

# Travelling Salesman Problems

Alessandro Hill

rev. 1.0(AH) – 2024



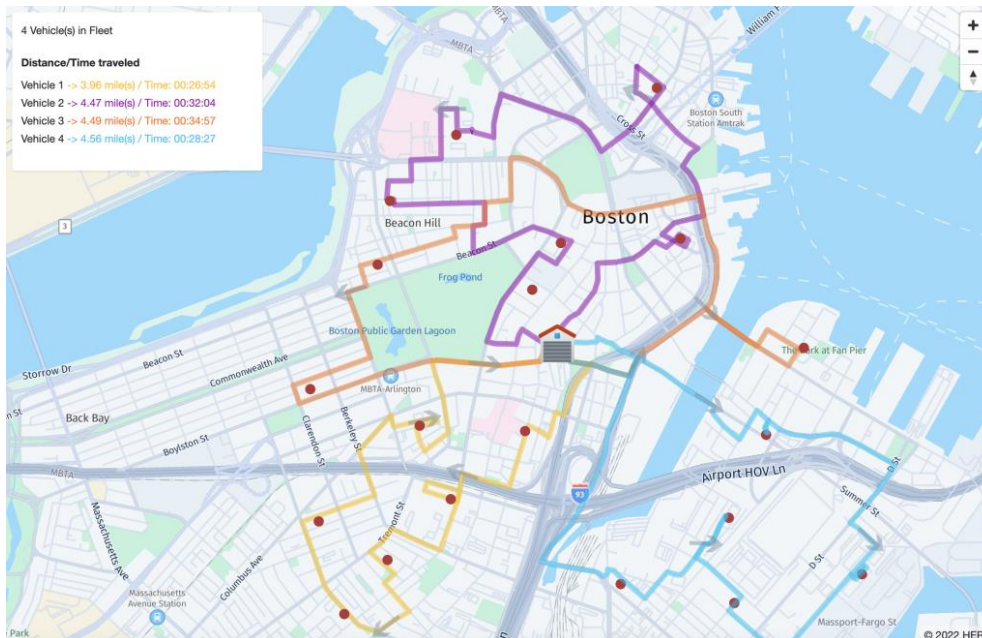
# The Traveling Salesman Problem

Problema del commesso viaggiatore

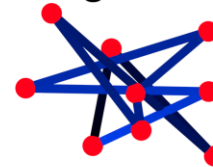
We are given a directed base graph with node set  $N$  (locations) and arc set  $A$  (travel segments). Each arc  $(i,j)$  has a non-negative weight  $c_{i,j}$  (distance/duration).

The **Travelling Salesman Problem (TSP)** asks for a tour (directed cycle, or circuit) of minimum total length that includes every node exactly once.\*

\*NP-hard (extremely difficult to optimize)



Length: 16.75

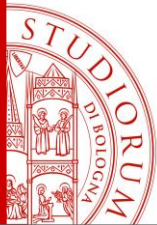


Random Path

Length: 8.53

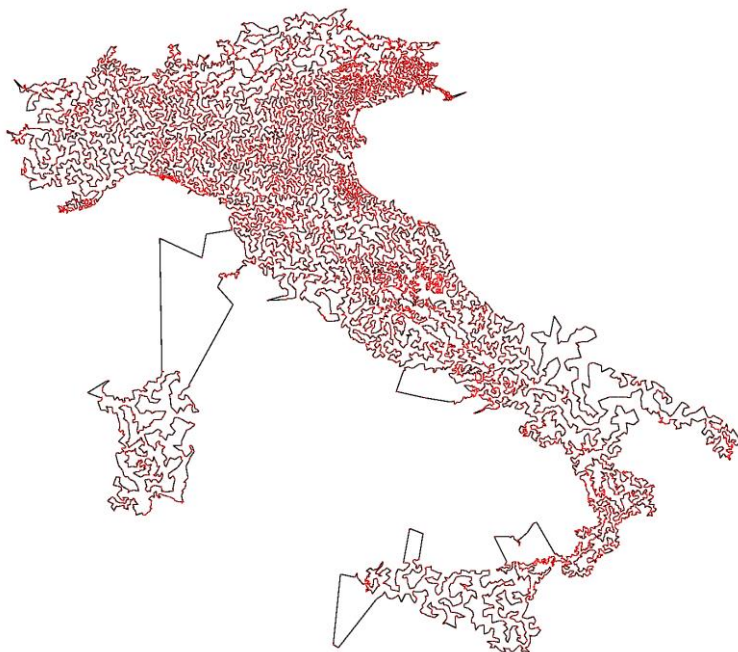


Optimized Path



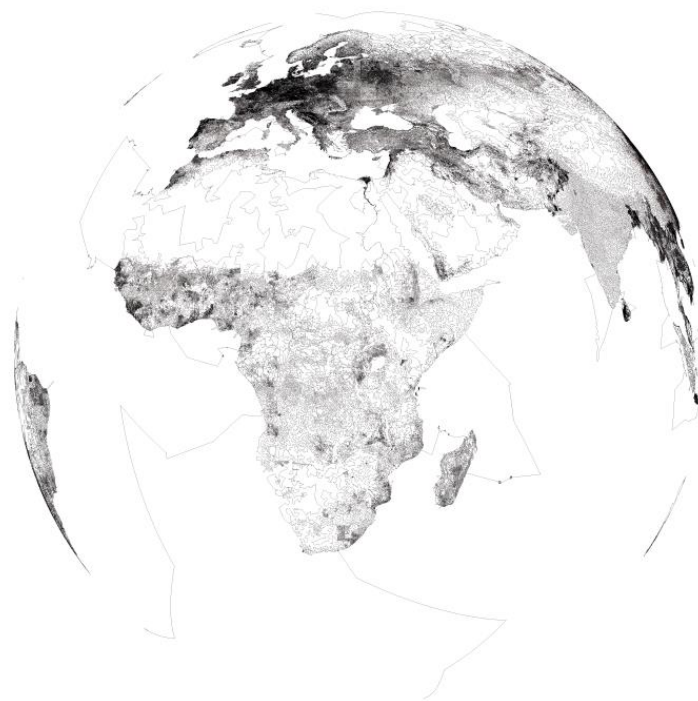
# The Traveling Salesman Problem

**16,862 Cities in Italy**



**557,315 km**

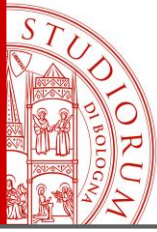
**1,904,711 Cities Worldwide**



**7,516,353,779 km**

Also see <http://www.math.uwaterloo.ca/tsp/world/index.html>



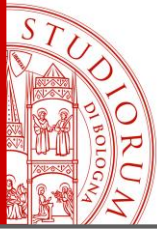


# The Traveling Salesman Problem

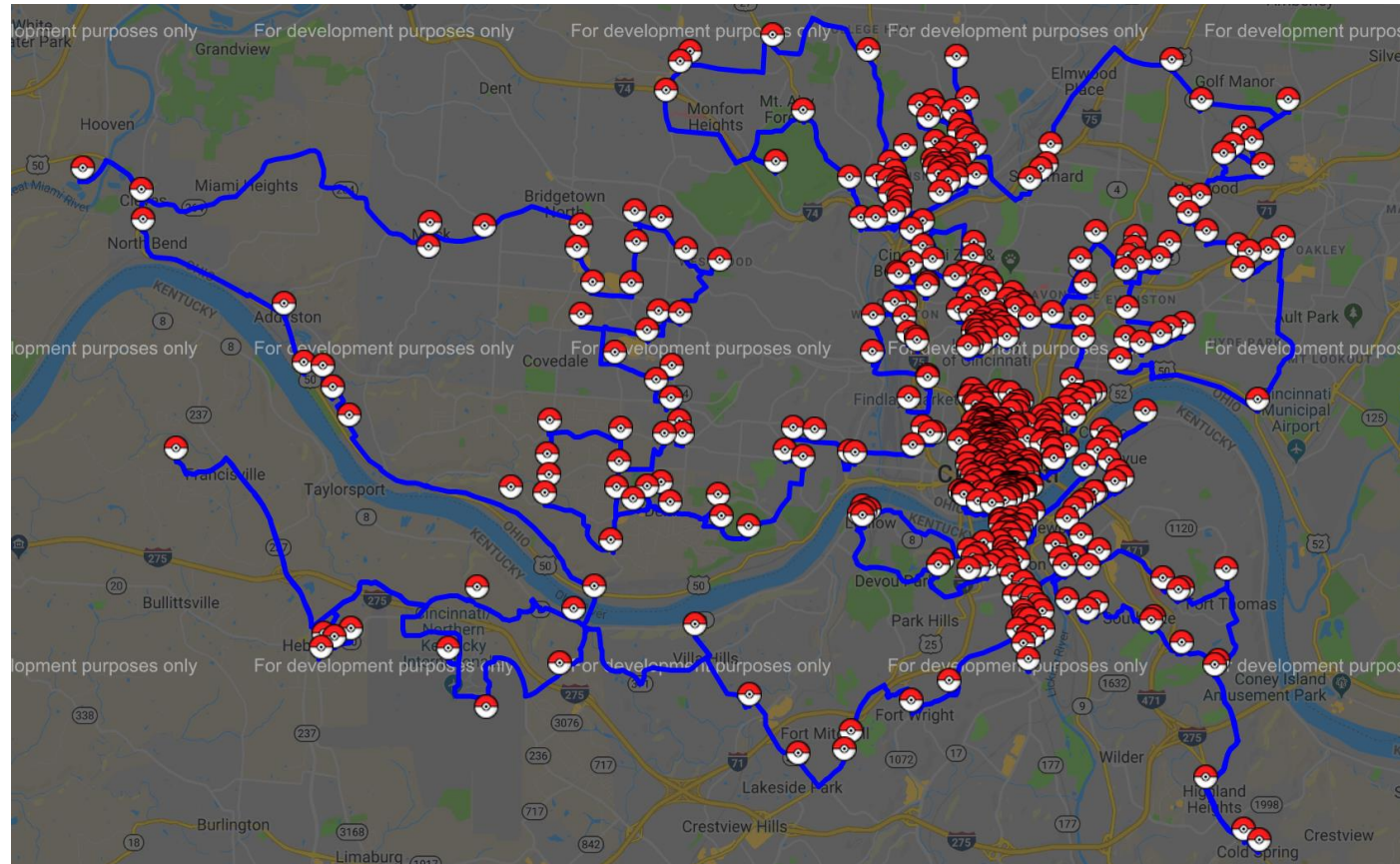


Also see <http://www2.oberlin.edu/math/faculty/bosch/tspart-page.html> and <https://cs.uwaterloo.ca/~csk/other/tsp/>

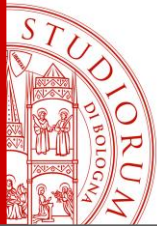




# The Traveling Salesman Problem



Also see <http://www.math.uwaterloo.ca/tsp/poke/index.html>



# The Traveling Salesman Problem

## IP Model\*

NON CI DEVONO ESSERE AUTOCICLI SU NODI SINGOLI (AD ESEMPIO 'A' NON ESCE DA 'A' PER ENTRARE IN 'A')

**Binary arc variables for each arc that could be part of a tour:**

$$x_{i,j} = \begin{cases} 1 & \text{if arc } (i,j) \text{ will be used on the tour,} \\ 0 & \text{otherwise.} \end{cases}$$

**Minimize**  $\sum_{(i,j) \in A} c_{i,j} x_{i,j}$

Idea: Each node  $v$  has exactly one incoming arc and one outgoing arc in a tour.

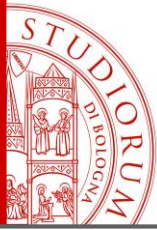
**Force node out-arc:**

$$x_{v,j_1} + \dots + x_{v,j_k} = 1 \quad \text{for out-neighbors } j_1, \dots, j_k \text{ of } v.$$

**Force node in-arc:**

$$x_{i_1,v} + \dots + x_{i_k,v} = 1 \quad \text{for in-neighbors } i_1, \dots, i_k \text{ of } v.$$

\*May contain sub-tours...



# The Traveling Salesman Problem

## IP Model\*\*

Devo scegliere uno dei due tipi di Vincoli per evitare di avere Sub-Tours

1

**Subtour Elimination Constraints (SEC):** Per ogni set di nodi, il numero di nodi visitati deve essere minore o uguale della cardinalità - 1

$$\sum_{(i,j) \in A[S]} x_{i,j} \leq |S| - 1 \quad \text{for each node subset } S \subset V$$

Impostare il Vincolo solo se ho bisogno (cioè solo se il sottoinsieme di nodi forma un ciclo, quindi devo impostare il vincolo per quel sottoinsieme)

2

Un nodo può avere un solo Numero di Sequenza (perché deve essere presente una sola volta nel Tour)

**Miller-Tucker-Zemlin Constraints (MTZ):**

$u$  = variabile di sequenza (indica il posizionamento di quel nodo nella sequenza del Tour)

Numero di Sequenza del  
Nodo Sorgente

Numero di Sequenza del  
Nodo Destinazione

$$u_i - u_j + 1 \leq |N| \cdot (1 - x_{i,j}) \quad \text{for } \{i, j\} \subset N; i, j \neq 1 \text{ (link variables)}$$

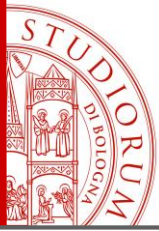
indica se l'arco che collega i due nodi (nodo sorgente e destinazione) è stato scelto per quel Tour

$$u_1 = 1 \quad \text{Nodo di Partenza deve avere numero di sequenza uguale ad 1}$$

$$u_i > 1 \quad \text{for } i \text{ in } N = \{1, 2, \dots\} \text{ (new node sequence variables)}$$

Gli altri nodi devono avere numero di sequenza maggiore di 1

**\*\*No sub-tours...** perché un nodo deve essere presente ESATTAMENTE una volta (a parte il Nodo di Partenza su cui il circuito si deve chiudere)



# The Traveling Salesman Problem

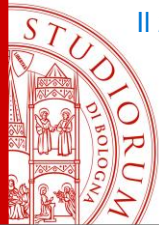
**Nearest-Neighbor Heuristic (NN) (can fail!)** Utile con Numero di Nodi Troppo Grande! (Metodo Approssimativo)

In pratica, é un Algo Greedy; vado a prendere ed aggiungere al tour il nodo che ha arco con peso più piccolo rispetto a quello da cui stiamo cercando il vicino (se non lo trova algoritmo fallisce)

1. Initialize empty tour T.
2. Pick arbitrary start node, add to tour T, and use as current node i.
3. Pick the node j that is
  - the closest to the current node i, and
  - not in T.Connect i to j, add j to T, and make j the current node.  
(If no node j was found, the heuristic fails.)
4. Go to 3 unless T contains all nodes.
5. Return tour T.

Posso applicare questo Algoritmo, sullo stesso Grafo, modificando il nodo di partenza per trovare, forse, percorsi ottimali rispetto a prima, ma non é detto che trovi l'ottimo.





Il 2-opt rimuove gli archi incrociati e li sostituisce con due archi che non si intersecano. Il risultato è un percorso più breve e più "ordinato".

# The Traveling Salesman Problem

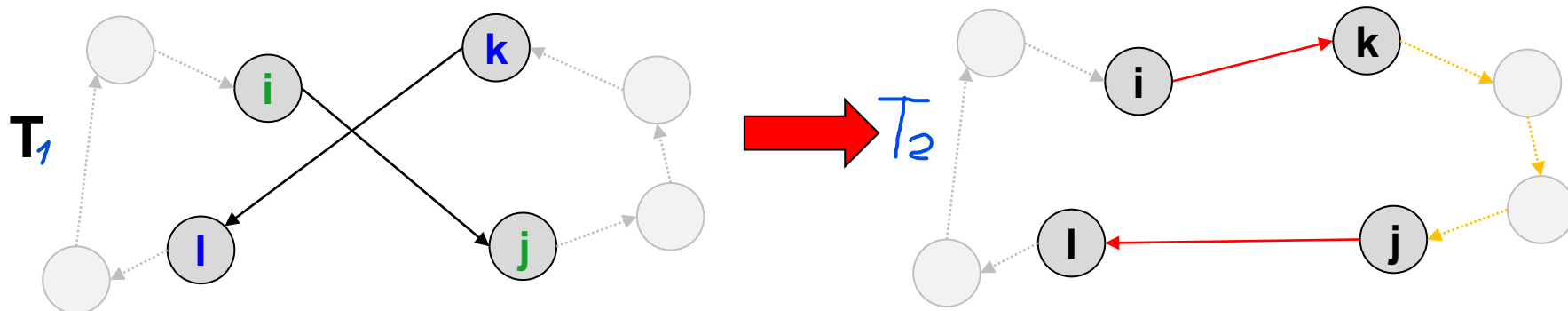
## Local Search: "2-Opt" (Metodo Euristico)

For a tour  $T$ . For two arcs  $(i,j)$  and  $(k,l)$ , let the first 2-opt move be defined as:

- Remove arcs  $(i,j)$  and  $(k,l)$  from  $T$ .
- Insert arcs  $(i,k)$  and  $(j,l)$  into  $T$ .
- Reverse path from  $j$  to  $k$  in  $T$ .

Per applicare il 2-opt, è necessario che nel grafo di base siano possibili i collegamenti tra i nodi  $i$  e  $k$  e tra  $l$  e  $j$ . Senza questi archi, la mossa del 2-opt non sarebbe valida o applicabile.

The second 2-opt move inserts  $(k,i)$  and  $(l,j)$ , and reverses path from  $l$  to  $i$ . (considerando il tour 1)



## Local Search Heuristic:

INPUT: Feasible tour  $T$ .

1. Find 2-opt move of highest improvement in  $T$ . Si cerca la mossa 2-Opt che produce il miglior miglioramento nella lunghezza del percorso (riduzione della distanza).
2. If improvement  $> 0$  then implement move for  $T$  and go to 1. Se il miglioramento è positivo (ovvero se la distanza totale è ridotta), viene applicata la mossa 2-Opt trovata.
3. Return tour  $T$ .

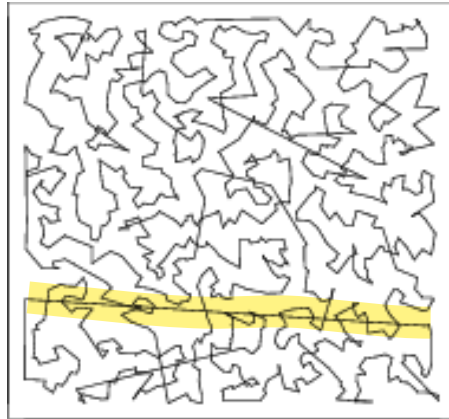


# The Traveling Salesman Problem

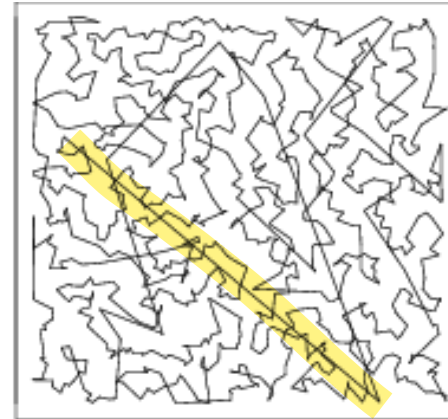
## Examples

Greedy Tour = mano mano che vado avanti, scelgo le connessioni minime possibili da applicare (Scelgo peso minimo e possibile da applicare (non per forza deve essere vicino a quello scelto per ultimo))

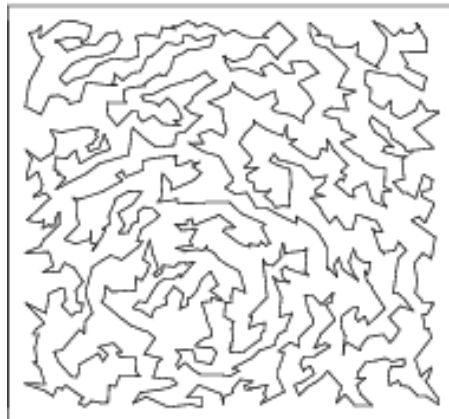
Esempio:  
Supponiamo di avere questi archi con i rispettivi costi:  
 $A \rightarrow B = 10$ ,  $A \rightarrow C = 15$ ,  
 $B \rightarrow C = 5$ ,  $B \rightarrow D = 20$ ,  
 $C \rightarrow D = 10$ ,  $A \rightarrow D = 25$   
Il Greedy inizia scegliendo l'arco con il costo più basso tra tutti gli archi disponibili:  $B \rightarrow C = 5$ .  
Successivamente, cerca il prossimo arco più economico, escludendo quelli che formerebbero cicli prematuri. Ad esempio, può scegliere  $A \rightarrow B = 10$ .  
Continua così finché non completa il tour, evitando alla fine però di chiudere cicli prematuramente o lasciare nodi isolati.



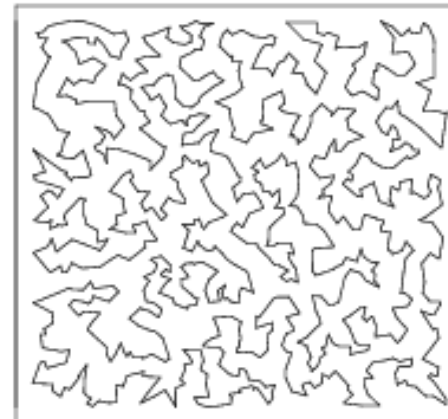
Greedy Tour



Nearest Neighbor Tour



Savings Tour



Optimal Tour

Sia NN sia Greedy Tour, non guarda il Grafo, su cui trovare il Tour minimo, nel suo insieme ; quindi, alla fine, scelgo, per collegare inizio con fine, un arco molto costoso (quello evidenziato)

IN PRATICA, APPLICO NN SU OGNI NODO COME NODO PARTENZA, APPLICO 2-OPT SU OGNI TOUR E POI SCELGO IL MIGLIORE



# The Traveling Salesman Problem

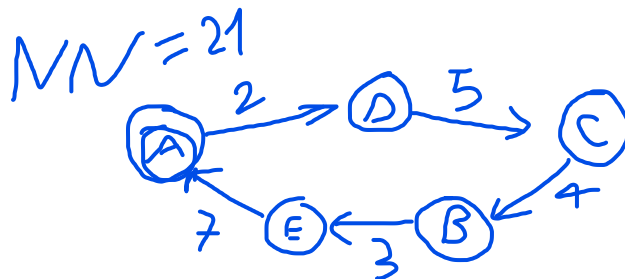
What is the shortest tour that visits all cities A, ..., E?

Travel Distances

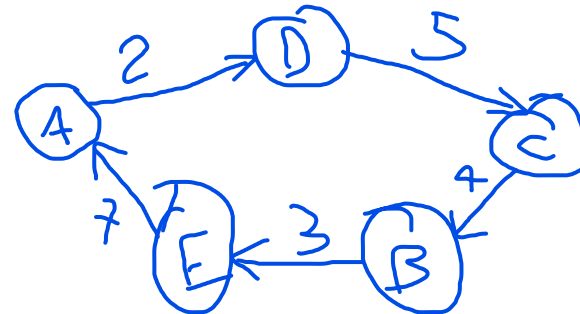
Cities	A	B	C	D	E
A	0	3	4	2	7
B	3	0	4	6	3
C	4	4	0	5	8
D	2	6	5	0	6
E	7	3	8	6	0

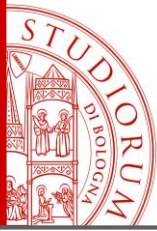
- 1) A  $\rightarrow$  D = 2 (arco più piccolo possibile)
- 2) B  $\rightarrow$  E = 3
- 3) C  $\rightarrow$  B = 4
- 4) D  $\rightarrow$  C = 5
- 5) E  $\rightarrow$  A = 7

Se cicli intermedi o non soluzione per chiudere il Tour, soluzione non valida



GREEDY = 21



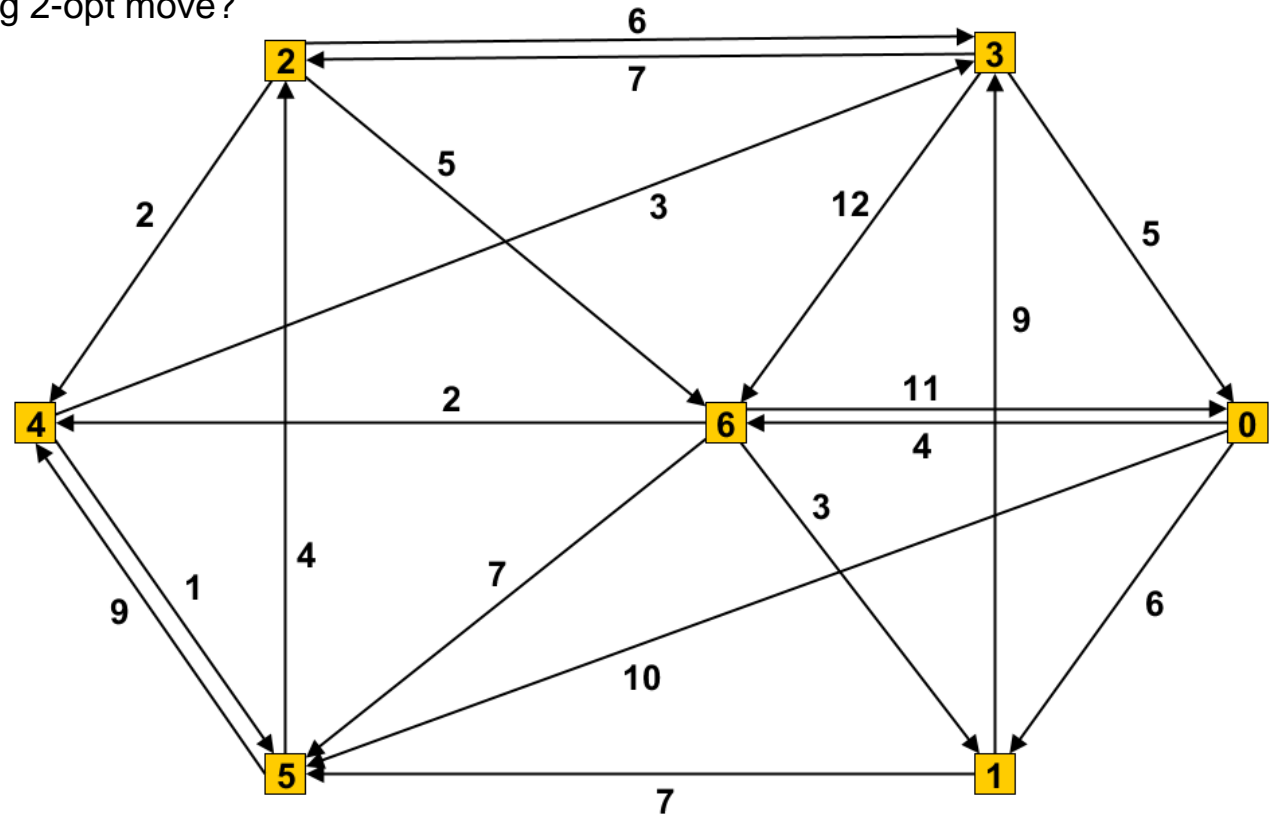


# The Traveling Salesman Problem

## 1) Find a TSP tour using the nearest-neighbor heuristic.

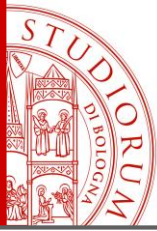
Start from **a)** node 1; **b)** node 4; **c)** node 5.

Can we find an improving 2-opt move?



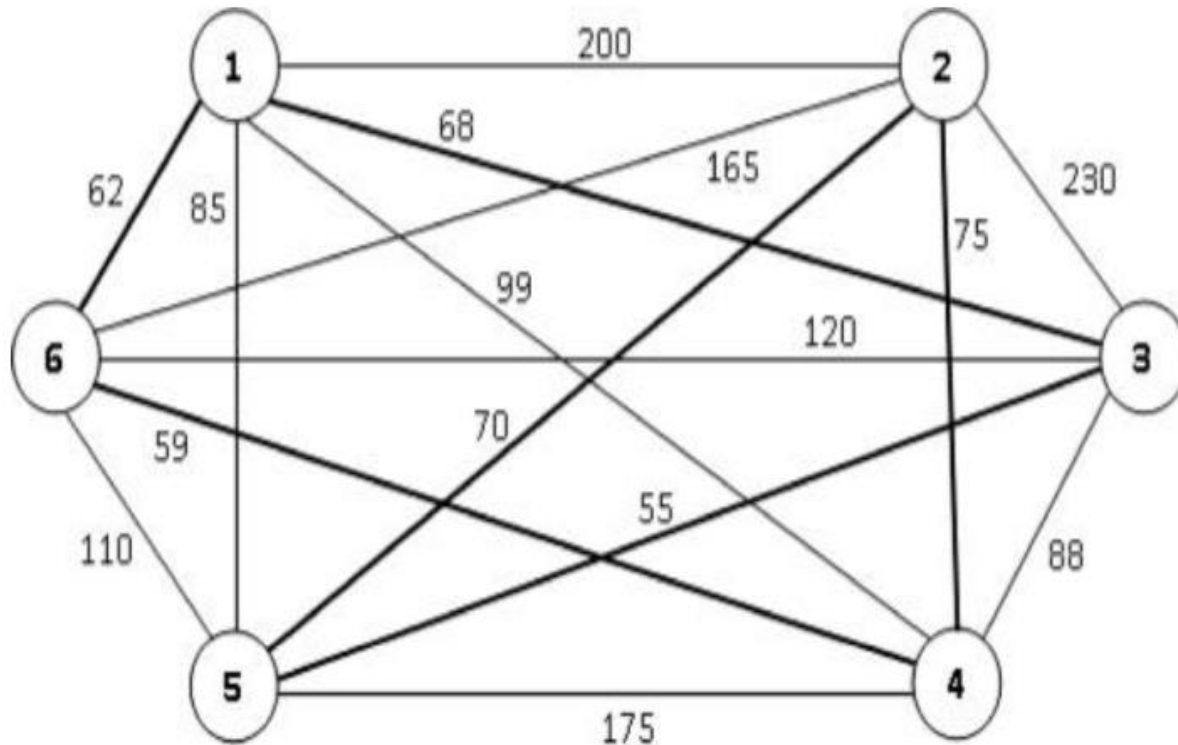
## 2) Solve the TSP above using IP.

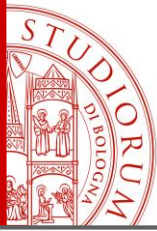




# The Traveling Salesman Problem

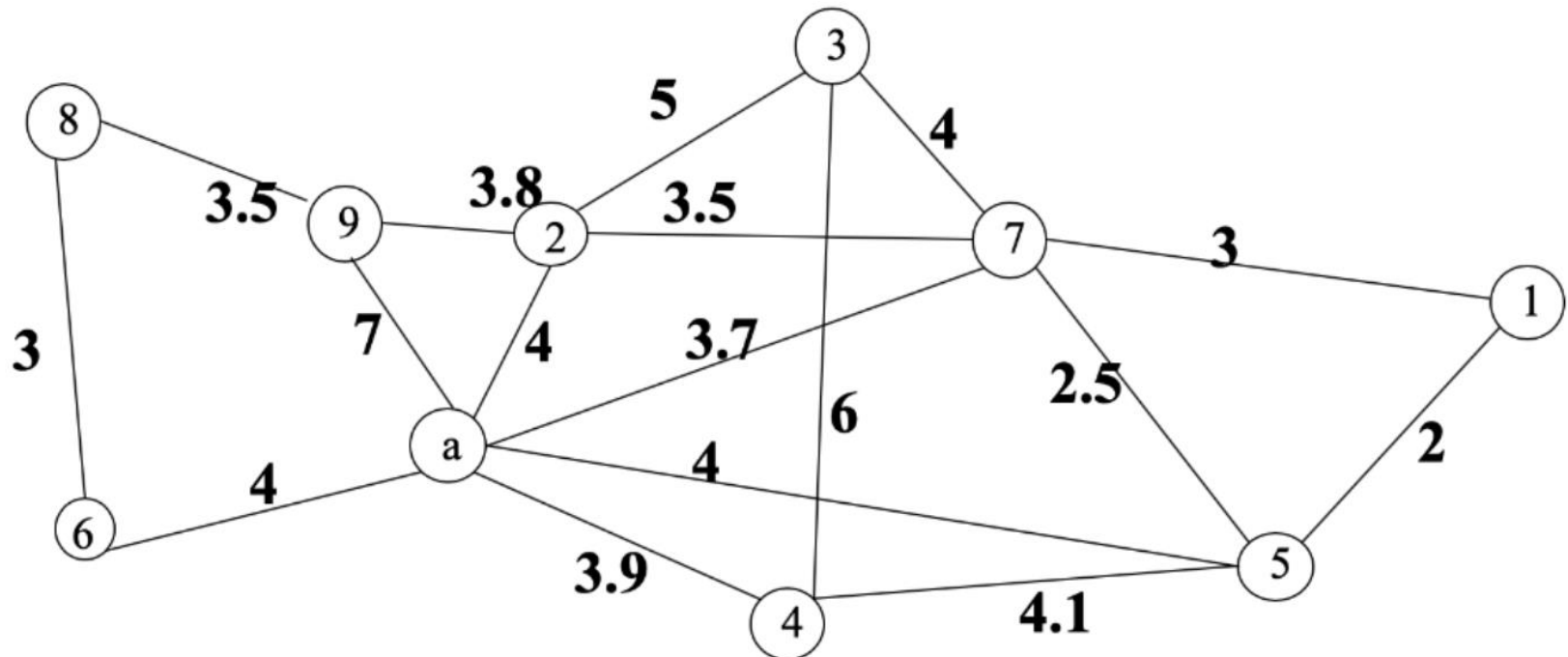
3) Find a TSP tour using the nearest-neighbor heuristic and compare your results to an IP solution.





# The Traveling Salesman Problem

4) Find a TSP tour using the nearest-neighbor heuristic and compare your results to an IP solution.





# The Traveling Salesman Problem

## 5) Group Exercise:

- 1. Find 5-10 points of interest that we would like to tour in Cesena.**  
For example, museums, shops, restaurants, bars, cinemas, parks.  
They could also all be located within a building.  
Other cities are of course also fine.
- 2. Estimate all the travel distances/durations.**  
Use an online map system such as Google Maps.  
Is there an advanced tool that returns a distance matrix?  
You can use walking, cycling, driving to do the tour.  
Enter your data in a spreadsheet.
- 3. Find a short tour using the NN-Heuristic.**  
Use different starting points if necessary.
- 4. Build and solve an IP model.**  
Include either SEC or MTZ inequalities.
- 5. Visualize your best tour using yEd.**  
Can you add a map snapshot as background?

