Local search and approximation algorithms

Lorenzo Castelli, Università degli Studi di Trieste.

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Mathematical optimisation 2021

Local search

Once an initial feasible solution, called the incumbent, has been found, it is natural to try to improve the solution. The idea of a local search heuristic is to define a neighbourhood of solutions close to the incumbent. Then the best solution in the neighbourhood is found. If it is better than the incumbent, it replaces it, and the procedure is repeated. Otherwise the incumbent is locally optimal with respect to the neighbourhood, and the heuristics terminates.

Knapsack - Local search

Starting from the greedy solution we now see a possible local search algorithm that examines each element in solution in sequence and, if possible and convenient, exchanges it with a subsequent element that is not in solution.

Let ***zg*** the value of the greedy solution and ***cu*** the remaining unused capacity.

Knapsack - Local search


```
begin
  call Greedy and get zg and cu
  z := zg
  for i := 1 to n do
    if x(i) = 1 then
        for j := i + 1 to n do
            if x(j) = 0 and cu + w(i) >= w(j) and p(j) > p(i) then
                x(i) := 0, x(j) := 1,
                z := z - p(i) + p(j), cu := cu + w(i) - w(j)
            end-if
        end-for
    end-if
  end-for
end
```

Knapsack - Local search

$$\begin{aligned} \max & \mathbf{100}x_1 + \mathbf{60}x_2 + \mathbf{70}x_3 + \mathbf{45}x_4 + \mathbf{45}x_5 + \mathbf{4}x_6 + \mathbf{4}x_7 + \mathbf{4}x_8 + \mathbf{15}x_9 \\ & \mathbf{10}x_1 + \mathbf{10}x_2 + \mathbf{12}x_3 + \mathbf{8}x_4 + \mathbf{8}x_5 + x_6 + x_7 + x_8 + \mathbf{4}x_9 \leq \mathbf{26} \\ & x_j \in \{\mathbf{0}, \mathbf{1}\} \text{ for } \boldsymbol{i} = \mathbf{1}, \ldots, \mathbf{9} \end{aligned}$$

The greedy solution is $(\mathbf{1}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{0})$, with $\boldsymbol{zg} = \mathbf{172}$ and $\boldsymbol{cu} = \mathbf{3}$.

For $\boldsymbol{i} = \mathbf{2}$ and $\boldsymbol{j} = \mathbf{3}$ an exchange can be made, such that we get

$$\boldsymbol{x} = (\mathbf{1}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{0}), \text{ with } \boldsymbol{z} = \mathbf{182} \text{ and } \boldsymbol{cu} = \mathbf{1}$$

No further exchanges are possible.

The optimal solution is $(\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0})$ with $\boldsymbol{z}^* = \mathbf{190}$

Knapsack - Local search

i	1	2	3	4	5	6	7	8	9
p	100	60	70	45	45	4	4	4	15
x	1	1	0	0	0	1	1	1	0
w	10	10	12	8	8	1	1	1	4

z = 172, cu = 3

We see that cu+w(2) ≥ w(3) and p(3) > p(2), i.e., 3+10 ≥ 12 and 70 > 60, then the exchange can take place

i	1	2	3	4	5	6	7	8	9
p	100	60	70	45	45	4	4	4	15
x	1	0	1	0	0	1	1	1	0
w	10	10	12	8	8	1	1	1	4

z = 182, cu = 1

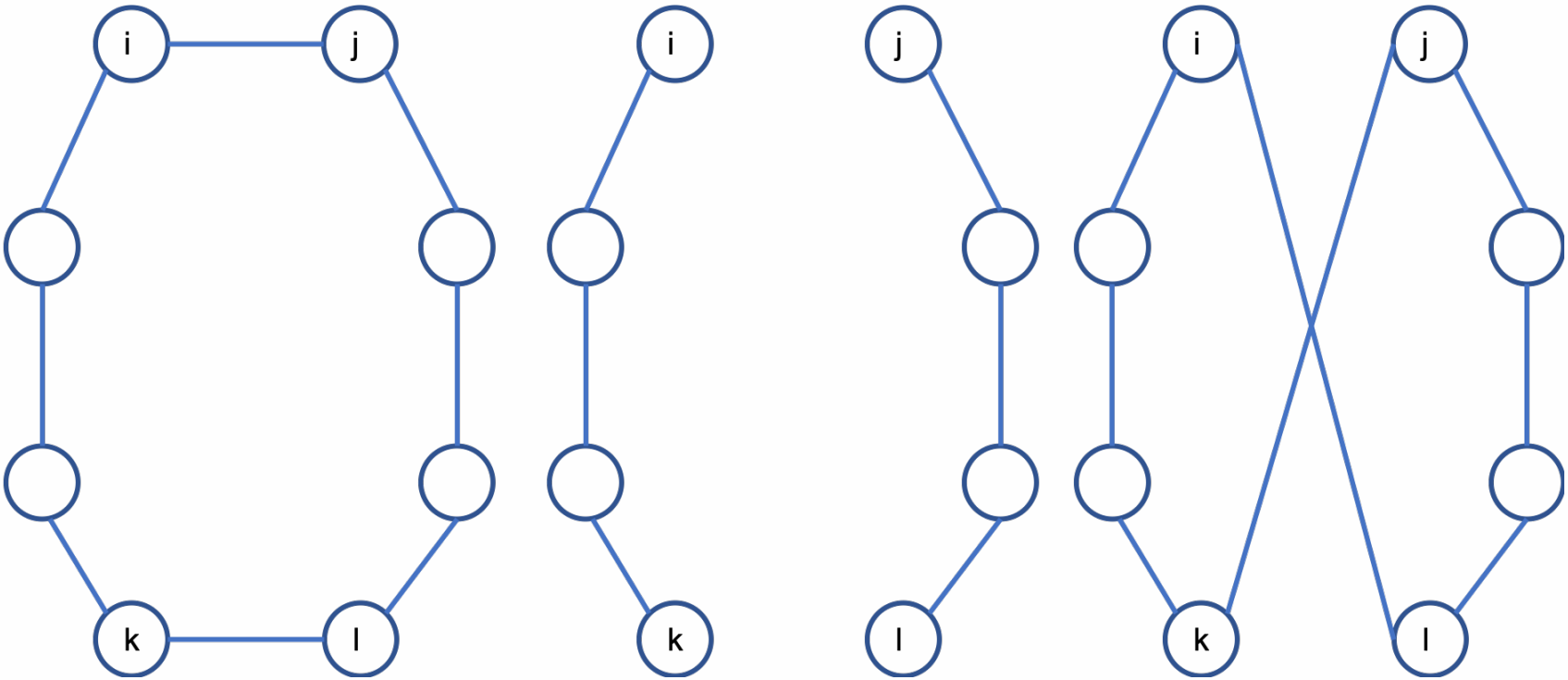
The linear relaxation gives $(\mathbf{1, 1, 0.5, 0, 0, 0, 0, 0, 0})$ with $\boldsymbol{zl = 195}$. Hence before starting the B&B algorithm, we know that $\mathbf{182 \leq z^* \leq 195}$.

TSP - Local search: $k - opt$

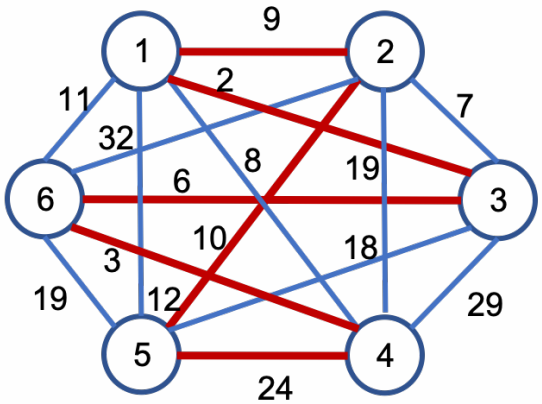
At each iteration k arcs are removed from the current tour and the resulting k paths are connected with other arcs (if possible) so as to form a new tour of lower cost. For $k = 2$:

procedure Two-Opt
begin
let C be the initial tour
while there are $(i,j),(k,l) \in C : c_{ij} + c_{kl} > c_{il} + c_{kj}$ **do**
$C := C \backslash \{(i,j),(k,l)\} \bigcup \{(i,l),(k,j)\}$
end-while
end

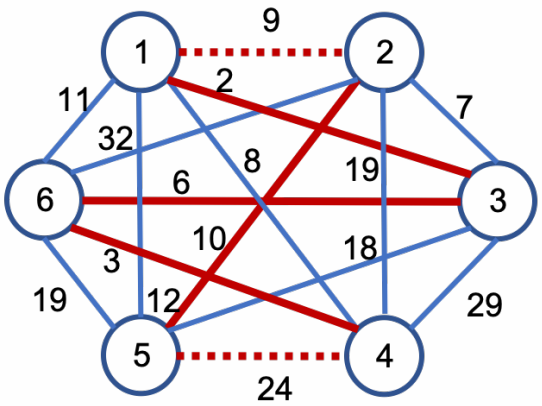
TSP - 2 – *opt*



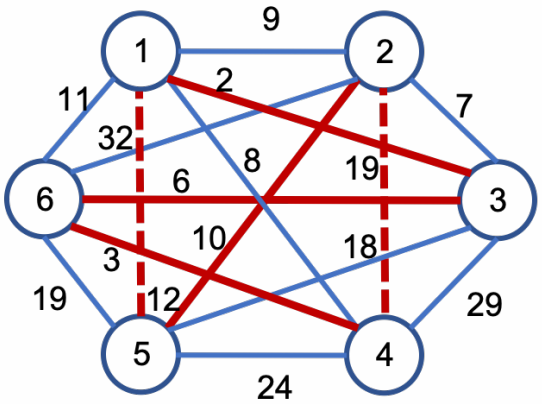
TSP - 2 – *opt*



1-3-6-4-5-2-1. **Cost: 54**



Remove 2-1 and 4-5. Cost: 32



Add 4-2 and 5-1. Cost: 30

New tour: 1-3-6-4-2-5-1. **Cost 52**

Metaheuristics

Local search algorithms terminate the execution when no move can improve the current solution. The basic rule of accepting only convenient moves can however lead to lose the possibility of improvement.
In the knapsack example, the local search algorithm terminates in

$$\boldsymbol{x} = (\mathbf{1}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{0}), \text{ with } \boldsymbol{z} = \mathbf{182} \text{ and } \boldsymbol{cu} = \mathbf{1},$$

as no move can improve it.
Suppose to accept the inconvenient exchange between $\boldsymbol{x_3}$ and $\boldsymbol{x_9}$. We get

$$\boldsymbol{x} = (\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1}), \text{ with } \boldsymbol{z} = \mathbf{127} \text{ and } \boldsymbol{cu} = \mathbf{9}.$$

At the next iteration it is possible to exchange $\boldsymbol{x_6}$ and $\boldsymbol{x_2}$ and get

$$\boldsymbol{x} = (\mathbf{1}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{1}), \text{ with } \boldsymbol{z} = \mathbf{183} \text{ and } \boldsymbol{cu} = \mathbf{0}.$$

Metaheuristics

A local search algorithm stops when it determines a local maximum of the objective function, and a possible way to avoid this is to modify it in such a way that it can accept, in certain situations, moves that **worsen** the value of the current solution.
This is the main feature of metaheuristic algorithms, today's most widespread tool for the approximate solution of optimisation problems.
The simplest way is to introduce elements of randomisation in the construction of the solutions and / or in the determination of the moves.

Metaheuristics

The main metaheuristic approaches are:

- Taboo search
- Simulated annealing
- Genetic algorithms
- Ant colony optimisation
- Swarm particle optimisation
- Variable neighbourhood search

Approximation algorithm

Definition

A polynomial algorithm $\mathcal{A}$ is said to be a δ-approximation algorithm if for every problem instance I with an optimal solution $OPT(I)$

$$R_{A(I)} = \frac{A(I)}{OPT(I)} \leq \delta.$$

$\delta \geq 1$ for minimisation problems and ≤ 1 for maximisation problems.
The smallest value of δ is the approximation ratio $R_{\mathcal{A}}$ of the algorithm $\mathcal{A}$.
For maximisation problems, sometimes $\frac{1}{\delta}$ is considered to be the approximation ratio.

Approximation algorithms for the TSP

We present two approximation algorithms for the TSP

- MST approximation: $R_{\mathcal{A}} = 2$
- Christofides' algorithm: $R_{\mathcal{A}} = \frac{3}{2}$

Assumption

The triangle inequality always holds

$$c_{ik} \leq c_{ij} + c_{jk} \quad \forall i, j, k \in \{1, 2, \ldots, n\}$$

In other words, the shortest distance between two points (cities) is the straight line (direct route)

MST vs TSP

The MST provides a *lower* bound on the optimal tour length: deleting any arc from a tour yields a spanning tree consisting of a single path through all the cities.
Thus the optimal travelling salesman tour must be strictly longer than the minimum spanning tree.

$$\boldsymbol{MST}(\boldsymbol{I}) < \boldsymbol{OPT}(\boldsymbol{I})$$

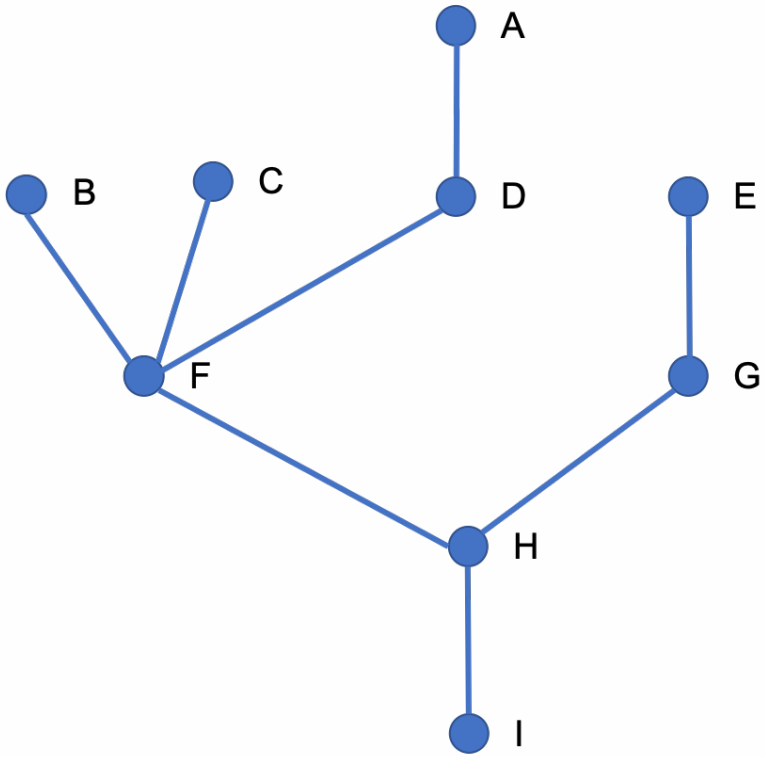
We see now that the triangle inequality allows us to use the MST to obtain an *upper* bound of the optimal tour length.

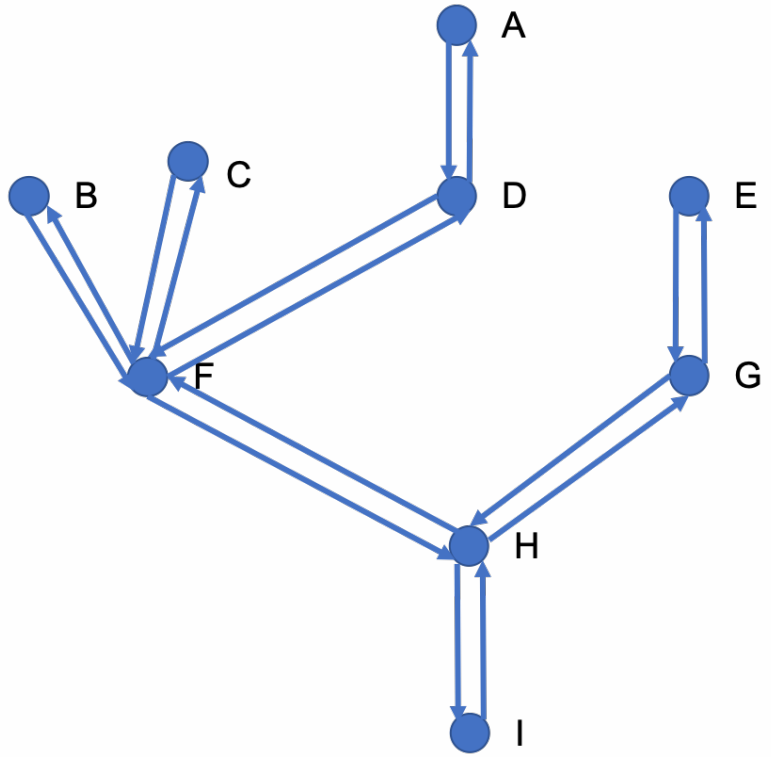
Using the MST

- We visit all the cities, but are only allowed to use arcs of the MST
- We start at a 'leaf' of the tree (vertex of degree 1) and apply the following strategy
  - If there is any untraversed arc leaving the current vertex, follow that arc to a new vertex.
  - If all the arcs from the current vertex have been traversed, go back along the arc by which the current vertex was first reached to the vertex from which it was visited.
  - Halt when we eventually return to our starting vertex.

The procedure is called the depth first traversal of the MST

Depth first traversal of the MST





Depth first traversal of the MST

- We visit all the cities.
- We traverse no arcs of the MST more than twice.
- Thus the route for visiting all the cities has length no more than twice that of the MST, and since $\boldsymbol{MST}(\boldsymbol{I}) < \boldsymbol{OPT}(\boldsymbol{I})$ at most twice the length of the optimum travelling salesman tour.

The only thing that prevents this traversal from being a travelling salesmen tour is the fact that it may visit some cities more than once.

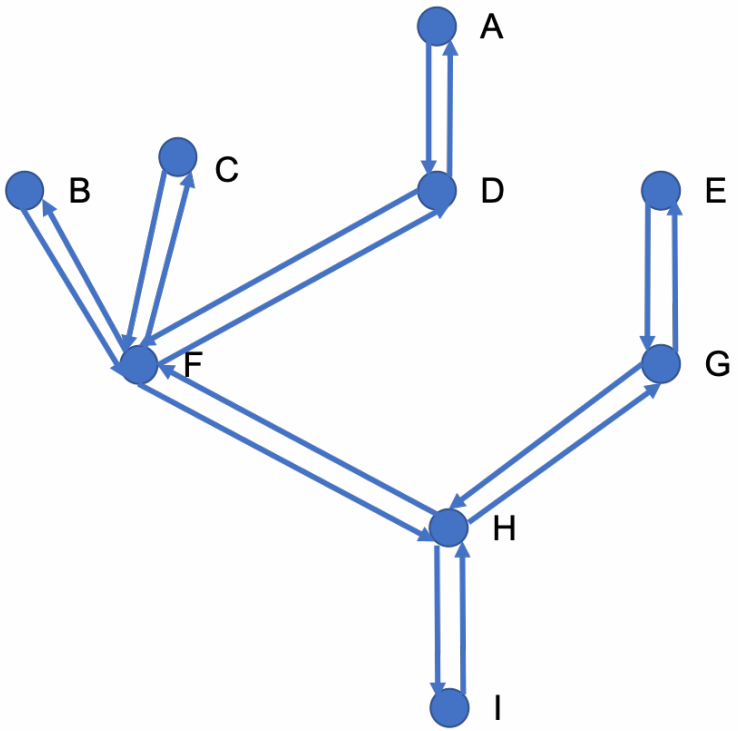
Introducing shortcuts

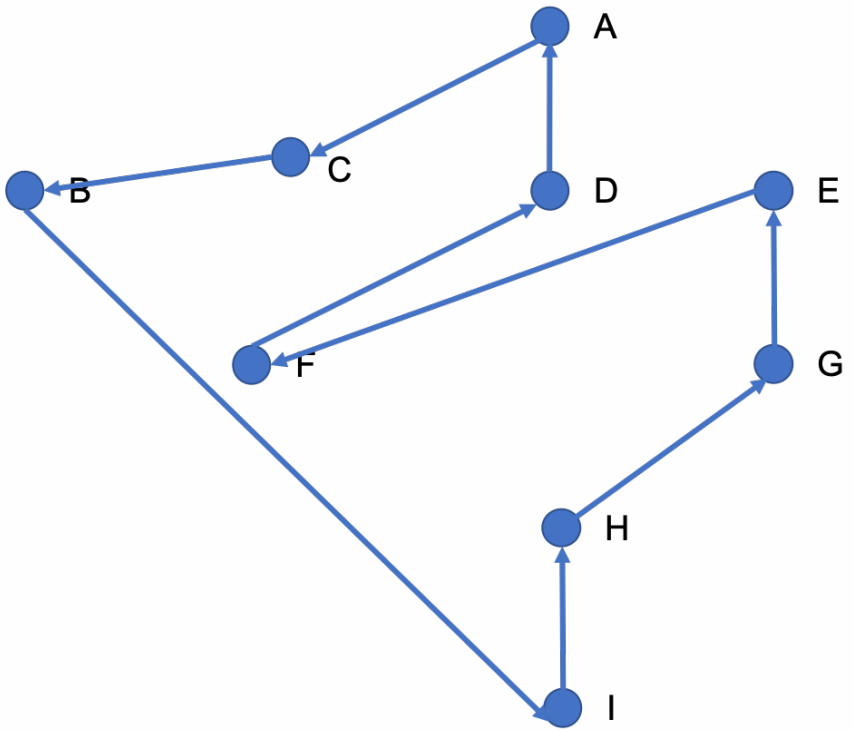
Thanks to the triangle inequality, we can avoid repeated cities by introducing 'shortcuts' that do not increase the total length of the traversal

- Start, as before, at a leaf of the MST.
- Whenever the depth first traversal would lead us back to an already-visited city, skip ahead in the traversal and go directly to the next as-yet-unvisited city.
- The direct route can be no longer than the previous indirect one
- If all cities have been visited go back to the starting point.

We have now constructed an actual tour (it visit no city except the starting point more than once) and because its length is no more than that of the original depth first traversal, this tour has length at most twice that of the optimal tour.

Shortcuts in the double MST





The minimum spanning tree algorithm

This procedure is called the minimum spanning tree algorithm.

Theorem

For all TSP instances I that obey the triangle inequality, if $MSTA(I)$ is the length of the tour constructed by the minimum spanning tree algorithm applied to I, then

$$MSTA(I) \leq 2OPT(I)$$

Eulerian graphs and tours

- An Eulerian graph is a connected graph in which every vertex has even degree.
- Eulerian graphs are those graphs that contain an Eulerian tour, i.e., a cycle that passes through every arc exactly once.

The minimum spanning tree algorithm can be reinterpreted in these terms

- We start with a MST, double its edge to obtain an Eulerian graph
- We find an Eulerian tour for this graph and the convert it to a Hamiltonian tour by using shortcuts.
- By the triangle inequality the Hamiltonian tour can be no longer than the Eulerian tour and hence at most twice the length of the MST

If we find better (shorter) Eulerian graphs connecting the cities, we can have better (shorter) Hamiltonian tours.

Matching

Definition

Given a set containing an even number of cities, a matching is a collection of arcs M such that each city is the endpoint of exactly one arc in M.

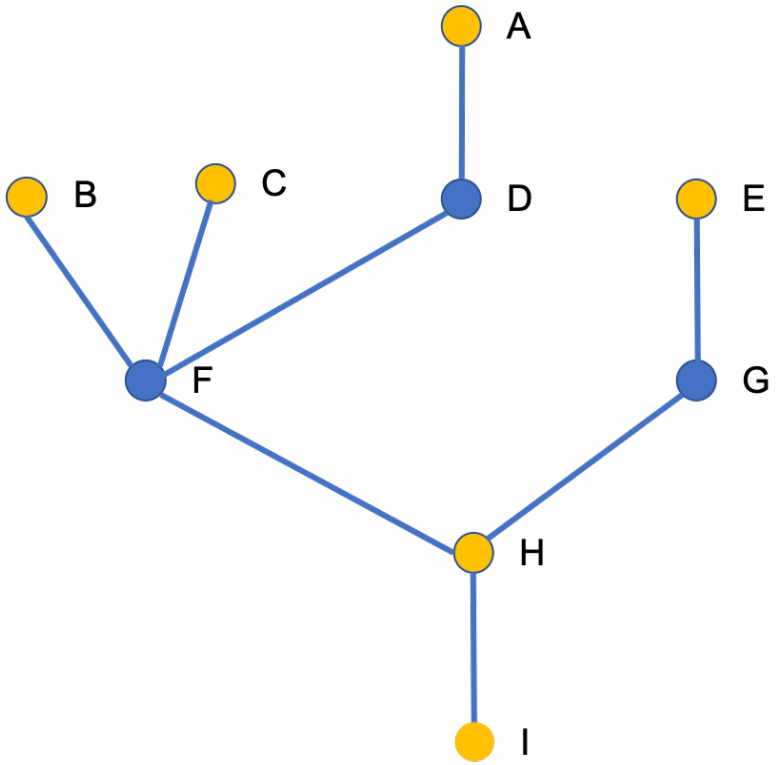
Definition

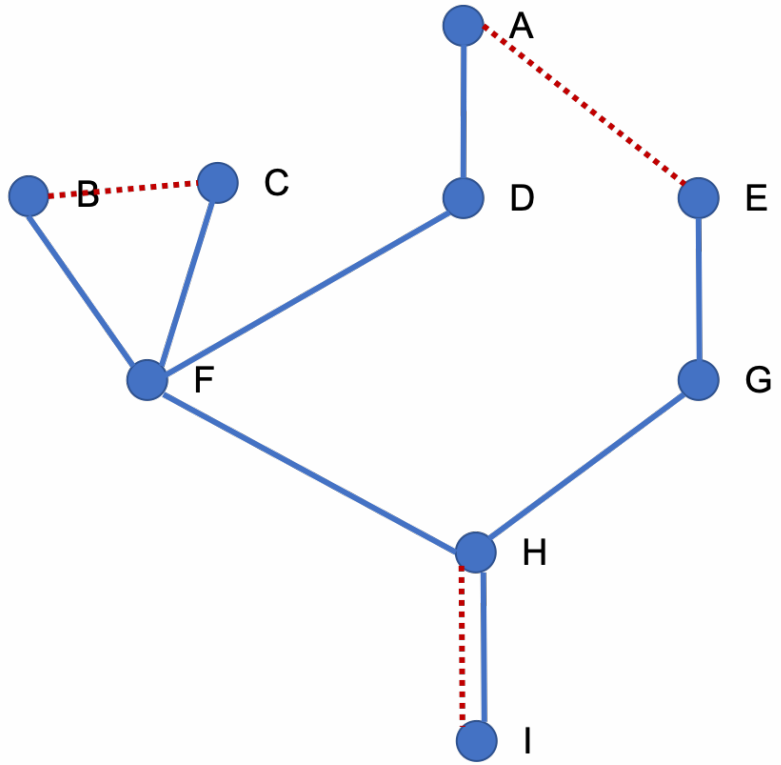
A minimum weight matching is one for which the total length of the arcs is minimum.

Matching for Eulerian graphs

- Consider the minimum spanning tree $\boldsymbol{T}$ for a TSP instance
- Certain of the vertices in $\boldsymbol{T}$ already have even degree (they do not need to receive more arcs if we want to turn the tree in an Eulerian graph)
- There might be some vertices with odd degree
- There must be an even number of these odd-degree vertices, since the sum of all vertex degrees must be even
- Thus by adding a matching for the odd-degree vertices in $\boldsymbol{T}$, we construct an Eulerian graph that includes $\boldsymbol{T}$
- If we add to $\boldsymbol{T}$ a minimum weight matching for its odd-degree vertices, we obtain an Eulerian graph that has minimum length among those that contain $\boldsymbol{T}$

Another Eulerian graph





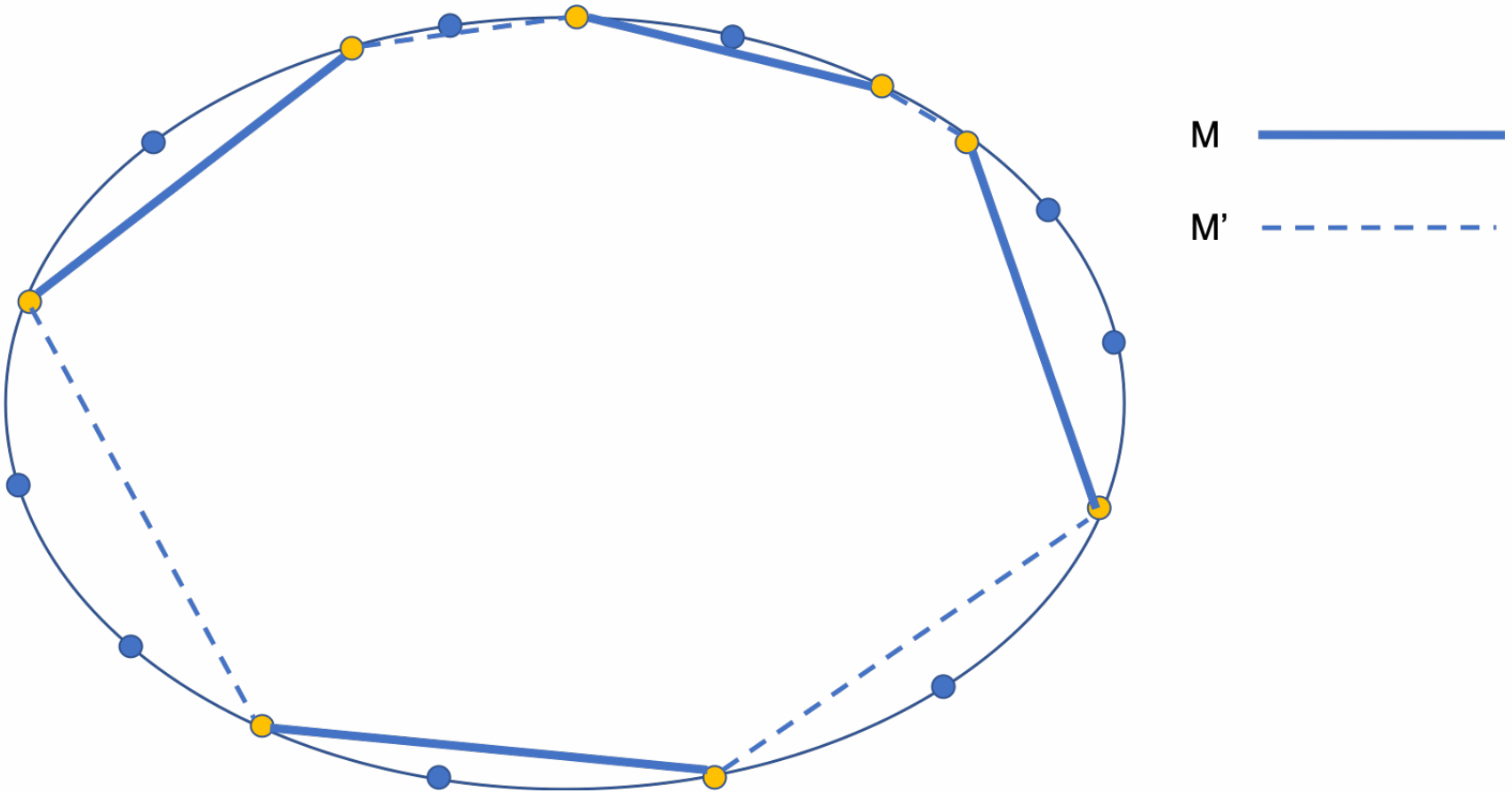
Length of the minimum weight matching

- Consider an Hamiltonian tour, with the cities that correspond to odd-degree vertices in $\boldsymbol{T}$ emphasised
- The tour determines two matchings $\boldsymbol{M}$ and $\boldsymbol{M'}$
- If $\boldsymbol{I}$ denotes the given TSP instance, by the triangle inequality we must have

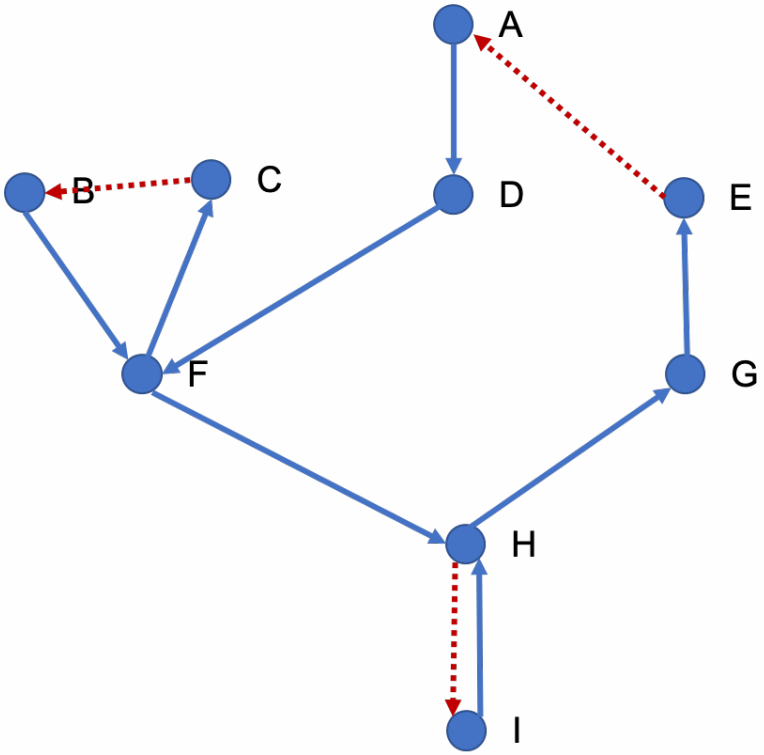
$$\boldsymbol{LENGTH}(\boldsymbol{M}) + \boldsymbol{LENGTH}(\boldsymbol{M'}) \leq \boldsymbol{OPT}(\boldsymbol{I})$$

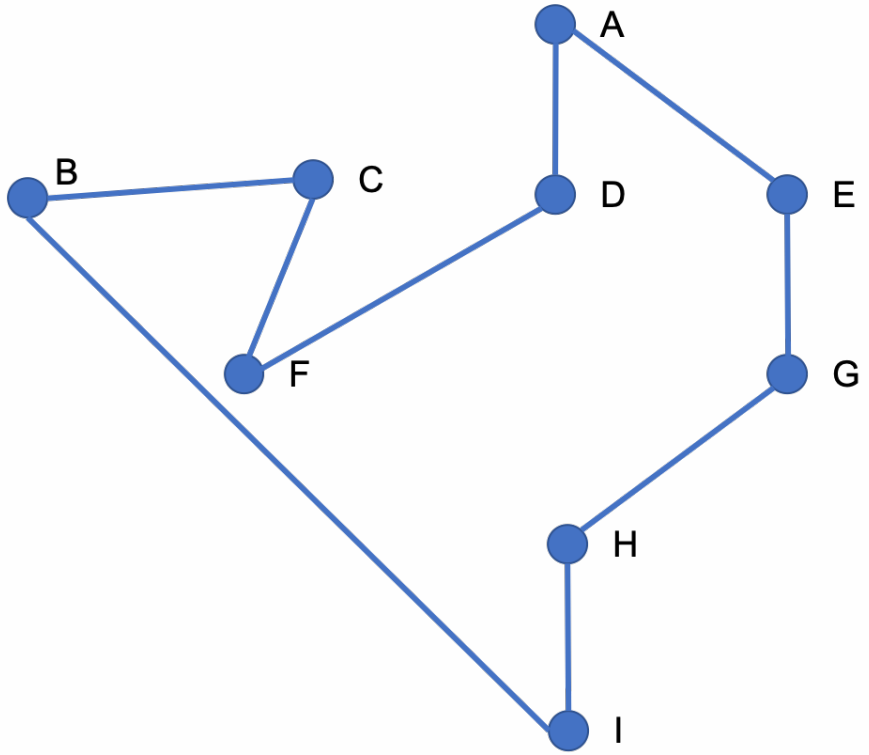
- Hence one of $\boldsymbol{M}$ and $\boldsymbol{M'}$ must have length less than or equal to $\boldsymbol{OPT}(\boldsymbol{I})/\boldsymbol{2}$.
- Thus the length of a minimum weight matching (let it be matching $\boldsymbol{M}$) for the odd-degree vertices of $\boldsymbol{T}$ must also be at most $\boldsymbol{OPT}(\boldsymbol{I})/\boldsymbol{2}$

TSP and Matching



Christophides' algorithm





Length of the Hamiltonian tour

Since $\boldsymbol{LENGTH}(\boldsymbol{T}) < \boldsymbol{OPT}(\boldsymbol{I})$ and shortcuts can be applied to the Eulerian tour to transform it into a Hamiltonian tour, if $\boldsymbol{C}(\boldsymbol{I})$ is the length of the tour constructed according to this algorithm, called Christofides' algorithm, we have that

$$\boldsymbol{C}(\boldsymbol{I}) \leq \boldsymbol{LENGTH}(\boldsymbol{T}) + \boldsymbol{LENGTH}(\boldsymbol{M}) < \frac{\boldsymbol{3}}{\boldsymbol{2}}\boldsymbol{OPT}(\boldsymbol{I}).$$