# Algorithms and Data Structures
# Module 4. NP-hard problems

## Lecture 20
## **Inexact algorithms. Part 1.**

# TSP: solving (lecture 18)

Possible options for solving any NP-hard problem (e.g. TSP):

- Exactly but inefficiently:
  - ✓exhaustive search (brute-force, backtracking)
  - ✓smart search (branch-and-bound)
- Exactly, efficiently, but not universally:
  - ✓efficiently solvable special cases.
- Efficiently but inexactly:
  - ✓approximate algorithms,
  - ✓heuristics

# Approximate algorithms

Let $x$ be a problem instance and $opt(x)$ denote an optimal solution for $x$. An *approximate* algorithm $A$ is an algorithm that returns a feasible solution $A(x)$ for any given $x$.

- We are interested in *polynomial-time* approximate algorithms only.
- We need a measure of quality to compare approximate algorithms to each other.

*Performance* (*approximation*) *ratio* for a minimization problem:

$$R(A) = \sup \left\{ \frac{c(A(x))}{c(opt(x))} : x \in X \right\}$$

# Approximate algorithms

$R(A)$ can be a number (>1), a function of $|x|$ or $+\infty$.

**Definition**

An algorithm with $R(A) = r = const$ is called an **$r$ −approximate** algorithm.

**Definition**

An inexact algorithm which has no performance guarantee ($R(A) = +\infty$) is called a ***heuristics***.

The quality (performance) of heuristics should be estimated in practical computational experiments.

# 2-approximate algorithm for MTSP

Let us consider a 2-approximate algorithm for Metric TSP.

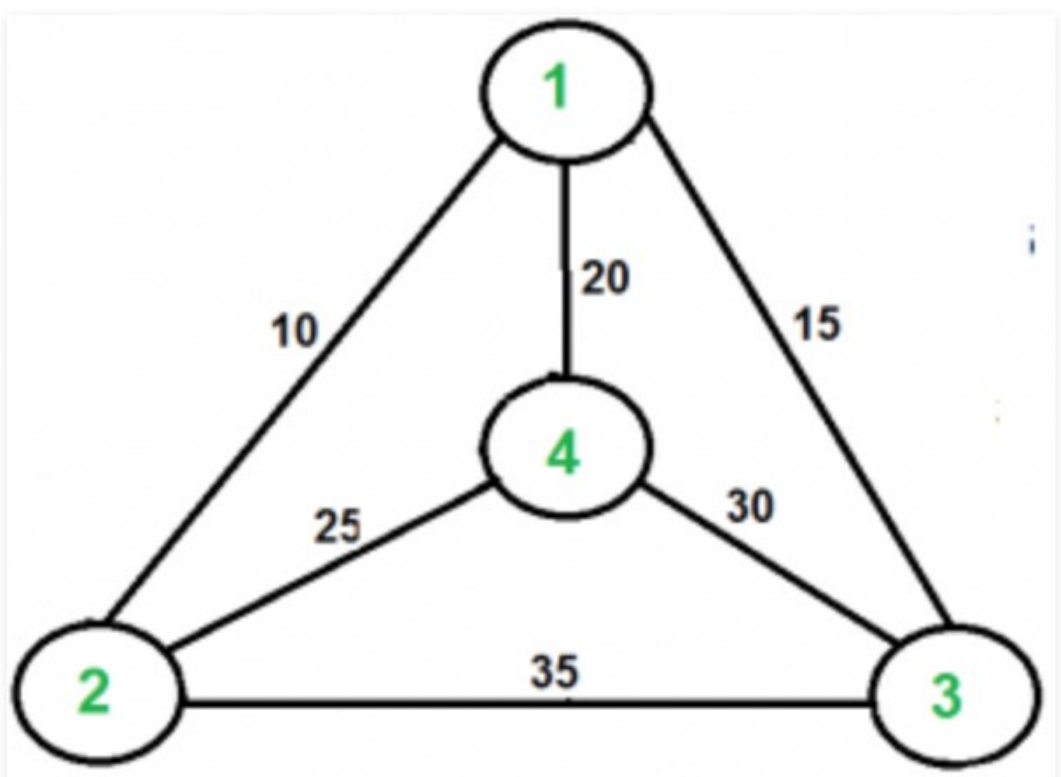
Let $G(V, E)$ be a connected undirected graph with metric weight function $w: E \longrightarrow R_+$.

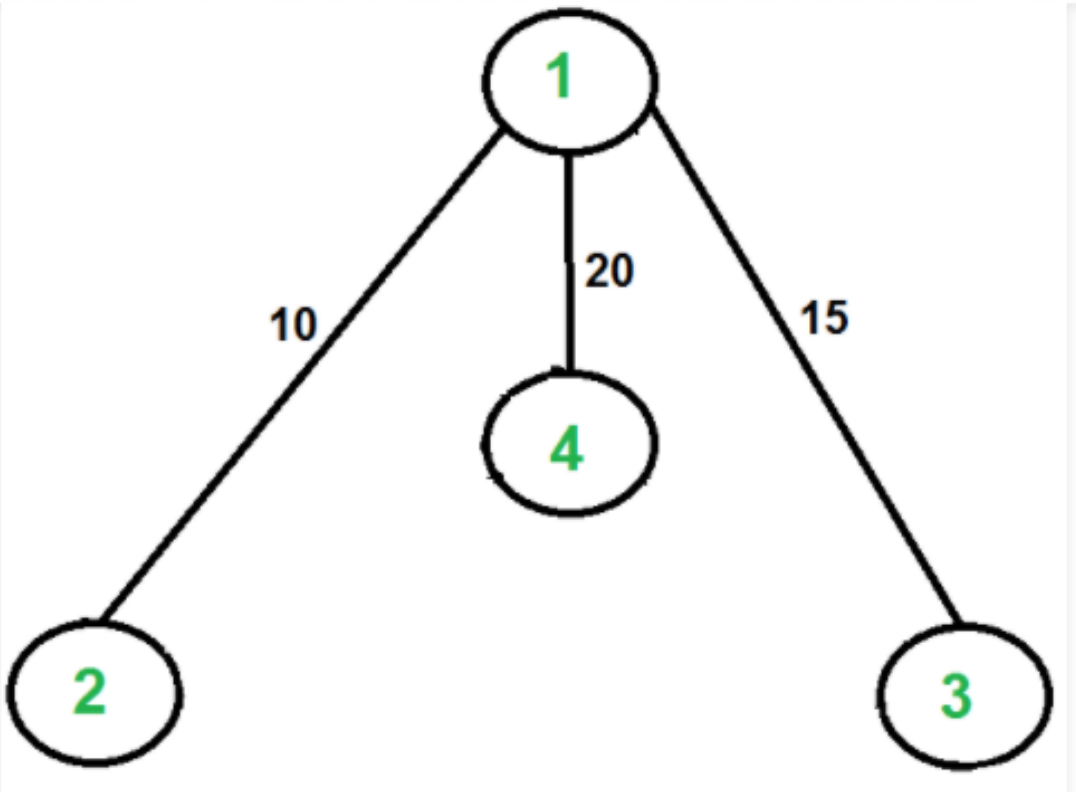
Let $Z^*$ be an optimal Hamiltonian cycle. Let us build a Hamiltonian path $P$ by removing an arbitrary edge from $Z^*$. It is obvious that $w(Z^*) \geq w(P)$.

Let $T$ be an MST (minimum spanning tree) for $G$. It is obvious that $w(P) \geq w(T)$.

Let us traverse $T$ in the preorder (depth-first) order. Let $W$ be the resulting walk. Thus, $w(W) = 2 \cdot w(T)$ => $w(W) \leq 2 \cdot w(P) \leq 2 \cdot w(Z^*)$.

# 2-approximate algorithm for MTSP




*W*=1,2,1,4,1,3,1

# 2-approximate algorithm for MTSP

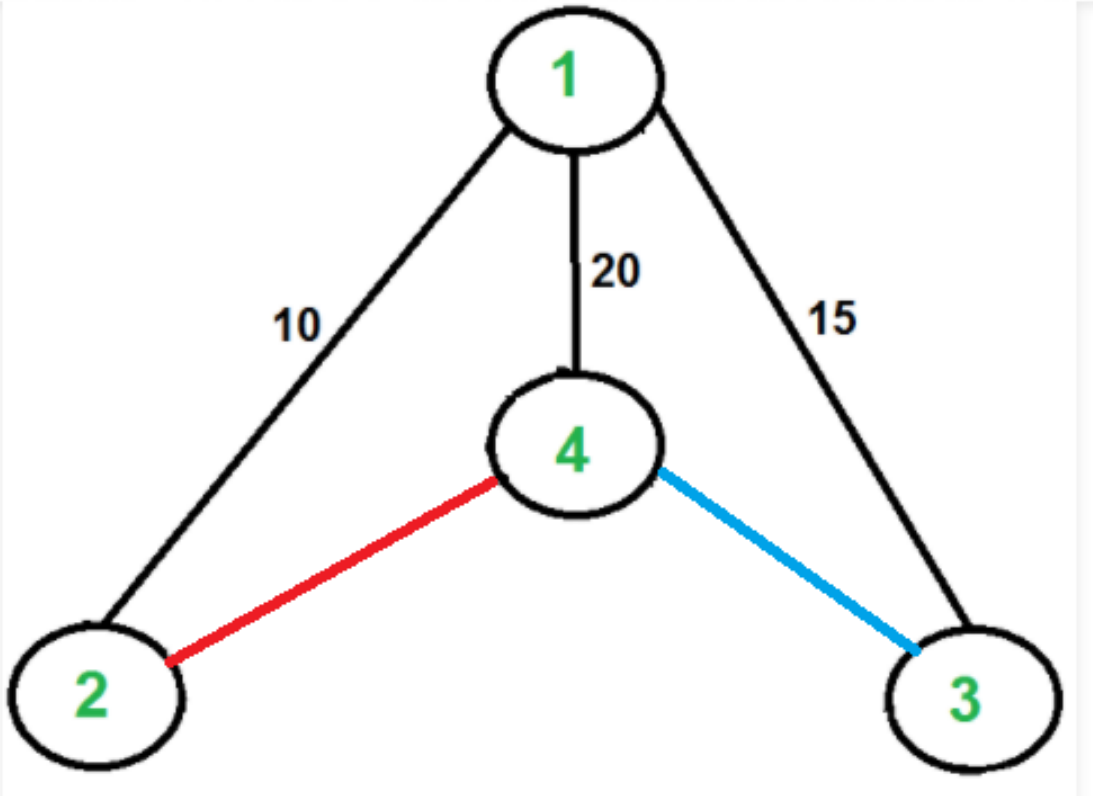
*W*=1,2,1,4,1,3,1

Let us build a cycle by removing duplicates from *W*:

$$Z = 1,2,4,3,1$$

Due to the triangle inequality,
$w(Z) \leq w(W) \leq 2 \cdot w(Z^*)$

Thus, we have designed for MTSP a 2-approximate algorithm with $O(m \log n)$ time complexity.



# 2-approximate algorithm for MTSP

For MTSP there is a better algorithm. Christofides algorithm is can be viewed as a modification of the above-discussed algorithm; it has approximation ratio 1.5.

It is proven that for the general TSP there cannot exist an approximate algorithm with constant performance ratio. (Of course, this statement was proven with presupposition that $P \neq NP$.)

Thus, the only way to solve the general TSP is with ***heuristics***.

# Heuristics for TSP

There are many ideas/principles that can be used for designing a heuristics algorithm.

We will study just two of them:

- Greedy heuristics.
- Local search.

# Greedy algorithms (lecture 8)

Key characteristics of a greedy algorithm:

- Can solve an optimization problem.
- Builds solution iteratively, adding one element after another.
- At each step, adds the element which is the best at the current situation.
- Does not revise the decisions (one-pass algorithm).

# Greedy heuristics for TSP

**1. Nearest neighbor.**

Build the tour (Hamiltonian cycle) iteratively, moving to unvisited vertex which is the nearest to the current vertex.

1. Select the starting vertex, either randomly or by rule. Assign this vertex to the current vertex. Label the current vertex as visited.
2. From the current vertex move to the nearest unvisited vertex.
3. Label the current vertex as visited.
4. If all vertices are visited, move to the starting vertex, building a cycle. Otherwise, go to 2.

Time complexity: $O(n^2)$.

Usual quality of approximate solution: 25% greater than the optimal ($R$=1.25).

# Greedy heuristics for TSP

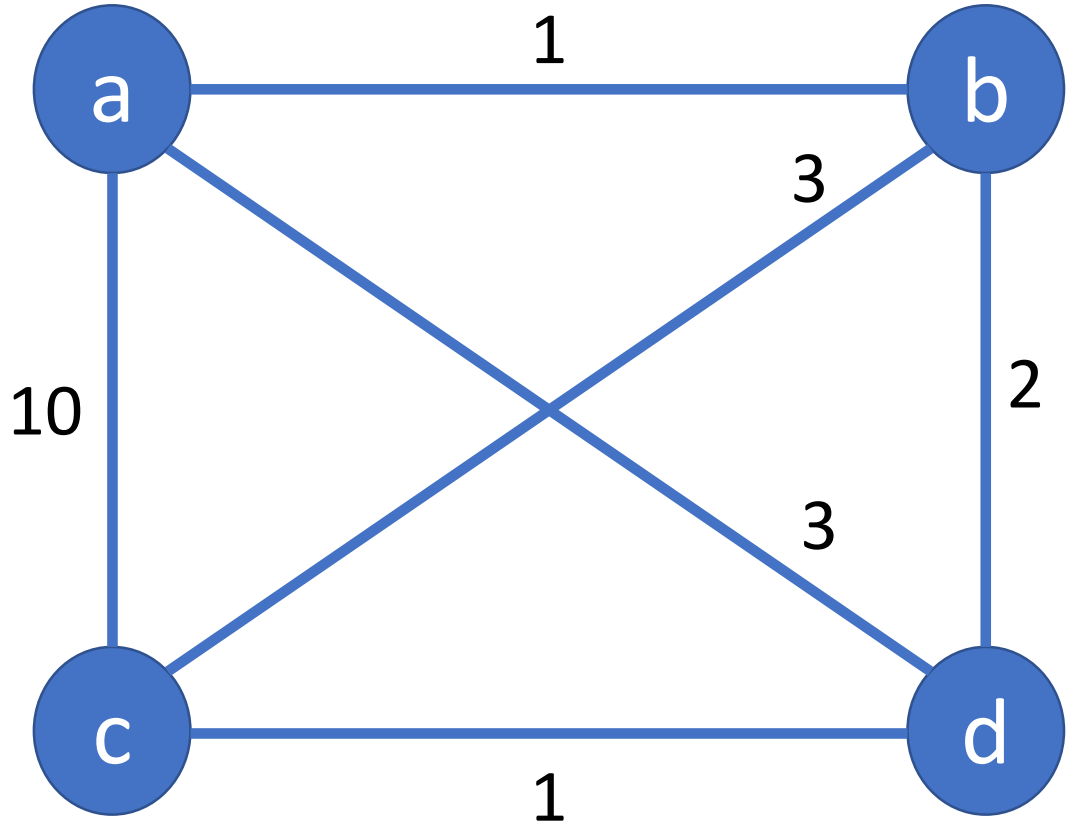
**1. Nearest neighbor.**

Nearest neighbor solution:

a,b,d,c,a: weight 14

Optimal solution:

a,b,c,d,a: weight 8



# Greedy heuristics for TSP

## 2. Lightest edge selection.

Build the tour (Hamiltonian cycle) iteratively, at each iteration adding the lightest admissible edge. An edge is admissible iff adding this edge to the current partial solution does not produce:

- a loop of size less than $n$;
- vertex degree greater than 2.

Algorithm:

1. Sort edges by weights in ascending order.
2. Process edges in this order.
3. For each edge, check its admissibility. If the edge is admissible, add it to the current partial solution.

Time complexity: $O(n^2 \log n)$.

Usual quality of approximate solution: 15-20% greater than the optimal.

# Greedy heuristics for TSP

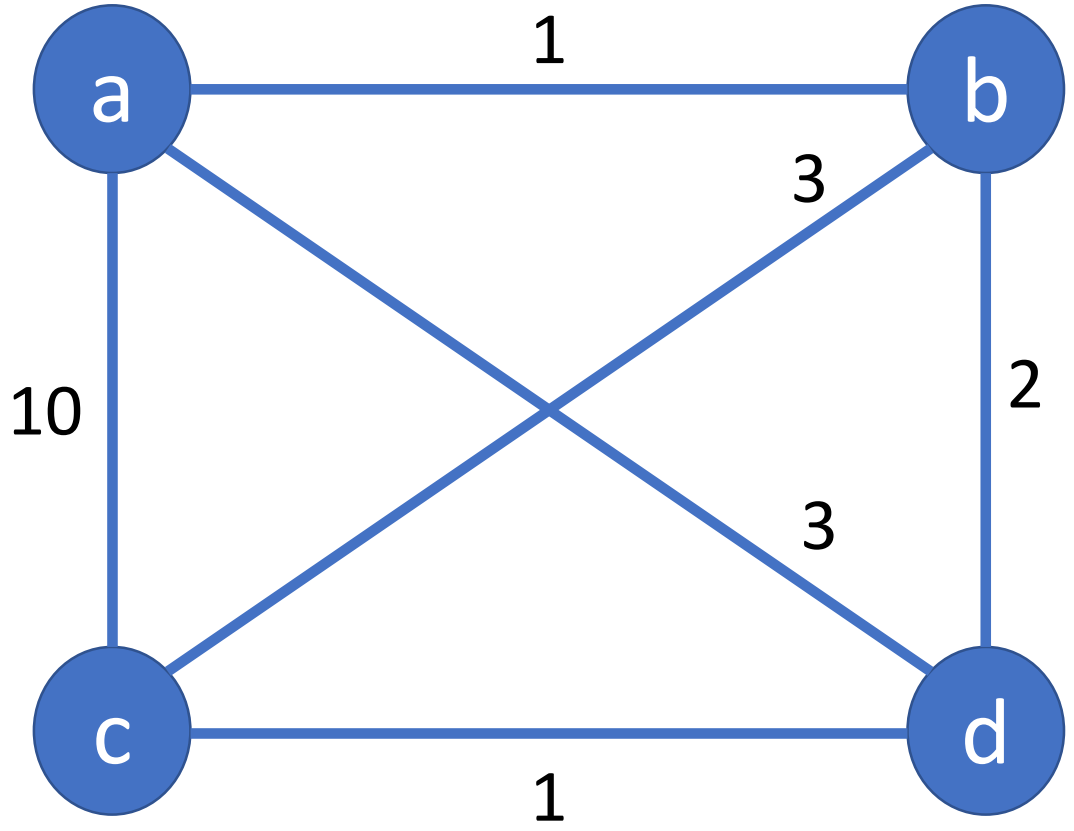
**2. Lightest edge selection.**

Heuristic solution:

a,b,d,c,a: weight 14

Optimal solution:

a,b,c,d,a: weight 8



# Greedy heuristics for TSP

**3. Greedy vertex insertion.**

Build the tour (Hamiltonian cycle) by iteratively inserting a vertex into the current partial solution.

1. Build the starting current solution as a minimum weight ‘triangle’ (for closed TSP).
2. Repeat until all vertices are in the current solution:
   - Select the vertex $v$ which is the closest to the current cycle $^{(*)}$.
   - Select two neighbor vertices $x$ and $y$ within the cycle which are closest to $v$.
   - Insert $v$ between $x$ and $y$.

Time complexity: $O(n^2)$.

Usual quality of approximate solution: 25% greater than the optimal.

# Greedy heuristics for TSP

(*) Select the vertex $v$ which is the closest to the current cycle.

- For each unvisited vertex $v$ and each pair $x, y$ of visited vertices calculate $w(v,x) + w(v,y) - w(x,y)$.
- Find $v, x, y$ for which this value is minimum.
- Insert $v$ between $x$ and $y$.

# Greedy heuristics for TSP

**3. Greedy vertex insertion.**

Greedy solution:

a,b,c,d,a: weight 8

Optimal solution:

a,b,c,d,a: weight 8