerian graph $G$, the End-Pairing Algorithm [Hierholzer 1873] finds an Eulerian tour in $G$

## End-Pairing Algorithm

Step 1. Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \dots, v, v_2, \dots, v_1)$ by iteratively traversing untraversed edges until $v_1$ is reached

Step 2. If all edges have been traversed, stop

Step 3. Build a second cycle $(v, u_1, \dots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

Step 4. Merge the two cycles $(v_1, \dots, v, v_2, \dots, v_1)$ and $(v, u_1, \dots, u_2, v)$ into the single cycle $(v_1, \dots, v, u_1, \dots, u_2, v, v_2, \dots, v_1)$

Step 5. Return to Step 2

Time complexity $O(|E|)$

# Example of End-Pairing Algorithm I

## End-Pairing Algorithm

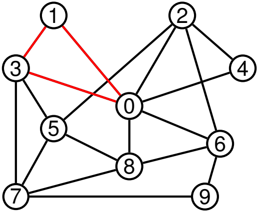
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \ldots, v, v_2, \ldots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \ldots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \ldots, v, v_2, \ldots, v_1)$ and $(v, u_1, \ldots, u_2, v)$ into the single cycle $(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

**Step 5.** Return to Step 2




$$(v_1, \ldots, v, v_2, \ldots, v_1) = (0, 1, 3, 0)$$

# Example of End-Pairing Algorithm II

## End-Pairing Algorithm

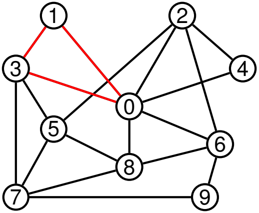
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \dots, v, v_2, \dots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \dots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \dots, v, v_2, \dots, v_1)$ and $(v, u_1, \dots, u_2, v)$ into the single cycle $(v_1, \dots, v, u_1, \dots, u_2, v, v_2, \dots, v_1)$

**Step 5.** Return to Step 2




$$(v_1, \dots, v, v_2, \dots, v_1) = (0, 1, 3, 0)$$

# Example of End-Pairing Algorithm III

## End-Pairing Algorithm

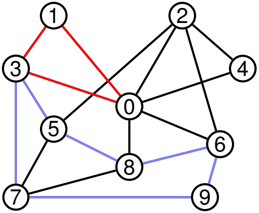
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \ldots, v, v_2, \ldots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \ldots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \ldots, v, v_2, \ldots, v_1)$ and $(v, u_1, \ldots, u_2, v)$ into the single cycle $(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

**Step 5.** Return to Step 2




$$(v_1, \ldots, v, v_2, \ldots, v_1) = (0, 1, 3, 0)$$
$$(v, u_1, \ldots, u_2, v) = (3, 7, 9, 6, 8, 5, 3)$$

# Example of End-Pairing Algorithm IV

## End-Pairing Algorithm

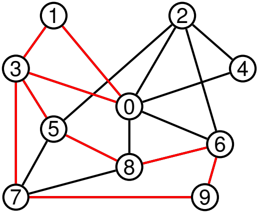
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \ldots, v, v_2, \ldots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \ldots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \ldots, v, v_2, \ldots, v_1)$ and $(v, u_1, \ldots, u_2, v)$ into the single cycle $(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

**Step 5.** Return to Step 2




$(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$
$(0, 1, 3, 7, 9, 6, 8, 5, 3, 0)$

# Example of End-Pairing Algorithm V

## End-Pairing Algorithm

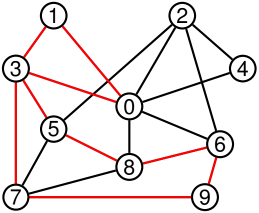
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \ldots, v, v_2, \ldots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \ldots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \ldots, v, v_2, \ldots, v_1)$ and $(v, u_1, \ldots, u_2, v)$ into the single cycle $(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

**Step 5.** Return to Step 2




$(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

$(0, 1, 3, 7, 9, 6, 8, 5, 3, 0)$

# Example of End-Pairing Algorithm VI

## End-Pairing Algorithm

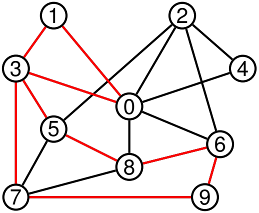
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \ldots, v, v_2, \ldots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \ldots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \ldots, v, v_2, \ldots, v_1)$ and $(v, u_1, \ldots, u_2, v)$ into the single cycle $(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

**Step 5.** Return to Step 2




$(v_1, \ldots, v, v_2, \ldots, v_1)$
$(0, 1, 3, 7, 9, 6, 8, 5, 3, 0)$

# Example of End-Pairing Algorithm VII

## End-Pairing Algorithm

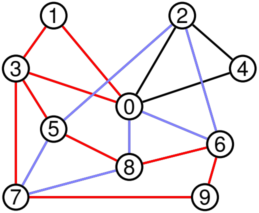
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \ldots, v, v_2, \ldots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \ldots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \ldots, v, v_2, \ldots, v_1)$ and $(v, u_1, \ldots, u_2, v)$ into the single cycle $(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

**Step 5.** Return to Step 2




$(v_1, \ldots, v, v_2, \ldots, v_1)$

$(0, 1, 3, 7, 9, 6, 8, 5, 3, 0)$

$(v, u_1, \ldots, u_2, v) = (6, 2, 5, 7, 8, 0, 6)$

# Example of End-Pairing Algorithm VIII

## End-Pairing Algorithm

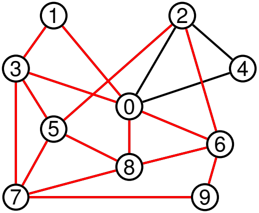
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \dots, v, v_2, \dots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \dots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \dots, v, v_2, \dots, v_1)$ and $(v, u_1, \dots, u_2, v)$ into the single cycle $(v_1, \dots, v, u_1, \dots, u_2, v, v_2, \dots, v_1)$

**Step 5.** Return to Step 2




$(v_1, \dots, v, u_1, \dots, u_2, v, v_2, \dots, v_1)$

$(0, 1, 3, 7, 9, 6, 2, 5, 7, 8, 0, 6, 8, 5, 3, 0)$

# Example of End-Pairing Algorithm IX

## End-Pairing Algorithm

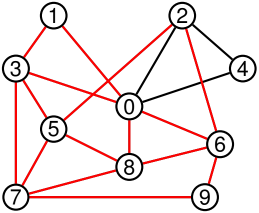
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \ldots, v, v_2, \ldots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \ldots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \ldots, v, v_2, \ldots, v_1)$ and $(v, u_1, \ldots, u_2, v)$ into the single cycle $(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

**Step 5.** Return to Step 2




$(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

$(0, 1, 3, 7, 9, 6, 2, 5, 7, 8, 0, 6, 8, 5, 3, 0)$

# Example of End-Pairing Algorithm X

## End-Pairing Algorithm

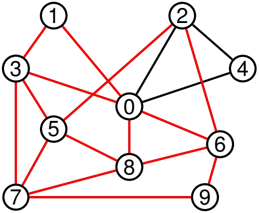
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \ldots, v, v_2, \ldots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \ldots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \ldots, v, v_2, \ldots, v_1)$ and $(v, u_1, \ldots, u_2, v)$ into the single cycle $(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

**Step 5.** Return to Step 2




$(v_1, \ldots, v, v_2, \ldots, v_1)$

$(0, 1, 3, 7, 9, 6, 2, 5, 7, 8, 0, 6, 8, 5, 3, 0)$

# Example of End-Pairing Algorithm XI

## End-Pairing Algorithm

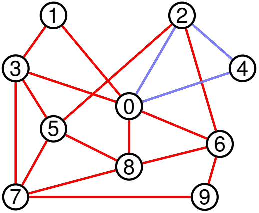
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \ldots, v, v_2, \ldots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \ldots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \ldots, v, v_2, \ldots, v_1)$ and $(v, u_1, \ldots, u_2, v)$ into the single cycle $(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

**Step 5.** Return to Step 2




$(v_1, \ldots, v, v_2, \ldots, v_1)$

$(0, 1, 3, 7, 9, 6, 2, 5, 7, 8, 0, 6, 8, 5, 3, 0)$

$(v, u_1, \ldots, u_2, v) = (0, 4, 2, 0)$

# Example of End-Pairing Algorithm XII

## End-Pairing Algorithm

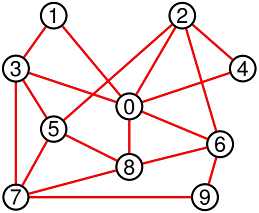
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \ldots, v, v_2, \ldots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \ldots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \ldots, v, v_2, \ldots, v_1)$ and $(v, u_1, \ldots, u_2, v)$ into the single cycle $(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

**Step 5.** Return to Step 2




$(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$
$(0, 1, 3, 7, 9, 6, 2, 5, 7, 8, 0, 4, 2, 0, 6, 8, 5, 3, 0)$

# Example of End-Pairing Algorithm XIII

## End-Pairing Algorithm

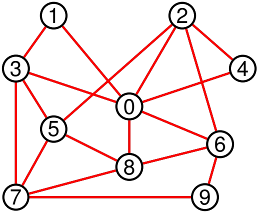
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \ldots, v, v_2, \ldots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \ldots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \ldots, v, v_2, \ldots, v_1)$ and $(v, u_1, \ldots, u_2, v)$ into the single cycle $(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

**Step 5.** Return to Step 2




$(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

$(0, 1, 3, 7, 9, 6, 2, 5, 7, 8, 0, 4, 2, 0, 6, 8, 5, 3, 0)$

# Example of End-Pairing Algorithm XIV

## End-Pairing Algorithm

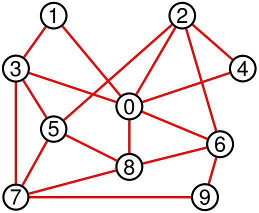
**Step 1.** Starting from an arbitrary vertex $v_1$, build a cycle $(v_1, \ldots, v, v_2, \ldots, v_1)$ by iteratively traversing untraversed edges until $v_1$

**Step 2.** If all edges have been traversed, stop

**Step 3.** Build a second cycle $(v, u_1, \ldots, u_2, v)$ starting from an edge $\{v, u_1\}$ incident to the first cycle and using untraversed edges only

**Step 4.** Merge the two cycles $(v_1, \ldots, v, v_2, \ldots, v_1)$ and $(v, u_1, \ldots, u_2, v)$ into the single cycle $(v_1, \ldots, v, u_1, \ldots, u_2, v, v_2, \ldots, v_1)$

**Step 5.** Return to Step 2




$(v_1, \ldots, v, v_2, \ldots, v_1)$

(0, 1, 3, 7, 9, 6, 2, 5, 7, 8, 0, 4, 2, 0, 6, 8, 5, 3, 0)

# Definition

The (Undirected) Chinese Postman Problem (CPP) [Guan 1962]

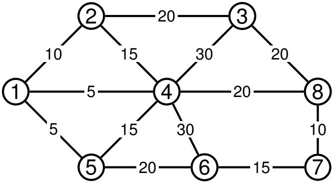
## Input

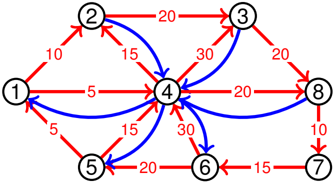
- An undirected graph $G = (V, E)$, where $V$ is the vertex set and $E$ is the edge set
- A cost $c_{ij}$ associated with each edge $\{i, j\} \in E$

## Goal

A least-cost tour that traverses all edges of $G$

The CPP models the mail delivery problem: “What is the shortest walk for a deliveryman who has to deliver mail in all streets of a network?”




Feasible CPP solution
$(1, 2, 3, 8, 7, 6, 4, 6, 5, 4, 5, 1, 4, 2, 4, 3, 4, 8, 4, 1)$
of cost $10 + 20 + 20 + 10 + 15 + 30 + 30 + 20$
$+15 + 15 + 5 + 5 + 15 + 15 + 30 + 30 + 20$
$+20 + 5 = 330$

# Eulerian Tour vs Chinese Postman

## Eulerian Tour Problem

- Feasibility problem
- No costs associated with edges
- Each edge traversed exactly once

## Chinese Postman Problem

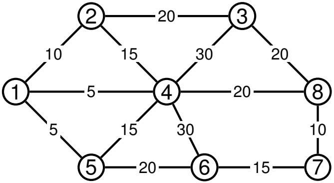
- Optimization problem (minimize cost)
- A cost associated with each edge
- Each edge traversed once or more

### Observation: Special Case of the CPP

If the graph $G$ is Eulerian, any Eulerian tour is an optimal CPP solution on graph $G$, whose cost equals the sum of the costs of all edges of $G$

- Therefore, if $G$ is Eulerian, solving the CPP is trivial and can be done with the End-Pairing algorithm
- The CPP is interesting if $G$ is not Eulerian and connected, which are the assumptions of the next slides

# Feasible Solutions of the CPP

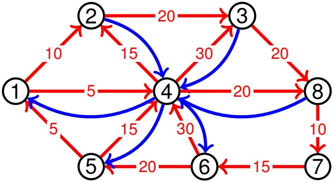



$G$ is not Eulerian, indeed

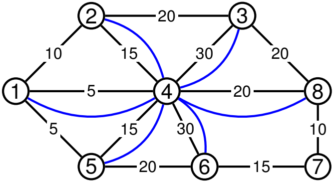
$$\delta(1) = \delta(2) = \delta(3) = \delta(5) = \delta(6) = \delta(8) = 3$$

$$\delta(4) = 6 \qquad \delta(7) = 2$$

The CPP solution (1, 2, 3, 8, 7, 6, 4, 6, 5, 4, 5, 1, 4, 2, 4, 3, 4, 8, 4, 1) is an Eulerian tour of a graph $\widehat{G}$ obtained from $G$ by duplicating some of the edges




$G$




$\widehat{G}$

Vertex degrees in $\widehat{G}$

$\delta(1) = 4 \quad \delta(2) = 4$

$\delta(3) = 4 \quad \delta(4) = 12$

$\delta(5) = 4 \quad \delta(6) = 4$

$\delta(7) = 2 \quad \delta(8) = 4$

# Formulation of the CPP I

## Observation: Solving the CPP

Because each feasible CPP solution on graph $G$ corresponds to an Eulerian tour of graph $\widehat{G}$, the CPP can be solved with a two-phase algorithm

1. Duplicate some of the edges of $G$ to obtain the Eulerian graph $\widehat{G}$
2. Find an Eulerian tour of $\widehat{G}$

- Phase 2 does not influence the cost of the CPP solution obtained (it is simply the sum of the costs of all edges) and can easily by solved with the End-Pairing algorithm
- The optimization problem consists of minimizing the costs of the edges added in Phase 1: this problem is called Augmentation Problem

## Observation: Solutions of the Augmentation Problem

- If $\delta(i)$ in $G$ is even, then an even number of edges incident to $i$ must be added
- If $\delta(i)$ in $G$ is odd, then an odd number of edges incident to $i$ must be added

# Formulation of the CPP II

## Augmentation Problem: Variables

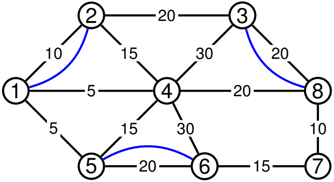
- $\xi_{ij} \in \mathbb{Z}_+$: number of times edge $\{i,j\} \in E$ is added
- $\pi_i \in \mathbb{Z}_+$: to ensure that a proper number of edges incident to $i \in V$ are added

$$
\begin{aligned}
\min \sum_{\{i,j\}\in E} & c_{ij}\xi_{ij} & \\
\text{s.t.} \sum_{\{i,j\}\in\delta(i)} & \xi_{ij} = 2\pi_i & \forall i \in V : |\delta(i)| \text{ is even} \\
\sum_{\{i,j\}\in\delta(i)} & \xi_{ij} = 2\pi_i + 1 & \forall i \in V : |\delta(i)| \text{ is odd} \\
& \xi_{i,j} \in \mathbb{Z}_+ & \forall \{i,j\} \in E \\
& \pi_i \in \mathbb{Z}_+ & \forall i \in V
\end{aligned}
$$

# Formulation of the CPP III

$$
\begin{array}{llllllllllllll}
\min & 10\xi_{12} & +5\xi_{14} & +5\xi_{15} & +20\xi_{23} & +15\xi_{24} & +30\xi_{34} & +20\xi_{38} & +15\xi_{45} & +30\xi_{46} & +20\xi_{48} & +20\xi_{56} & +15\xi_{67} & +10\xi_{78} \\
\text{s.t.} & \xi_{12} & +\xi_{14} & +\xi_{15} & & & & & & & & & & =2\pi_1+1 \\
& \xi_{12} & & & +\xi_{23} & +\xi_{24} & & & & & & & & =2\pi_2+1 \\
& & & & \xi_{23} & & +\xi_{34} & +\xi_{38} & & & & & & =2\pi_3+1 \\
& & \xi_{14} & & & +\xi_{24} & +\xi_{34} & & +\xi_{45} & +\xi_{46} & +\xi_{48} & & & =2\pi_4 \\
& & & \xi_{15} & & & & & +\xi_{45} & & & +\xi_{56} & & =2\pi_5+1 \\
& & & & & & & & & \xi_{46} & & +\xi_{56} & +\xi_{67} & =2\pi_6+1 \\
& & & & & & & & & & & & \xi_{67} & +\xi_{78} =2\pi_7 \\
& & & & & & & \xi_{38} & & & +\xi_{48} & & & +\xi_{78} =2\pi_8+1 \\
& \xi_{12}, & \xi_{14}, & \xi_{15}, & \xi_{23}, & \xi_{24}, & \xi_{34}, & \xi_{38}, & \xi_{45}, & \xi_{46}, & \xi_{48}, & \xi_{56}, & \xi_{67}, & \xi_{78} \in \mathbb{Z}_+ \\
\end{array}
$$

$$\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \pi_8 \in \mathbb{Z}_+$$




Optimal Solution $\xi_{12} = \xi_{38} = \xi_{56} = 1$

CPP Optimal Solution
(1, 2, 3, 8, 7, 6, 5, 4, 2, 1, 4, 3, 8, 4, 6, 5, 1)
of cost 265

# Alternative Solution Method for the CPP I

## Algorithm

Step 1. Identify the set $O$ of vertices of $G$ with odd degree

Step 2. Find the cost of the shortest path $s_{ij}$ between each pair of vertices $i, j \in O : i < j$

Step 3. Find the best combination of shortest paths by solving a min-cost matching problem

Step 4. Add the edges of the optimal solution of Step 3, and run the End-Pairing algorithm

# Alternative Solution Method for the CPP II

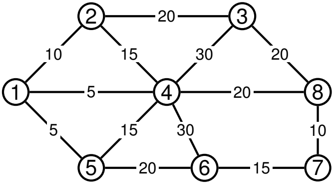
## Algorithm

Step 1. Identify the set $O$ of vertices of $G$ with odd degree

Step 2. Find the cost of the shortest path $s_{ij}$ between each pair of vertices $i, j \in O : i < j$

Step 3. Find the best combination of shortest paths by solving a min-cost matching problem

Step 4. Add the edges of the optimal solution of Step 3, and run the End-Pairing algorithm




$\delta(1) = 3 \quad \delta(2) = 3 \quad \delta(3) = 3 \quad \delta(4) = 6$

$\delta(5) = 3 \quad \delta(6) = 3 \quad \delta(7) = 2 \quad \delta(8) = 3$

$O = \{1, 2, 3, 5, 6, 8\}$

# Alternative Solution Method for the CPP III

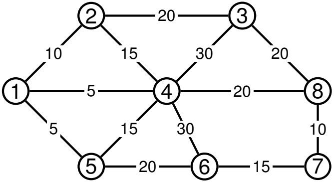
## Algorithm

**Step 1.** Identify the set $O$ of vertices of $G$ with odd degree

**Step 2.** Find the cost of the shortest path $s_{ij}$ between each pair of vertices $i, j \in O : i < j$

**Step 3.** Find the best combination of shortest paths by solving a min-cost matching problem

**Step 4.** Add the edges of the optimal solution of Step 3, and run the End-Pairing algorithm




$$O = \{1, 2, 3, 5, 6, 8\}$$

$$s_{ij} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 5 & 6 & 8 \end{array} \\ \begin{pmatrix} 10 & 30 & 5 & 25 & 25 \\ & 20 & 15 & 35 & 35 \\ & & 35 & 45 & 20 \\ & & & 20 & 35 \\ & & & & 25 \end{pmatrix} \end{array}$$

# Alternative Solution Method for the CPP IV

## Algorithm

**Step 1.** Identify the set $O$ of vertices of $G$ with odd degree

**Step 2.** Find the cost of the shortest path $s_{ij}$ between each pair of vertices $i, j \in O : i < j$

**Step 3.** Find the best combination of shortest paths by solving a min-cost matching problem

**Step 4.** Add the edges of the optimal solution of Step 3, and run the End-Pairing algorithm

### Variables

$\phi_{ij} \in \{0, 1\}$: equal to 1 if vertices $i \in O$ and $j \in O$ are matched (with $i < j$)

$$\begin{aligned} \min \quad & \sum_{i,j \in O : i < j} s_{ij}\phi_{ij} \\ \text{s.t.} \quad & \sum_{i \in O : i < k} \phi_{ik} + \sum_{j \in O : j > k} \phi_{kj} = 1 && \forall k \in O \\ & \phi_{ij} \in \{0, 1\} && i, j \in O : i < j \end{aligned}$$

# Alternative Solution Method for the CPP V

## Algorithm

**Step 1.** Identify the set $O$ of vertices of $G$ with odd degree

**Step 2.** Find the cost of the shortest path $s_{ij}$ between each pair of vertices $i, j \in O : i < j$

**Step 3.** Find the best combination of shortest paths by solving a min-cost matching problem

**Step 4.** Add the edges of the optimal solution of Step 3, and run the End-Pairing algorithm

$$s_{ij} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 5 & 6 & 8 \end{array} \\ \begin{pmatrix} 10 & 30 & 5 & 25 & 25 \\ & 20 & 15 & 35 & 35 \\ & & 35 & 45 & 20 \\ & & & 20 & 35 \\ & & & & 25 \end{pmatrix} \end{array}$$

Examples of feasible solutions

- $\phi_{15} = \phi_{23} = \phi_{68} = 1$ of cost 50
- $\phi_{12} = \phi_{56} = \phi_{38} = 1$ of cost 50
- $\phi_{13} = \phi_{26} = \phi_{58} = 1$ of cost 100

The red and brown solutions are optimal

# Alternative Solution Method for the CPP VI

## Algorithm

**Step 1.** Identify the set $O$ of vertices of $G$ with odd degree

**Step 2.** Find the cost of the shortest path $s_{ij}$ between each pair of vertices $i, j \in O : i < j$

**Step 3.** Find the best combination of shortest paths by solving a min-cost matching problem

**Step 4.** Add the edges of the optimal solution of Step 3, and run the End-Pairing algorithm

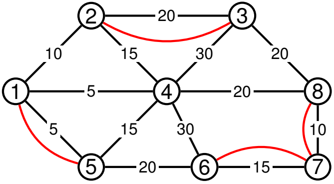
Optimal solution

$\phi_{15} = \phi_{23} = \phi_{68} = 1$

$$s_{ij} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 5 & 6 & 8 \end{array} \\ \begin{pmatrix} 10 & 30 & 5 & 25 & 25 \\ & 20 & 15 & 35 & 35 \\ & & 35 & 45 & 20 \\ & & & 20 & 35 \\ & & & & 25 \end{pmatrix} \end{array}$$

Edges of the shortest paths:

- From 1 to 5: $\{1, 5\}$
- From 2 to 3: $\{2, 3\}$
- From 6 to 8: $\{6, 7\}, \{7, 8\}$



# Alternative Solution Method for the CPP VII

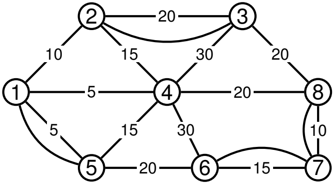
## Algorithm

**Step 1.** Identify the set $O$ of vertices of $G$ with odd degree

**Step 2.** Find the cost of the shortest path $s_{ij}$ between each pair of vertices $i, j \in O : i < j$

**Step 3.** Find the best combination of shortest paths by solving a min-cost matching problem

**Step 4.** Add the edges of the optimal solution of Step 3, and run the End-Pairing algorithm




Optimal solution
$(1, 2, 3, 4, 2, 3, 8, 7, 6, 7, 8, 4, 5, 1, 5, 6, 4, 1)$
of cost 265

# Definition I

The (Undirected) Rural Postman Problem (RPP) [Orloff 1974]

## Input

- An undirected graph $G = (V, E)$, where $V$ is the vertex set and $E$ is the edge set
- A cost $c_{ij}$ associated with each edge $\{i, j\} \in E$
- A subset $E_R \subseteq E$ of required edges

## Goal

A least-cost tour that traverses each required edge at least once

- Equivalently, the RPP consists of determining a least-cost set of deadhead edges that, together with $E_R$, yield an Eulerian graph
- Models mail delivery in rural areas where not all houses require deliveries
- If $E_R = E$, reduces to the CPP
- Is NP-hard [Lenstra and Rinnooy Kan 1976]

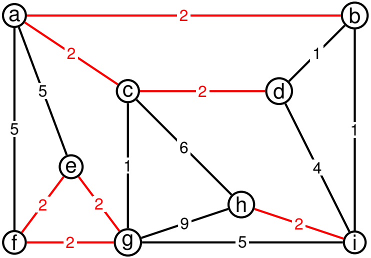
# Definition II

## Input

- An undirected graph $G = (V, E)$, where $V$ is the vertex set and $E$ is the edge set
- A cost $c_{ij}$ associated with each edge $\{i, j\} \in E$
- A subset $E_R \subseteq E$ of required edges

## Goal

A least-cost tour that traverses each required edge at least once




$$V = \{a, b, c, d, e, f, g, h, i\}$$

$$E = \{\{a, b\}, \{a, c\}, \{a, e\}, \{a, f\}, \{b, d\}, \{b, i\}, \{c, d\}, \{c, g\}, \{c, h\}, \{d, i\}, \{e, f\}, \{e, g\}, \{f, g\}, \{g, h\}, \{g, i\}, \{h, i\}\}$$

$$E_R = \{\{a, b\}, \{a, c\}, \{c, d\}, \{e, f\}, \{e, g\}, \{f, g\}, \{h, i\}\}$$

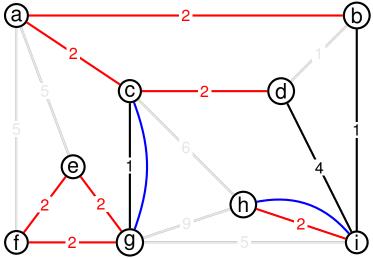
# Definition III

## Input

- An undirected graph $G = (V, E)$, where $V$ is the vertex set and $E$ is the edge set
- A cost $c_{ij}$ associated with each edge $\{i, j\} \in E$
- A subset $E_R \subseteq E$ of required edges

## Goal

A least-cost tour that traverses each required edge at least once




Feasible solution

$(a, b, i, h, i, d, c, g, f, e, g, c, a)$

- Edges $\{c, g\}$, $\{h, i\}$ traversed twice
- Edges $\{a, e\}$, $\{a, f\}$, $\{b, d\}$, $\{c, h\}$, $\{g, h\}$, $\{g, i\}$ not used
- Cost 2+1+2+2+4+2+1+2+2+2+1+2=23

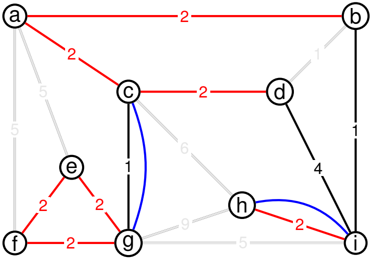
# Properties I

A vertex $i \in V$ is said to be R-odd (R-even, resp.) if an odd (even, resp.) number of required edges are incident to $i$, that is

- $i \in V$ is R-odd if $|\delta(i) \cap E_R|$ is odd
- $i \in V$ is R-even if $|\delta(i) \cap E_R|$ is even

## Property of Feasible Solutions

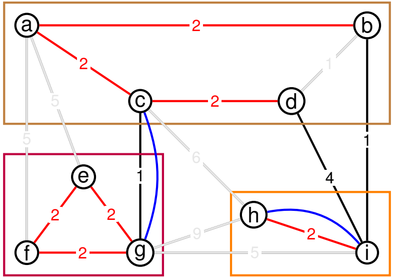
As any feasible solution is a tour, if $i$ is R-odd (R-even, resp.), then an odd (even, resp.) number of edges incident to $i$ are traversed as deadheading (i.e., with servicing them)




- $a$ is R-even $\Rightarrow$ 0 incident edges deadheaded
- $b$ is R-odd $\Rightarrow$ 1 incident edge deadheaded
- $c$ is R-even $\Rightarrow$ 2 incident edges deadheaded
- $i$ is R-odd $\Rightarrow$ 3 incident edges deadheaded

# Properties II

Identify the connected components $V_k$ $(k = 1, \ldots, p)$ of $G$ induced by $E_R$, that is




$p = 3$ connected components

- $V_1 = \{a, b, c, d\}$
- $V_2 = \{e, f, g\}$
- $V_3 = \{h, i\}$

## Property of Feasible Solutions

In any feasible solution, any subset of the connected components is linked to the other connected components by at least two edges (or by an edge traversed twice)

- $V_1$ is connected to $V_2 \cup V_3$ through edges $\{c, g\}$ (traversed twice), $\{b, i\}$ and $\{d, i\}$
- $V_2$ is connected to $V_1 \cup V_3$ through edge $\{c, g\}$ traversed twice
- $V_3$ is connected to $V_1 \cup V_2$ through edges $\{b, i\}$ and $\{d, i\}$

# Formulation of the RPP

Mathematical formulation proposed by [Corberán and Sanchis 1994]

## Variables

- $\xi_{ij} \in \mathbb{Z}_+$: number of times edge $\{i,j\} \in E$ is deadheaded
- $\pi_i \in \mathbb{Z}_+$: to ensure that a proper number of edges incident to $i \in V$ are added

$$
\begin{aligned}
\min \; & \sum_{\{i,j\}\in E} c_{ij}\xi_{ij} & \\
\text{s.t.} \; & \sum_{\{i,j\}\in\delta(i)} \xi_{ij} = 2\pi_i & \forall i \in V : i \text{ is R-even} \\
& \sum_{\{i,j\}\in\delta(i)} \xi_{ij} = 2\pi_i + 1 & \forall i \in V : i \text{ is R-odd} \\
& \sum_{\{i,j\}\in\delta(S)} \xi_{ij} \geq 2 & \forall S = \cup_{k\in P} V_k : P \subset \{1,2,\dots,p\}, P \neq \emptyset \\
& \xi_{i,j} \in \mathbb{Z}_+ & \forall \{i,j\} \in E \\
& \pi_i \in \mathbb{Z}_+ & \forall i \in V
\end{aligned}
$$

Branch-and-cut algorithms solve RPP instances with $|V| = 1000$, $|E| \leq 3080$, $p \leq 205$

# Constructive Heuristic

## Algorithm of [Frederickson 1979]

Step 1. Construct a Shortest Spanning Tree (SST) over an auxiliary graph $H$ containing one vertex for each connected component of $E_R$. The cost between any two vertices of $H$ is equal to that of a shortest path between them on $G$. Let $T$ be the edge set of the SSP

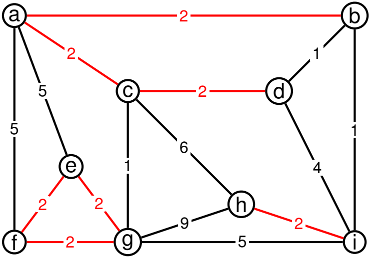
Step 2. Solve a minimum cost matching problem on the set of odd-degree vertices $O$ in the graph induced by $E_R \cup T$, where the costs are those of shortest paths on $G$. Let $M$ be the set of edges induced by the matching

Step 3. Determine an Eulerian tour in the graph induced by $E_R \cup T \cup M$

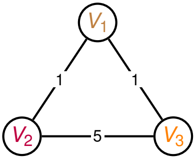
# Frederickson Algorithm: Step I

## Step 1

Construct a Shortest Spanning Tree (SST) over an auxiliary graph $H$ containing one vertex for each connected component of $E_R$. The cost between any two vertices of $H$ is equal to that of a shortest path between them on $G$. Let $T$ be the edge set of the SSP

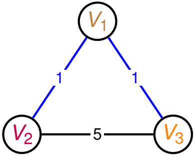



Graph $H$




Shortest Spanning Tree

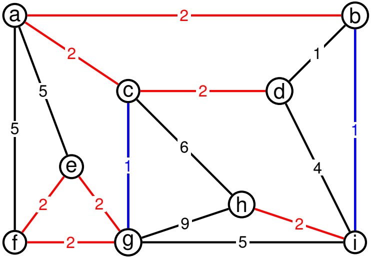
$T = \{\{c, g\}, \{b, i\}\}$



# Frederickson Algorithm: Step II

## Step 2

Solve a minimum cost matching problem on the set of odd-degree vertices $O$ in the graph induced by $E_R \cup T$, where the costs are those of shortest paths on $G$. Let $M$ be the set of edges induced by the matching




$$O = \{c, d, g, h\}$$

$$\begin{array}{c} \\ c \\ d \\ g \end{array} \begin{array}{c} \begin{array}{ccc} d & g & h \end{array} \\ \begin{pmatrix} 2 & 1 & 6 \\ & 3 & 4 \\ & & 7 \end{pmatrix} \end{array}$$

Three solutions of the matching problem

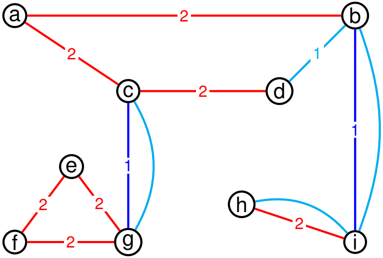
- $c - d$ and $g - h$ of cost 9
- $c - g$ and $d - h$ of cost 5
- $c - h$ and $d - g$ of cost 9

Optimal matching problem solution $c - g$, $d - h$, so $M = \{\{c, g\}, \{b, d\}, \{b, i\}, \{h, i\}\}$

# Frederickson Algorithm: Step III

## Step 3

Determine an Eulerian tour in the graph induced by $E_R \cup T \cup M$




RPP Solution
$(a, b, i, h, i, b, d, c, g, f, e, g, c, a)$
of cost 21

# Frederickson Algorithm: Properties

- Does not guarantee to find an optimal solution
- If $E_R$ is connected, it is exact
- If costs $c_{ij}$ satisfy the triangle inequality, it has a worst-case performance of $\frac{3}{2}$

# Bibliography I


Á. Corberán and J. M. Sanchis
A Polyhedral Approach for the Rural Postman Problem
*European Journal of Operational Research* 79, 95–114, 1994

L. Euler
Solutio Problematis ad Geometriam Situs Pertinentis
*Commentarii Academiae Scientiarum Imperialis Petropolitanae* 8, 128–140, 1736

G. N. Frederickson
Approximation Algorithms for Some Routing Problems
*Journal of the ACM* 26, 538–554, 1979

M. Guan
Graphi Programming using Odd and Even Points
*Chinese Mathematics* 1, 273–277, 1962

C. Hierholzer
Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren
*Mathematische Annalen* 6, 30–32, 1873

J. K. Lenstra and A. H. G. Rinnooy Kan
On General Routing Problems
*Networks* 6, 273–280, 1976

C. S. Orloff
A Fundamental Theorem in Vehicle Routing
*Networks* 27, 35–64, 1974

L. Poinsot
Mémoire sur les Polygones et les Polyèdres
*Journal de l'École Polytechnique* 4, 16–49, 1810.

# Contact

Evelien van der Hurk
Department of Management Engineering, Technical University of Denmark (DTU)

Building 426, Room 042 & Building 115
2800 Kgs. Lyngby, Denmark

evdh@dtu.dk
+45 45254821 phone