

# Assignment 3b: The Traveling Salesman Problem

MVE165/MMG631: Linear and integer optimisation with applications

Emil Gustavsson

Mathematical Sciences, Chalmers University of Technology

May 5, 2015

- 1 Problem background
- 2 Mathematical formulation
- 3 Algorithms
  - Heuristics
  - Relaxation algorithms
- 4 Assignment

# Traveling Salesman Problem (TSP)

- One of the most studied problems in the area of optimization.
- The name is a mystery, but gives a clear connection to the applications of the problem.
- 1832, handbook *Der Handlungsreisende* for traveling salesmen.
- Stated as a mathematical problem in the 1930's:

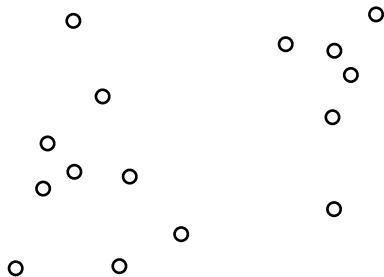
# Traveling Salesman Problem (TSP)

- One of the most studied problems in the area of optimization.
- The name is a mystery, but gives a clear connection to the applications of the problem.
- 1832, handbook *Der Handlungsreisende* for traveling salesmen.
- Stated as a mathematical problem in the 1930's:

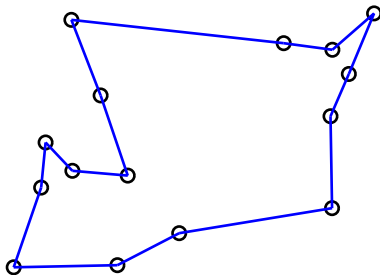
## Definition: Traveling Salesman Problem.

*Given a list of cities and their pairwise distances, find the shortest possible tour that visits each city exactly once.*

# Example



# Example



The TSP has many applications.

- Logistics.
- Production (microchips).
- DNA-sequencing.
- Agriculture.
- Internet planning.

The TSP is a *NP-complete* problem (the decision version of it).

- No polynomial algorithm for solving it to optimality.
- Exponential in the number of cities.
- $(N - 1)!$  different tours.

Large problems solved to optimality

- VLSI problem (85,900 nodes). Solved 2004, first studied 1991.
- Shortest tour between all the cities in Sweden (24,978 cities) found in 2001. Length  $\approx 72,500$  km



# Special cases

Let  $c_{ij}$  be the distance from city  $i$  to city  $j$ .

## Symmetric TSP (undirected graph)

$$c_{ij} = c_{ji}, \quad \forall \text{ cities } i, j$$

# Special cases

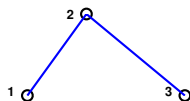
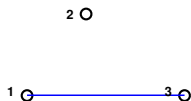
Let  $c_{ij}$  be the distance from city  $i$  to city  $j$ .

## Symmetric TSP (undirected graph)

$$c_{ij} = c_{ji}, \quad \forall \text{ cities } i, j$$

## Metric TSP (triangle inequality satisfied)

$$c_{ik} + c_{kj} \geq c_{ij}, \quad \forall \text{ cities } i, j, k$$



# Integer linear program (symmetric TSP)

- Consider a set  $\mathcal{N} = \{1, \dots, N\}$  of cities.
- Let  $c_{ij}$  be the distance from city  $i$  to city  $j$ .
- Let  $\mathcal{L} \subseteq \mathcal{N} \times \mathcal{N}$  denote the undirected links in the graph. We use  $\mathcal{L} = \{(i, j) : i, j \in \mathcal{N}, i < j\}$
- Introduce binary variables  $x_{ij}$  where

$$x_{ij} = \begin{cases} 1 & \text{if there is a connection between city } i \text{ and city } j \\ 0 & \text{otherwise} \end{cases}$$

# Linear integer program (symmetric TSP)

$$\begin{aligned} & \text{minimize} && \sum_{(i,j) \in \mathcal{L}} c_{ij} x_{ij}, \\ & \text{subject to} && \sum_{j \in \mathcal{N}: (i,j) \in \mathcal{L}} x_{ij} + \sum_{j \in \mathcal{N}: (j,i) \in \mathcal{L}} x_{ji} = 2, && i \in \mathcal{N} \\ & && \sum_{(i,j) \in \mathcal{L}: \{i,j\} \in \mathcal{S}} x_{ij} \leq |\mathcal{S}| - 1, \quad \forall \mathcal{S} \subset \mathcal{N} : 2 \leq |\mathcal{S}| \leq N - 2 \\ & && x_{ij} \in \{0, 1\}, && i, j \in \mathcal{N}. \end{aligned}$$

- Objective is to minimize the total length of the tour.
- First constraints makes sure that we visit each city once.
- Second constraint makes sure that no subtours are allowed.

# Linear integer program (symmetric TSP)

$$\begin{aligned} &\text{minimize} && \sum_{(i,j) \in \mathcal{L}} c_{ij} x_{ij}, \\ &\text{subject to} && \sum_{j \in \mathcal{N}: (i,j) \in \mathcal{L}} x_{ij} + \sum_{j \in \mathcal{N}: (j,i) \in \mathcal{L}} x_{ji} = 2, && i \in \mathcal{N} \\ &&& \sum_{(i,j) \in \mathcal{L}: \{i,j\} \in \mathcal{S}} x_{ij} \leq |\mathcal{S}| - 1, \quad \forall \mathcal{S} \subset \mathcal{N} : 2 \leq |\mathcal{S}| \leq N - 2 \\ &&& x_{ij} \in \{0, 1\}, && i, j \in \mathcal{N}. \end{aligned}$$

- Objective is to minimize the total length of the tour.
- First constraints makes sure that we visit each city once.
- Second constraint makes sure that no subtours are allowed.

# Linear integer program (symmetric TSP)

$$\begin{aligned} & \text{minimize} && \sum_{(i,j) \in \mathcal{L}} c_{ij} x_{ij}, \\ & \text{subject to} && \sum_{j \in \mathcal{N}: (i,j) \in \mathcal{L}} x_{ij} + \sum_{j \in \mathcal{N}: (j,i) \in \mathcal{L}} x_{ji} = 2, && i \in \mathcal{N} \\ & && \sum_{(i,j) \in \mathcal{L}: \{i,j\} \in \mathcal{S}} x_{ij} \leq |\mathcal{S}| - 1, \quad \forall \mathcal{S} \subset \mathcal{N} : 2 \leq |\mathcal{S}| \leq N - 2 \\ & && x_{ij} \in \{0, 1\}, && i, j \in \mathcal{N}. \end{aligned}$$

- Objective is to minimize the total length of the tour.
- First constraints makes sure that we visit each city once.
- Second constraint makes sure that no subtours are allowed.

# Linear integer program (symmetric TSP)

$$\begin{aligned} & \text{minimize} && \sum_{(i,j) \in \mathcal{L}} c_{ij} x_{ij}, \\ & \text{subject to} && \sum_{j \in \mathcal{N}: (i,j) \in \mathcal{L}} x_{ij} + \sum_{j \in \mathcal{N}: (j,i) \in \mathcal{L}} x_{ji} = 2, && i \in \mathcal{N} \\ & && \sum_{(i,j) \in \mathcal{L}: \{i,j\} \in \mathcal{S}} x_{ij} \leq |\mathcal{S}| - 1, \quad \forall \mathcal{S} \subset \mathcal{N} : 2 \leq |\mathcal{S}| \leq N - 2 \\ & && x_{ij} \in \{0, 1\}, && i, j \in \mathcal{N}. \end{aligned}$$

- Objective is to minimize the total length of the tour.
- First constraints makes sure that we visit each city once.
- **Second constraint makes sure that no subtours are allowed.**

# Subtour constraints

The no subtour constraint can also be formulated in two ways



# Subtour constraints

The no subtour constraint can also be formulated in two ways

- The number of links in any subset should be less than the number of cities in the subset.

$$\sum_{(i,j) \in \mathcal{L}: \{i,j\} \in \mathcal{S}} x_{ij} \leq |\mathcal{S}| - 1, \quad \forall \mathcal{S} \subset \mathcal{N} : 2 \leq |\mathcal{S}| \leq N - 2.$$

# Subtour constraints

The no subtour constraint can also be formulated in two ways

- The number of links in any subset should be less than the number of cities in the subset.

$$\sum_{(i,j) \in \mathcal{L}: \{i,j\} \in \mathcal{S}} x_{ij} \leq |\mathcal{S}| - 1, \quad \forall \mathcal{S} \subset \mathcal{N} : 2 \leq |\mathcal{S}| \leq N - 2.$$

- From each subset of cities, we must travel at least once to another city not included in the subset.

$$\sum_{(i,j) \in \mathcal{L}: i \in \mathcal{S}, j \in \mathcal{N} \setminus \mathcal{S}} x_{ij} + \sum_{(j,i) \in \mathcal{L}: i \in \mathcal{S}, j \in \mathcal{N} \setminus \mathcal{S}} x_{ji} \geq 2, \quad \forall \mathcal{S} \subset \mathcal{N} : 2 \leq |\mathcal{S}| \leq N - 2.$$

# Subtour constraints

The no subtour constraint can also be formulated in two ways

- The number of links in any subset should be less than the number of cities in the subset.

$$\sum_{(i,j) \in \mathcal{L}: \{i,j\} \in \mathcal{S}} x_{ij} \leq |\mathcal{S}| - 1, \quad \forall \mathcal{S} \subset \mathcal{N} : 2 \leq |\mathcal{S}| \leq N - 2.$$

- From each subset of cities, we must travel at least once to another city not included in the subset.

$$\sum_{(i,j) \in \mathcal{L}: i \in \mathcal{S}, j \in \mathcal{N} \setminus \mathcal{S}} x_{ij} + \sum_{(j,i) \in \mathcal{L}: i \in \mathcal{S}, j \in \mathcal{N} \setminus \mathcal{S}} x_{ji} \geq 2, \quad \forall \mathcal{S} \subset \mathcal{N} : 2 \leq |\mathcal{S}| \leq N - 2.$$

One problem with the model is that the number of subtour constraints in both formulations are  $\approx 2^N$ .

Exact algorithms.

- *Brute force technique.* Enumerate all possible tours, choose the shortest one.  $\sim \mathcal{O}(N!)$
- *Held-Karp algorithm.* Use dynamic programming.  $\sim \mathcal{O}(N^2 2^N)$

Prize for finding an algorithm  $\sim \mathcal{O}(1.9999^N)$

Exact algorithms.

- *Brute force technique.* Enumerate all possible tours, choose the shortest one.  $\sim \mathcal{O}(N!)$
- *Held-Karp algorithm.* Use dynamic programming.  $\sim \mathcal{O}(N^2 2^N)$

Prize for finding an algorithm  $\sim \mathcal{O}(1.9999^N)$

Have to use

- *Heuristics:* Find feasible and acceptable solutions.
- *Relaxation algorithms:* Produce lower bounds on optimal objective value.

Together these algorithms can provide us with an *optimality interval*

Strategies, rules for obtaining feasible solutions.

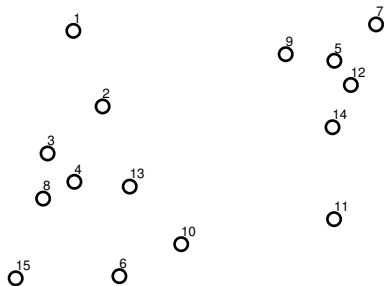
## *Deterministic*

- Based on simple rules for choosing tours: *Nearest neighbour, Insertion heuristics*
- Based on solving easier subproblems. *MST-heuristic, Christophides heuristic*

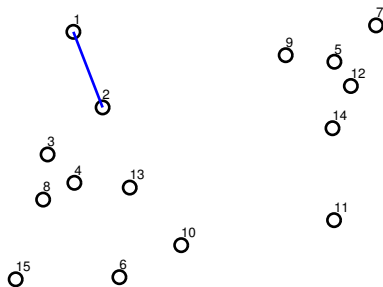
## *Probabilistic*

- Based on stochastic rules for choosing tours.
- *Genetic algorithms, simulated annealing, ant colony optimization.*

# Nearest neighbour heuristic

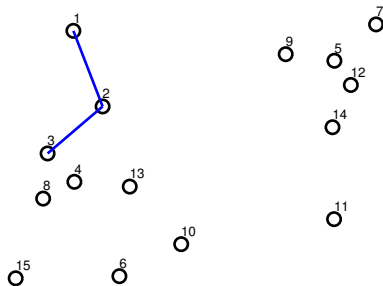


# Nearest neighbour heuristic

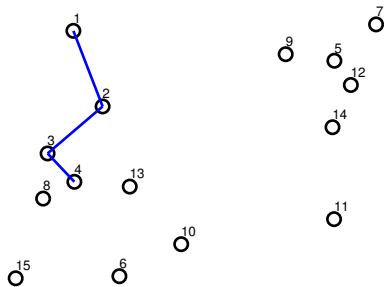




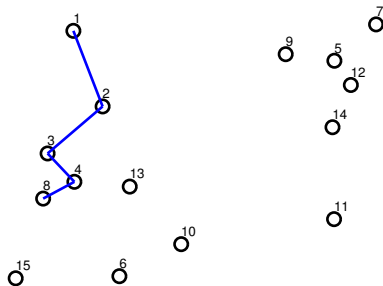
# Nearest neighbour heuristic



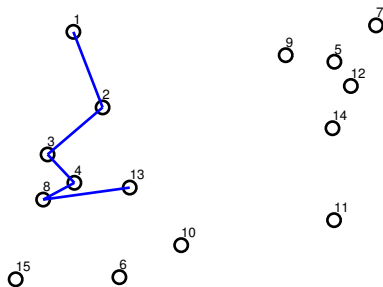
# Nearest neighbour heuristic



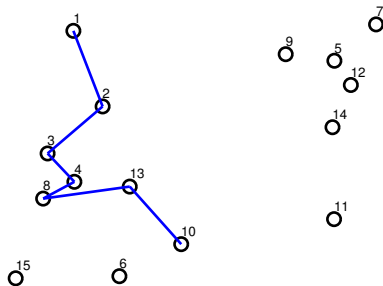
# Nearest neighbour heuristic



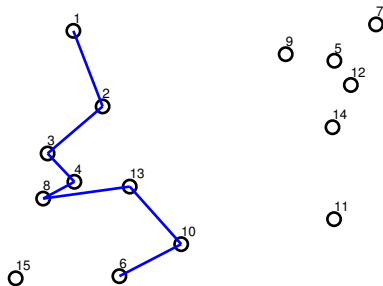
# Nearest neighbour heuristic



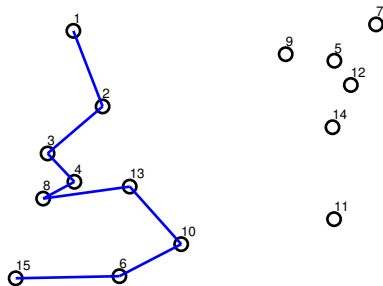
# Nearest neighbour heuristic



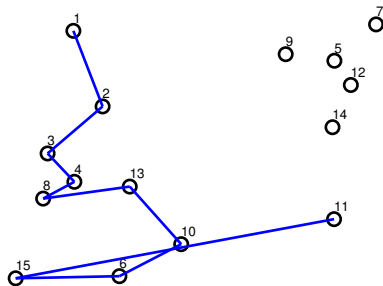
# Nearest neighbour heuristic



# Nearest neighbour heuristic

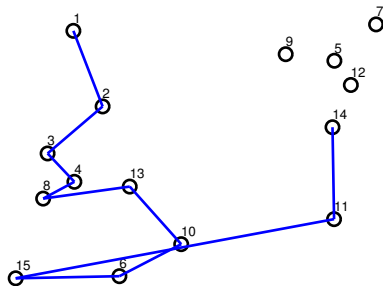


# Nearest neighbour heuristic

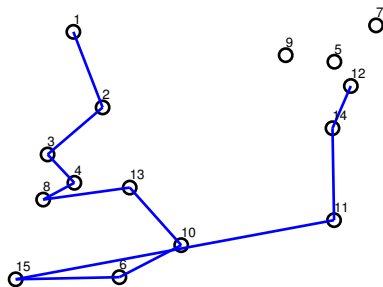




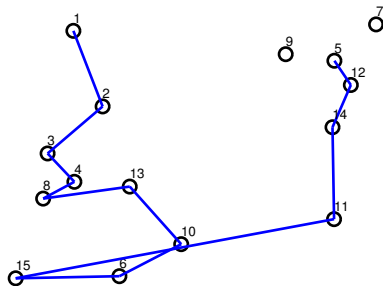
# Nearest neighbour heuristic



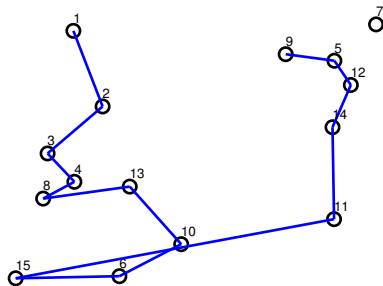
# Nearest neighbour heuristic



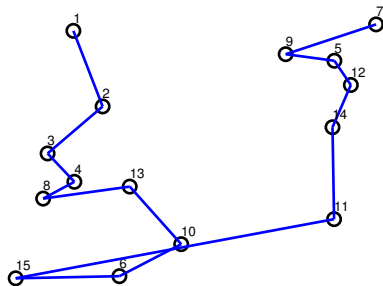
# Nearest neighbour heuristic



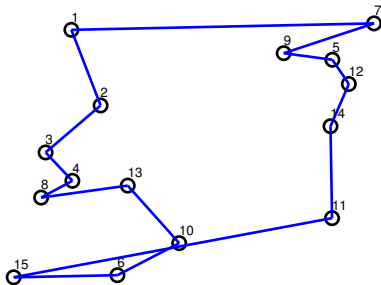
# Nearest neighbour heuristic



# Nearest neighbour heuristic



# Nearest neighbour heuristic



# Improvement heuristics

Algorithms for improving a feasible solution. Local search heuristics. Utilize the fact that we are considering *metric* TSP problems.



Examples: *k-opt heuristics, crossing elimination*

# Relaxation algorithms

Gives lower bounds on objective value. Trick is to relax the problem such that

- the reduced problem is easy to solve, and
- the lower bound given by the relaxation is good.

Tradeoff between the two objectives.

Examples: *Branch and bound*, *Cutting plane methods*, *1-tree Lagrangian relaxation*.



# 1-tree Lagrangian relaxation

Idea:

- Lagrangian relax the assignment constraints

$$\sum_{j \in \mathcal{N}: (i,j) \in \mathcal{L}} x_{ij} + \sum_{j \in \mathcal{N}: (j,i) \in \mathcal{L}} x_{ji} = 2 \quad i \in \mathcal{N}$$

for all nodes except one, say node  $s$ . This means assigning a Lagrangian multiplier to each node (node price).

# 1-tree Lagrangian relaxation

Idea:

- Lagrangian relax the assignment constraints

$$\sum_{j \in \mathcal{N}: (i,j) \in \mathcal{L}} x_{ij} + \sum_{j \in \mathcal{N}: (j,i) \in \mathcal{L}} x_{ji} = 2 \quad i \in \mathcal{N}$$

for all nodes except one, say node  $s$ . This means assigning a Lagrangian multiplier to each node (node price).

- Resulting problem is a *1-MST problem*, which is the problem of finding a minimum spanning tree on the nodes  $\mathcal{N} \setminus \{s\}$ , and then connecting node  $s$  to the tree by two links.

# 1-tree Lagrangian relaxation

Idea:

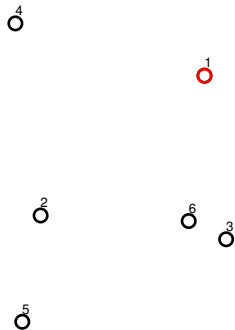
- Lagrangian relax the assignment constraints

$$\sum_{j \in \mathcal{N}: (i,j) \in \mathcal{L}} x_{ij} + \sum_{j \in \mathcal{N}: (j,i) \in \mathcal{L}} x_{ji} = 2 \quad i \in \mathcal{N}$$

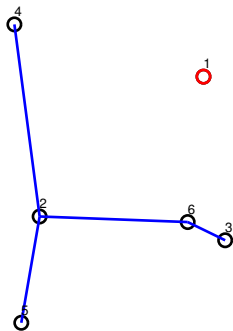
for all nodes except one, say node  $s$ . This means assigning a Lagrangian multiplier to each node (node price).

- Resulting problem is a *1-MST problem*, which is the problem of finding a minimum spanning tree on the nodes  $\mathcal{N} \setminus \{s\}$ , and then connecting node  $s$  to the tree by two links.
- Iteratively updating the node prices such that we increase our lower bound in each iteration.

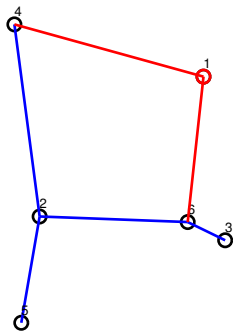
# 1-tree Lagrangian relaxation



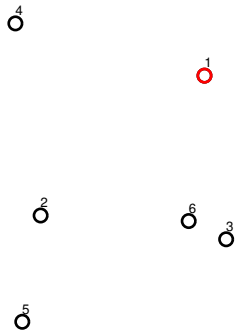
# 1-tree Lagrangian relaxation



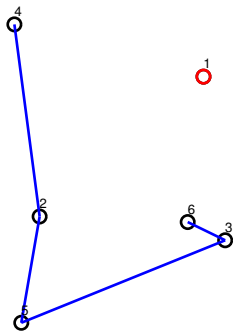
# 1-tree Lagrangian relaxation



# 1-tree Lagrangian relaxation

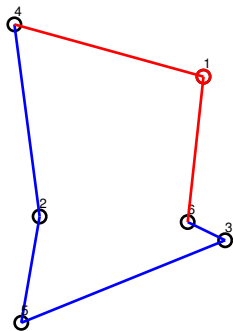


# 1-tree Lagrangian relaxation





# 1-tree Lagrangian relaxation



# Assignment 3a: The Traveling Salesman Problem

What do you do in the assignment?

- You get familiar with one of the most studied problems in optimization.
- You will use CPLEX to solve some small problems.
- You develop and implement different algorithms, both heuristics and relaxation algorithms.
- You get more familiar with *either* the theory of relaxation algorithms *or* probabilistic heuristics.

# Optimal tour of Sweden

