

# CIVIL-557

## Decision-Aid Methodologies in Transportation

### Lecture IV

### Metaheuristics

Fabian Torres

Transport and Mobility Laboratory TRANSP-OR  
École Polytechnique Fédérale de Lausanne EPFL

- 1 Heuristics
  - Greedy Heuristics
  - Local Search Heuristics
- 2 Metaheuristics
  - Simulated Annealing

- 1 Heuristics
  - Greedy Heuristics
  - Local Search Heuristics
- 2 Metaheuristics
  - Simulated Annealing

# What is a heuristic?

*"A heuristic is a problem-solving strategy that uses practical and common-sense approaches to find solutions that may **not be optimal** but are **good enough** for the given situation. It is a **rule of thumb** or an **educated guess** that is used when an exact or optimal solution is **difficult or impossible** to find."*

# Why use a heuristic?

## Different reasons may lead one to choose a heuristic:

- A solution is required rapidly, within a few seconds or minutes. The instance is so large and/or complicated that it cannot be formulated as an IP or MIP of reasonable size.
- Even though it has been formulated as an MIP, it is difficult or impossible for the branch-and-bound algorithm to find good feasible solutions.
- For certain combinatorial problems such as vehicle routing and machine scheduling, it is easy to find feasible solutions by inspection or knowledge of the problem structure. However, a general-purpose mixed-integer programming approach is ineffective.

# Greedy Algorithms

- Greedy heuristics construct a solution incrementally, starting with an empty solution and selecting the item with the **best immediate benefit** at each step.
- Greedy heuristics are simple and efficient, but may not always find the optimal solution.
- Greedy heuristics can be based on different criteria for selecting the "best" item at each step that minimizes cost.
- Greedy heuristics can be modified or combined with other methods, such as local search, to improve the solution.
- The effectiveness of a greedy heuristic depends on the problem structure and the quality of the criteria used to select the items at each step.

# Nearest Neighbor Heuristic

- It is a **greedy** heuristic that builds a solution iteratively;
- The nearest neighbor algorithm begins by selecting a **current** city randomly.
- The algorithm selects the nearest unvisited city to the current city at each step, until all cities are visited;
- It has a time complexity of  $\mathcal{O}(n^2)$ ;
- The **nearest neighbor heuristic** is classified as a **construction heuristic** that builds a solution from scratch.

# Nearest Neighbor Heuristic

**Input:** A set of  $N$  cities, their pairwise distances, and a starting city  $s$

**Output:** A tour of the cities

$s_{current} \leftarrow s$ ;

**while** *there are unvisited cities* **do**

**for**  $j$  *in* *Unvisited* **do**

        Find the nearest neighbor of  $s_{current}$ ;

        Let  $j^*$  be the unvisited city closest to  $s_{current}$ ;

        Add the edge  $(s_{current}, j^*)$  to the tour;

$s_{current} \leftarrow j^*$ ;

**end**

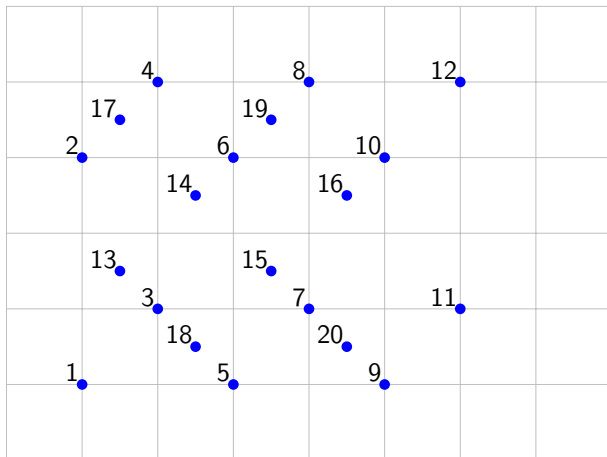
**end**

Add the edge  $(s_{current}, s)$  to complete the tour;

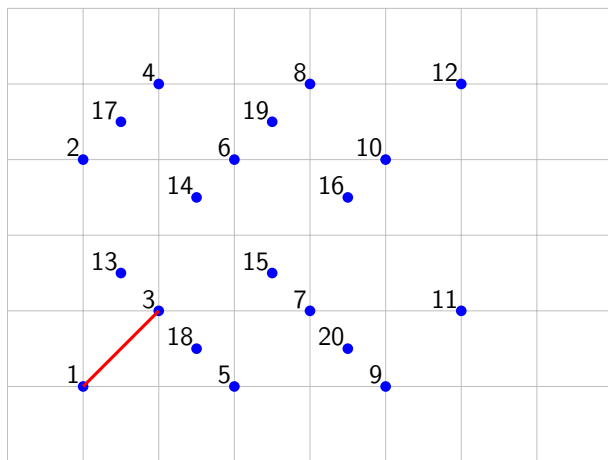
**return** the resulting tour;



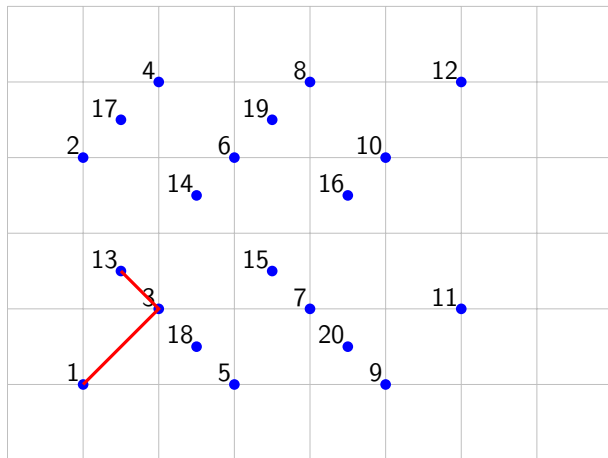
# Nearest Neighbor Example



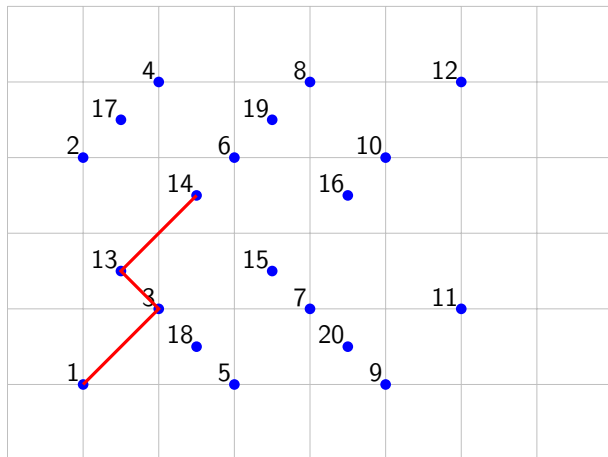
# Nearest Neighbor Example



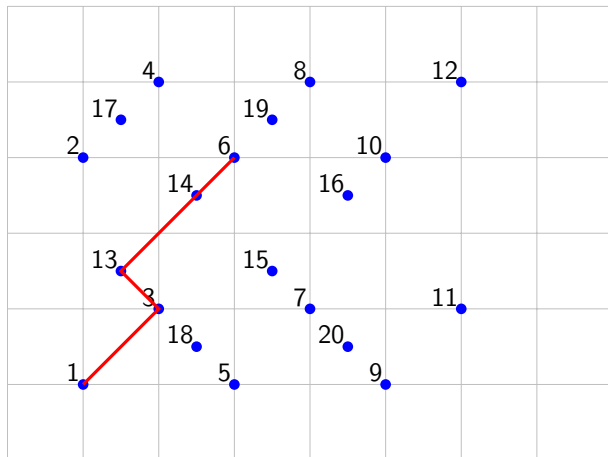
# Nearest Neighbor Example



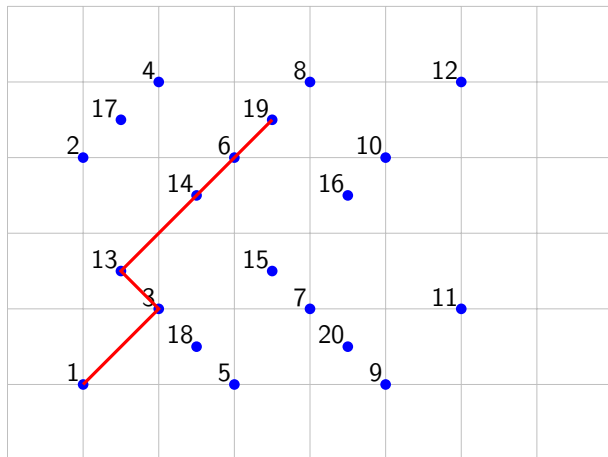
# Nearest Neighbor Example



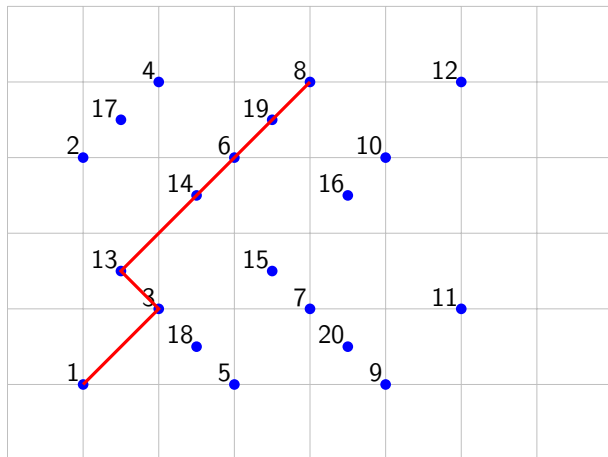
# Nearest Neighbor Example



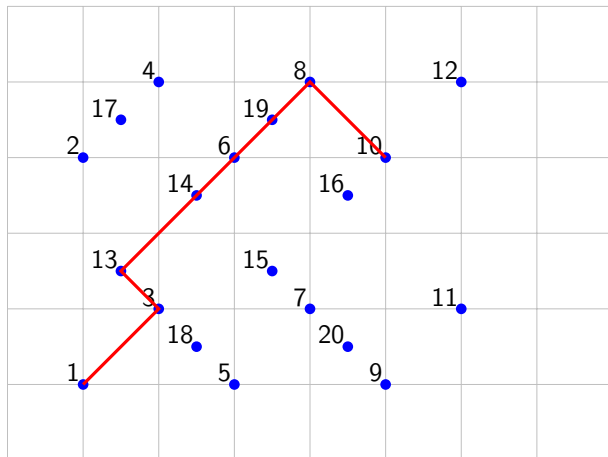
# Nearest Neighbor Example



# Nearest Neighbor Example

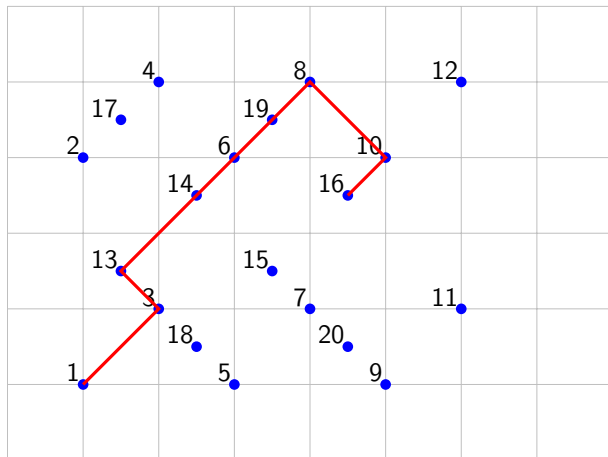


# Nearest Neighbor Example

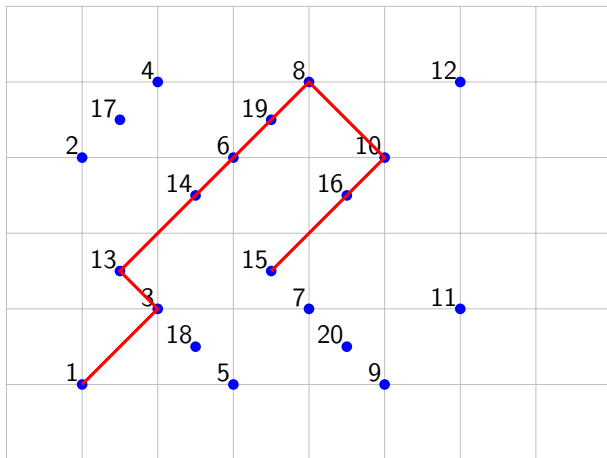




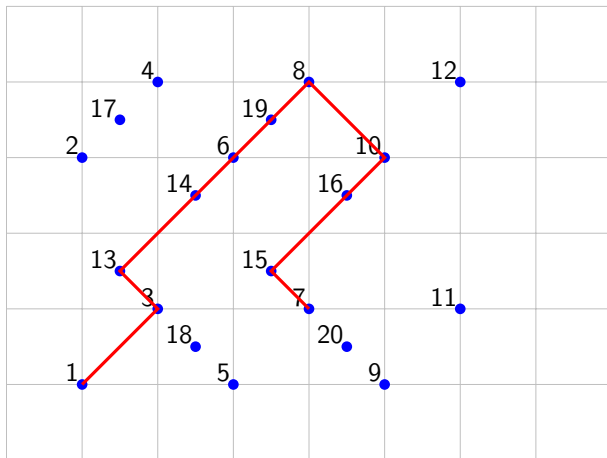
# Nearest Neighbor Example



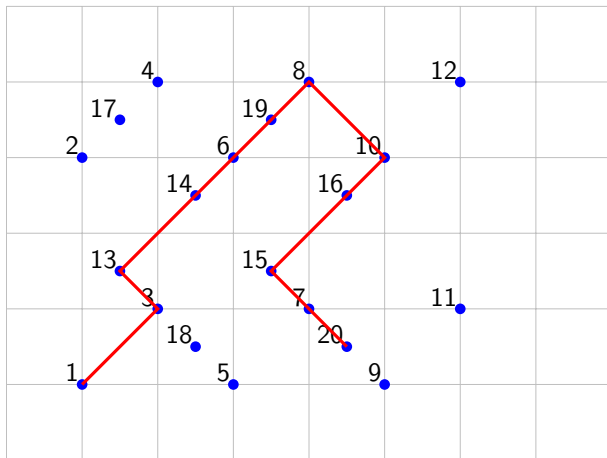
# Nearest Neighbor Example



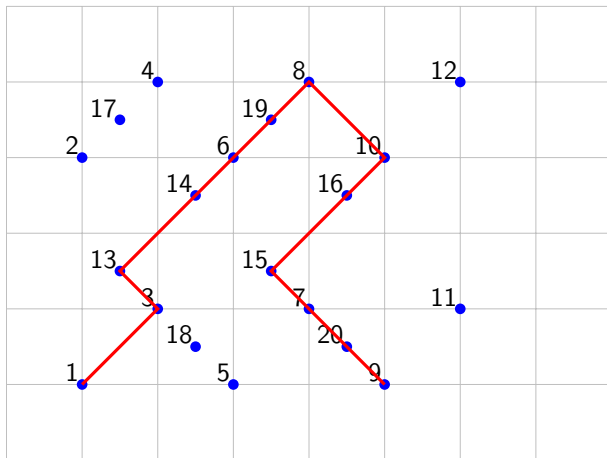
# Nearest Neighbor Example



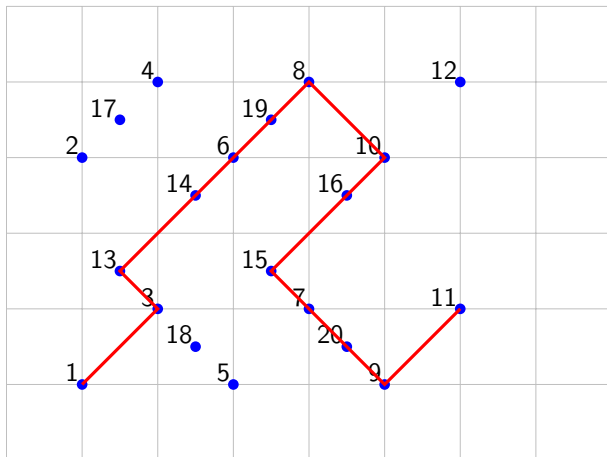
# Nearest Neighbor Example



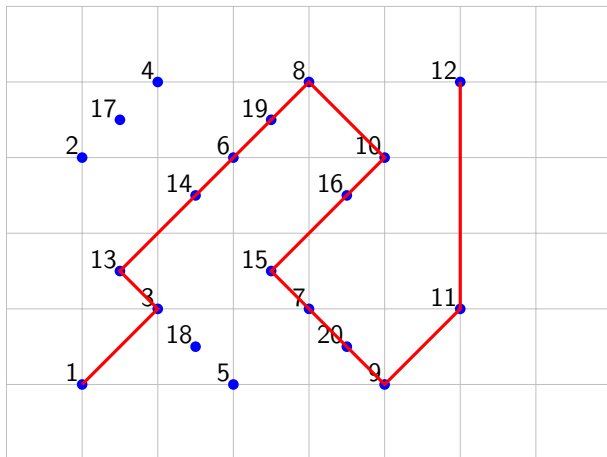
# Nearest Neighbor Example



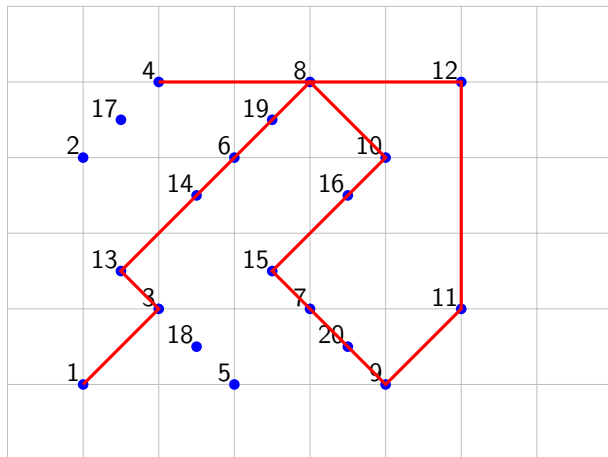
# Nearest Neighbor Example



# Nearest Neighbor Example

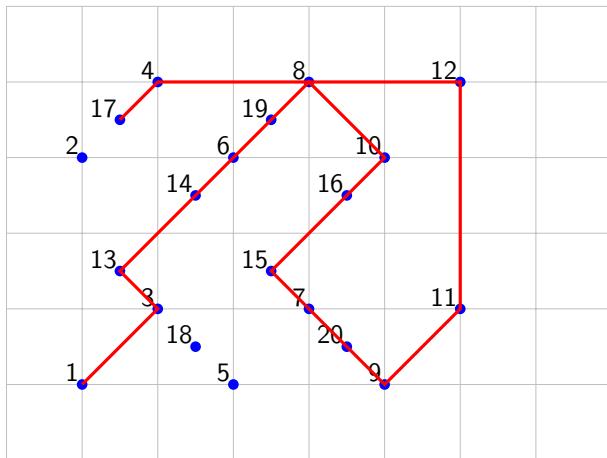


# Nearest Neighbor Example

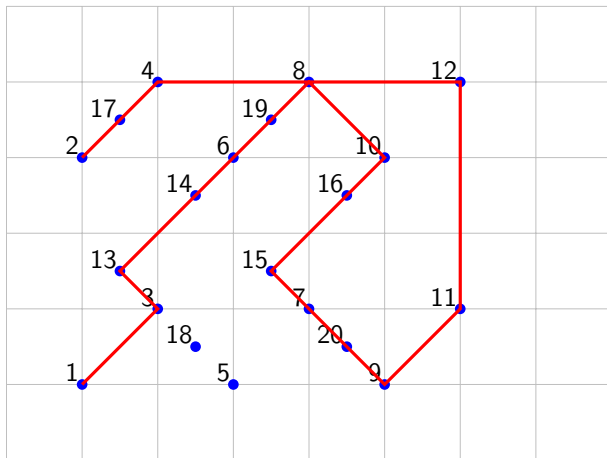




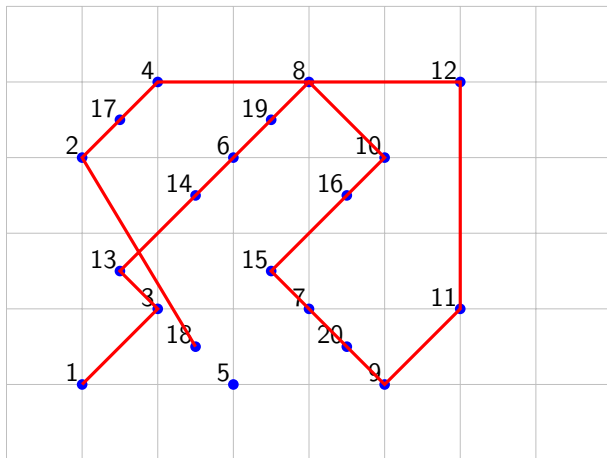
# Nearest Neighbor Example



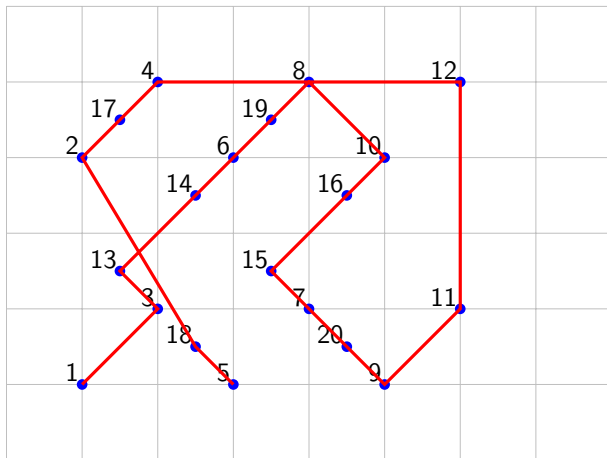
# Nearest Neighbor Example



# Nearest Neighbor Example

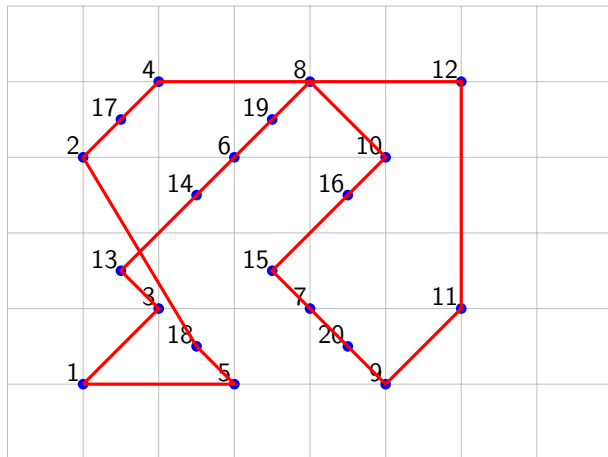


# Nearest Neighbor Example



# Nearest Neighbor Example

Total cost = 26.765



## Greedy heuristic!

- Sort the edges in nondecreasing order;
- Select the cheapest edge, if feasible, add the edge to the solution;
- Repeat until all cities are connected;
- Return to the starting city to complete the Hamiltonian tour.
- Not the nearest neighbor.

# Sorted Edges Heuristic

**Input:**  $N$  cities with pairwise distances  $d_{(i,j)}$

**Output:** A tour that visits each city exactly once and returns to the starting city

Sort = Sort the edges  $(i,j)$  in nondecreasing order of cost  $d_{(i,j)}$ ;

Initialize a list of connected components  $C$ ;

$C = \{\{i\} : \forall i \in N\}$ ;

**while**  $|C| > 1$  **do**

**for** edge  $(i,j)$  in the sorted list **do**

**if** adding edge  $(i,j)$  does not violate any constraints **then**

            Create a new component  $c(i - -j)$  that connects  $c(i)$  to  $c(j)$ ;

            Add  $c(i - -j)$  to the list of components  $C$ .;

            Remove components  $c(i)$  and  $c(j)$  from  $C$ ;

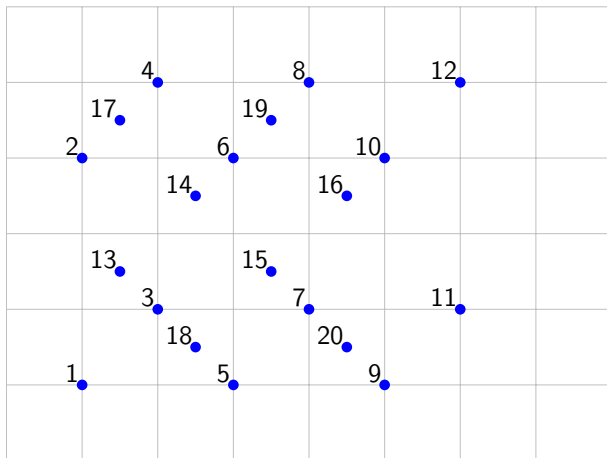
**end**

**end**

**end**

Return to the starting city to complete the tour;

# Sorted edges Example





# Sorted edges Example

List of sorted edges =  $\{(2, 17), (3, 13), (3, 18), (4, 17), (5, 18), (6, 14), (6, 19), (7, 15), (7, 20), (8, 19), (9, 20), (10, 16), (1, 3), (2, 4), (3, 5), (4, 6), (5, 7), (6, 8), (7, 9), (8, 10), (9, 11), (10, 12), (13, 14), (13, 18), (14, 15), (14, 17), (14, 19), (15, 16), (15, 18), (15, 20), (16, 19), (1, 13), (1, 18), (2, 13), (2, 14), (3, 14), (3, 15), (4, 14), (4, 19), (5, 15), (5, 20), (6, 15), (6, 16), (6, 17), (7, 16), (7, 18), (8, 16), (10, 19), (11, 20), (1, 5), (2, 6), (3, 7), (4, 8), (5, 9), (6, 10), (7, 11), (8, 12), (13, 15), (13, 17), (14, 16), (14, 18), (15, 19), (16, 20), (17, 19), (18, 20), (5, 13), (6, 13), (7, 14), (8, 14), (9, 15), (10, 15), (11, 16), (12, 16), (2, 3), (3, 6), (6, 7), (7, 10), (10, 11), (2, 19), (3, 17), (3, 20), (4, 13), (5, 14), (6, 18), (7, 13), (7, 19), (8, 15), (8, 17), (9, 16), (9, 18), (10, 14), (10, 20), (11, 15), (12, 19), (13, 19), (14, 20), (15, 17), (16, 18), (1, 14), (1, 15), (2, 15), (2, 18), (3, 16), (3, 19), (4, 15), (4, 16), (5, 16), (6, 20), (1, 2), (3, 4), (5, 6), (7, 8), (9, 10), (11, 12), (1, 7), (2, 8), (3, 9), (4, 10), (5, 11), (6, 12), (13, 16), (13, 20), (16, 17), (17, 18), (18, 19), (19, 20), (1, 17), (1, 20), (2, 16), (4, 18), (5, 19), (7, 17), (8, 13), (8, 20), (9, 14), (10, 17), (10, 18), (11, 18), (11, 19), (12, 15), (1, 6), (2, 5), (2, 7), (3, 8), (3, 10), (4, 7), (5, 10), (6, 9), (6, 11), (7, 12), (8, 11), (5, 17), (8, 18), (9, 13), (9, 19), (10, 13), (11, 14), (12, 14), (12, 20), (1, 9), (2, 10), (3, 11), (4, 12), (1, 4), (4, 5), (5, 8), (8, 9), (9, 12), (17, 20), (1, 16), (1, 19), (2, 20), (4, 20), (11, 13), (12, 17), (9, 17), (12, 18), (1, 8), (1, 10), (2, 9), (3, 12), (4, 9), (4, 11), (5, 12), (1, 11), (2, 12), (11, 17), (12, 13), (2, 11), (1, 12)\}$

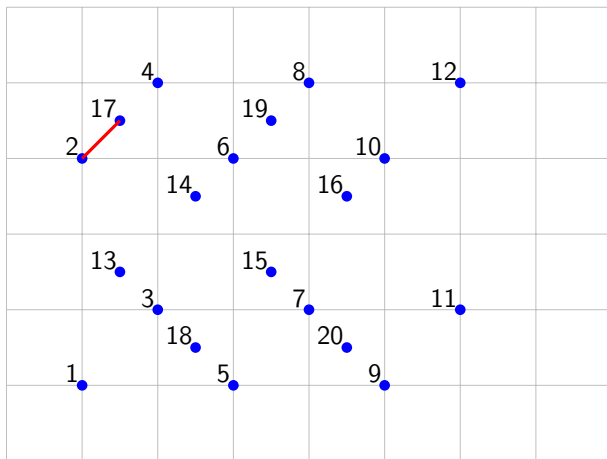
## Sorted edges Example

Initialize the set of connected components with a set of singletons for each city.

$$C = \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \\ \{11\}, \{12\}, \{13\}, \{14\}, \{15\}, \{16\}, \{17\}, \{18\}, \{19\}, \{20\}\}$$

# Sorted edges Example

Next cheapest = (2, 17)



## Sorted edges Example

(2, 17)

Add the set  $\{2, 17\}$  to  $C$ :

$$C = \{\{1\}, \{2\}, \{2, 17\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \\ \{11\}, \{12\}, \{13\}, \{14\}, \{15\}, \{16\}, \{17\}, \{18\}, \{19\}, \{20\}\}$$

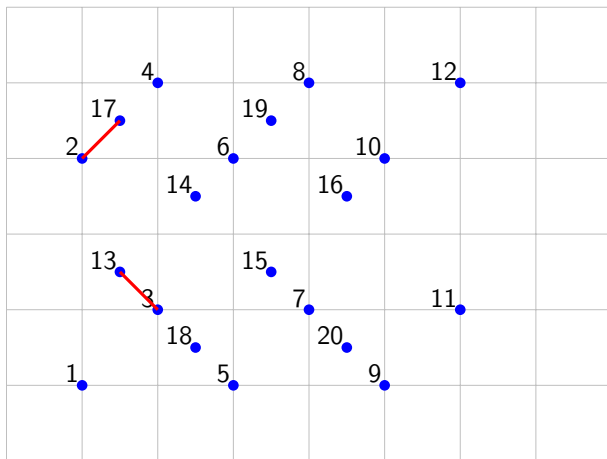
Remove the set  $\{2\}$  and  $\{17\}$ :

$$C = \{\{1\}, \{2, 17\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \\ \{11\}, \{12\}, \{13\}, \{14\}, \{15\}, \{16\}, \{18\}, \{19\}, \{20\}\}$$

$$|C| = 19$$

# Sorted edges Example

Next cheapest = (3, 13)



## Sorted edges Example

(3, 13)

Add the set  $\{3, 13\}$  to  $C$ :

$$C = \{\{1\}, \{2, 17\}, \{3, 13\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \\ \{11\}, \{12\}, \{13\}, \{14\}, \{15\}, \{16\}, \{18\}, \{19\}, \{20\}\}$$

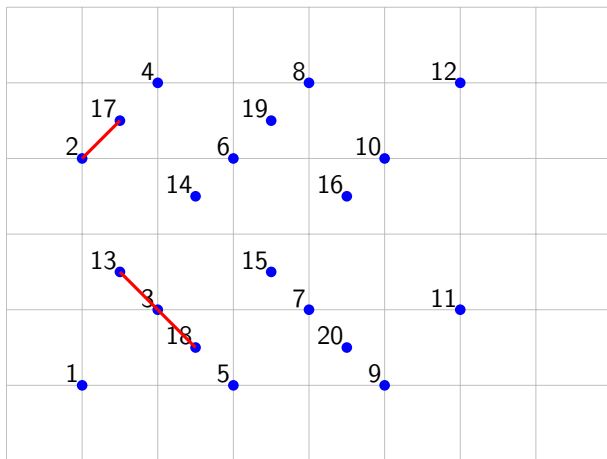
Remove the set  $\{3\}$  and  $\{13\}$ :

$$C = \{\{1\}, \{2, 17\}, \{3, 13\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \\ \{11\}, \{12\}, \{14\}, \{15\}, \{16\}, \{18\}, \{19\}, \{20\}\}$$

$$|C| = 18$$

# Sorted edges Example

Next cheapest = (3, 18)



## Sorted edges Example

(3, 18)

Add the set  $\{18, 3, 13\}$  to  $C$ :

$$C = \{\{1\}, \{2, 17\}, \{18, 3, 13\}, \{3, 13\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \\ \{11\}, \{12\}, \{14\}, \{15\}, \{16\}, \{18\}, \{19\}, \{20\}\}$$

Remove the set  $\{3, 13\}$  and  $\{18\}$ :

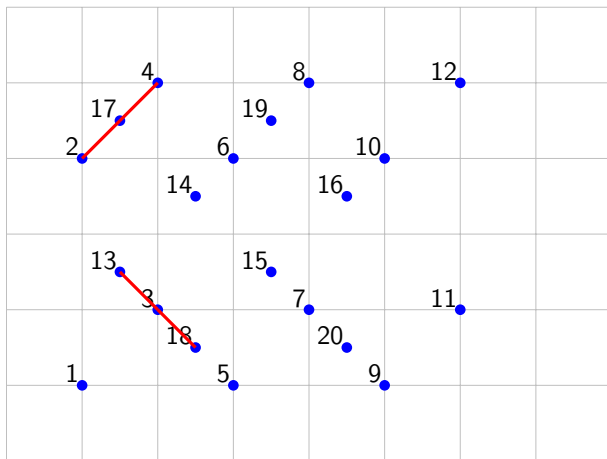
$$C = \{\{1\}, \{2, 17\}, \{18, 3, 13\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \\ \{11\}, \{12\}, \{14\}, \{15\}, \{16\}, \{19\}, \{20\}\}$$

$$|C| = 17$$



# Sorted edges Example

Next cheapest = (4, 17)



## Sorted edges Example

(4, 17)

Add the set  $\{2, 17, 4\}$  to  $C$ :

$$C = \{\{1\}, \{2, 17, 4\}, \{2, 17\}, \{18, 3, 13\}, \{3, 13\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \\ \{11\}, \{12\}, \{14\}, \{15\}, \{16\}, \{18\}, \{19\}, \{20\}\}$$

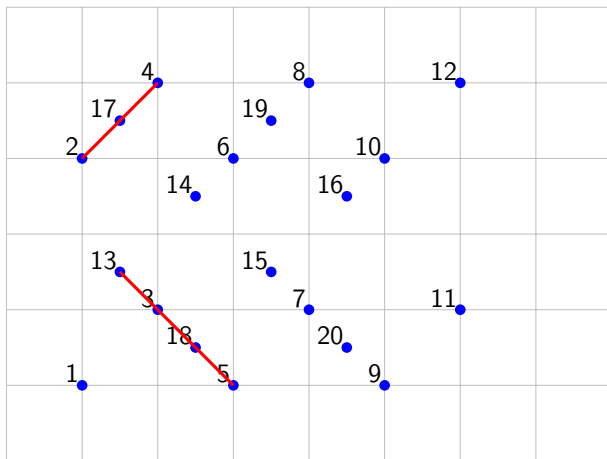
Remove the set  $\{2, 17\}$  and  $\{4\}$ :

$$C = \{\{1\}, \{2, 17, 4\}, \{18, 3, 13\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \\ \{11\}, \{12\}, \{14\}, \{15\}, \{16\}, \{19\}, \{20\}\}$$

$$|C| = 16$$

# Sorted edges Example

Next cheapest = (5, 18)



## Sorted edges Example

(5, 18)

Add the set  $\{5, 18, 3, 13\}$  to  $C$ :

$$C = \{\{1\}, \{2, 17, 4\}, \{5, 18, 3, 13\}, \{18, 3, 13\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \\ \{11\}, \{12\}, \{14\}, \{15\}, \{16\}, \{19\}, \{20\}\}$$

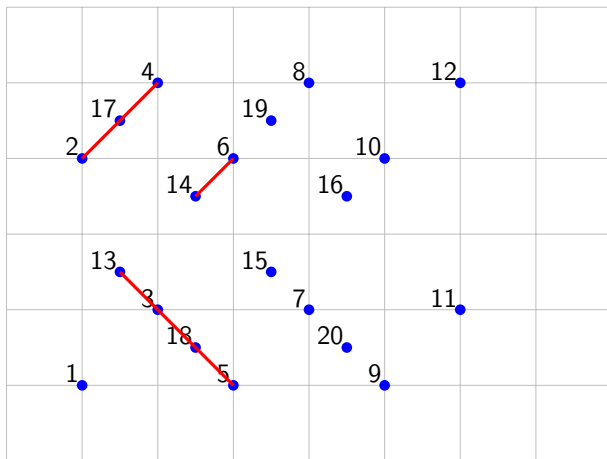
Remove the set  $\{2, 17\}$  and  $\{4\}$ :

$$C = \{\{1\}, \{2, 17, 4\}, \{5, 18, 3, 13\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \\ \{11\}, \{12\}, \{14\}, \{15\}, \{16\}, \{19\}, \{20\}\}$$

$$|C| = 15$$

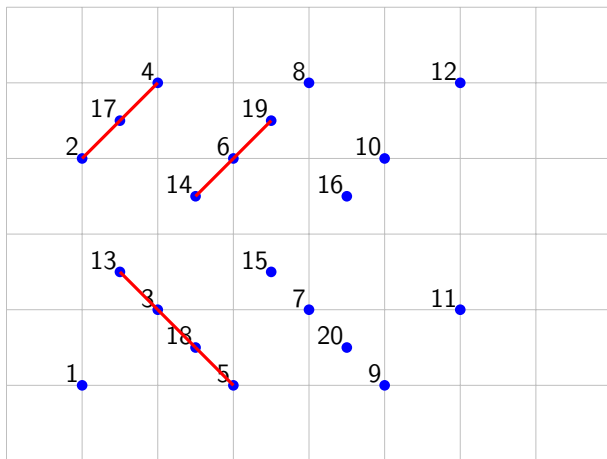
# Sorted edges Example

Next cheapest = (6, 14)



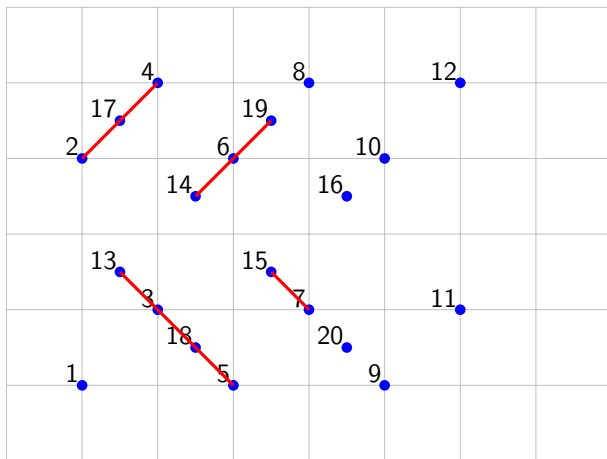
# Sorted edges Example

Next cheapest = (6, 19)



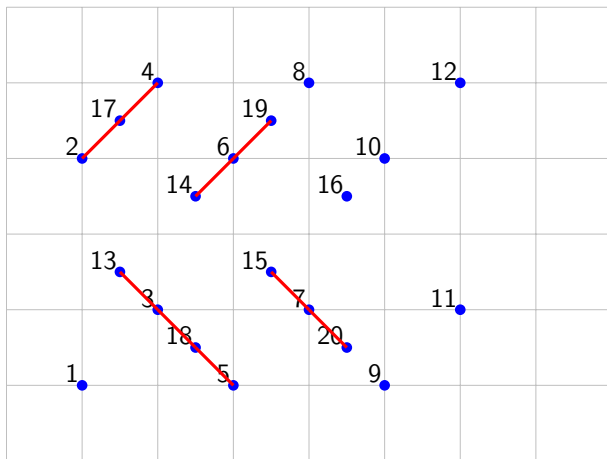
## Sorted edges Example

Next cheapest = (7, 15)



# Sorted edges Example

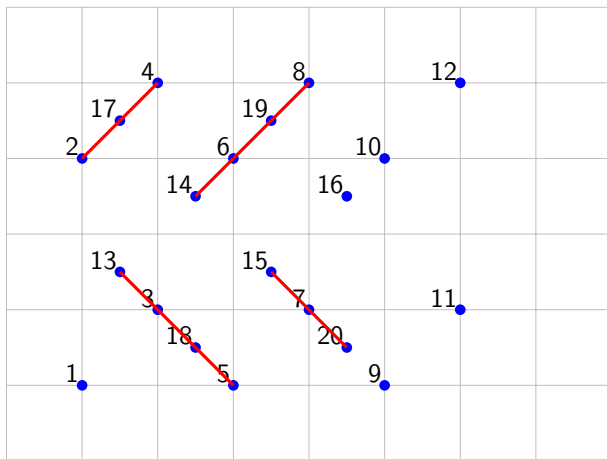
Next cheapest = (7, 20)





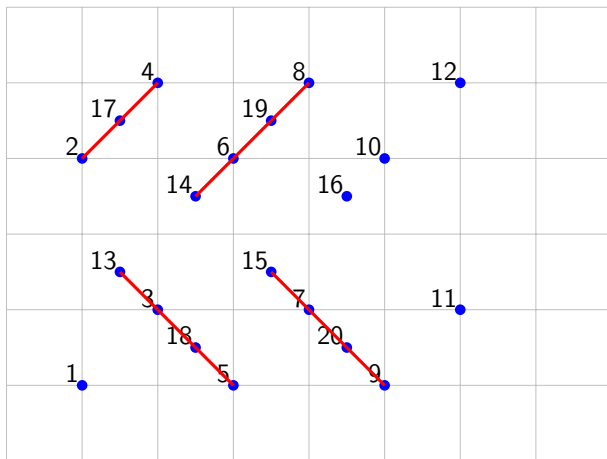
# Sorted edges Example

Next cheapest = (8, 19)



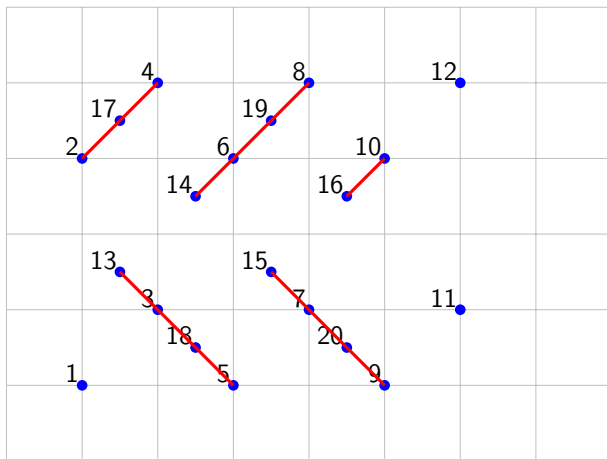
# Sorted edges Example

Next cheapest = (9, 20)



# Sorted edges Example

Next cheapest = (10, 16)



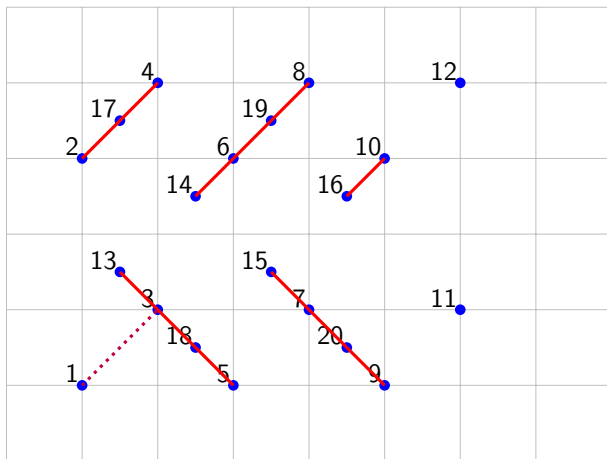
# Sorted edges Example

$$C = \{\{1\}, \{2, 17, 4\}, \{5, 18, 3, 13\}, \{14, 6, 19, 8\}, \{9, 20, 7, 15\}, \{10, 16\}, \\ \{11\}, \{12\}\}$$

$$|C| = 8$$

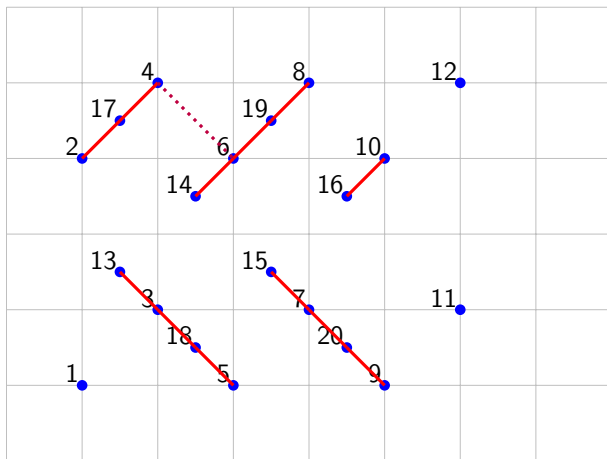
# Sorted edges Example

Next cheapest = (1, 3)



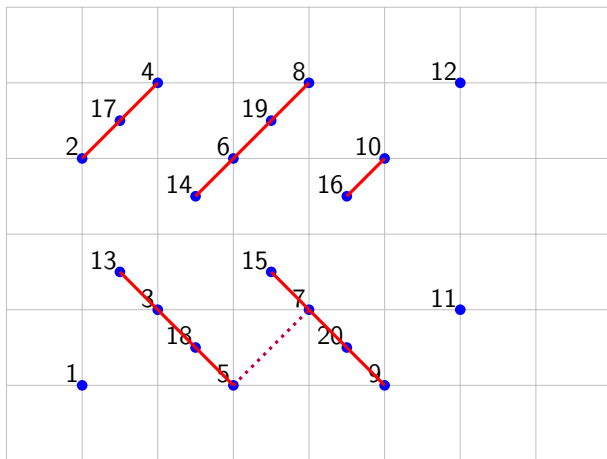
# Sorted edges Example

Next cheapest = (4, 6)



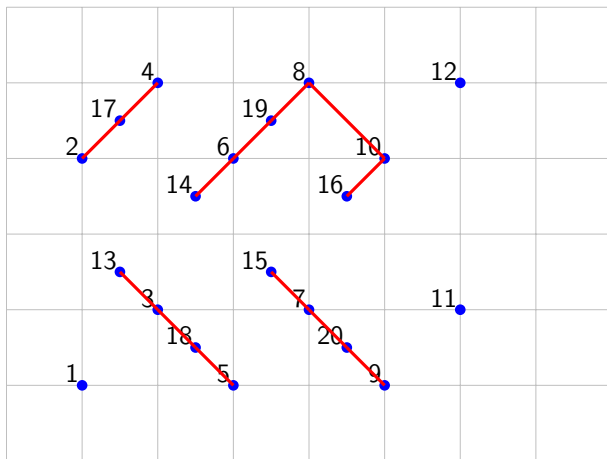
# Sorted edges Example

Next cheapest = (5, 7)



# Sorted edges Example

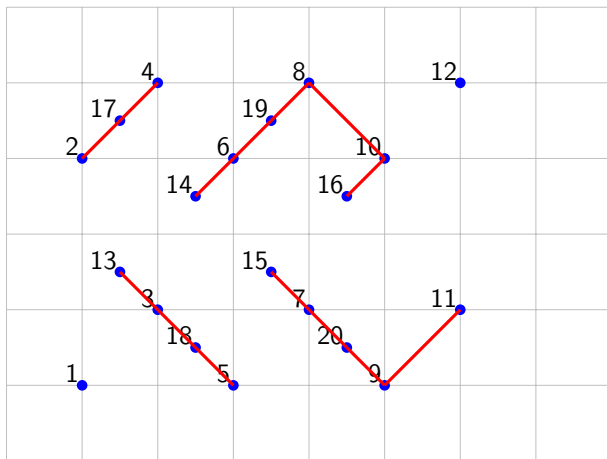
Next cheapest = (8, 10)





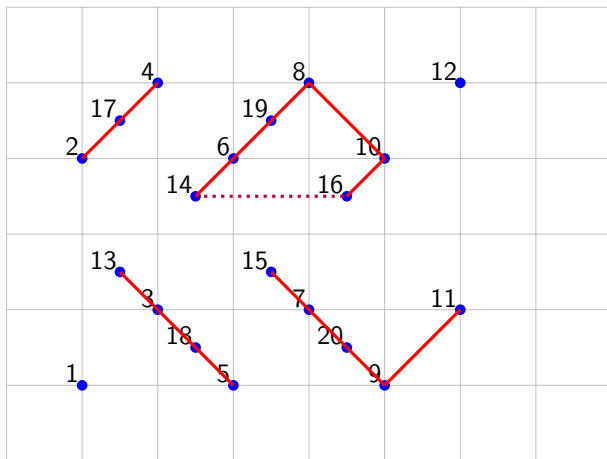
## Sorted edges Example

Next cheapest = (9, 11)



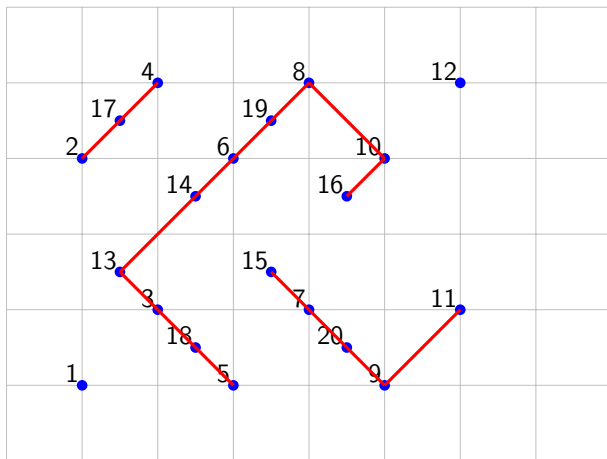
# Sorted edges Example

Next cheapest = (14, 16)



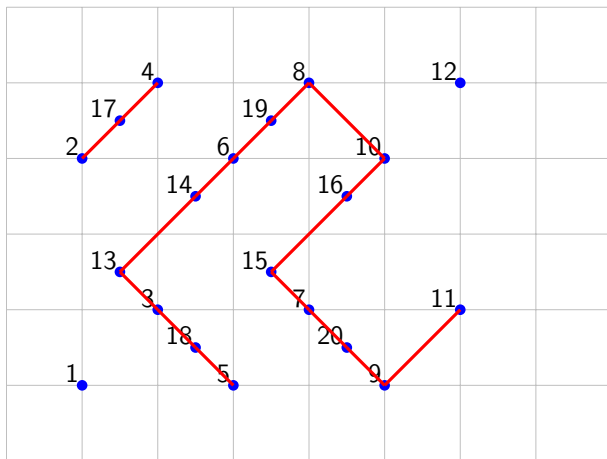
## Sorted edges Example

Next cheapest = (13, 14)



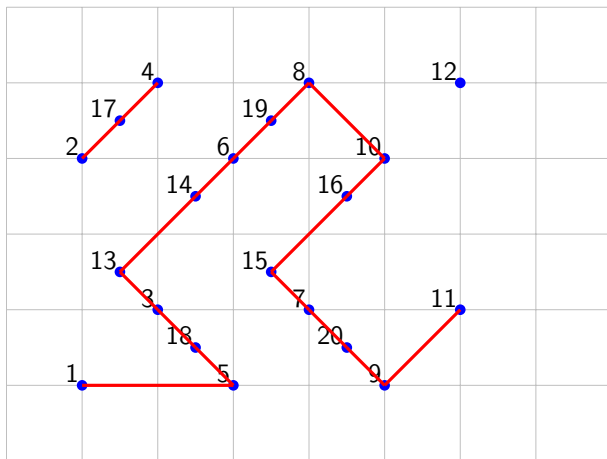
# Sorted edges Example

Next cheapest = (15, 16)



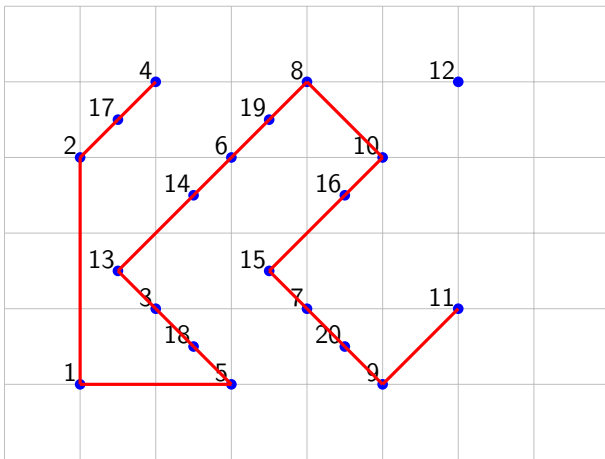
# Sorted edges Example

Next cheapest = (1, 5)



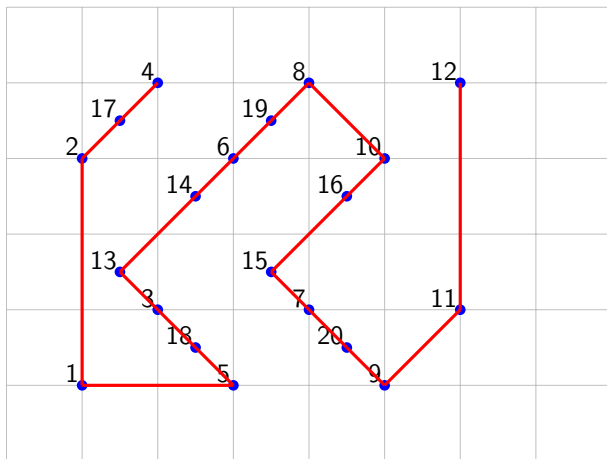
## Sorted edges Example

Next cheapest = (1, 2)



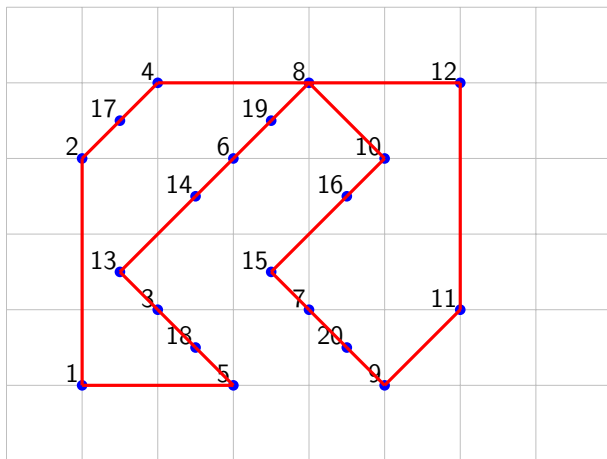
# Sorted edges Example

Next cheapest = (11, 12)



## Sorted edges Example

Next cheapest = (4, 12). Total cost = 26.142





## Sorted edges Example

$$C = \{\{2, 17, 4, 12, 11, 9, 20, 7, 15, 16, 10, 8, 19, 6, 14, 13, 3, 18, 5, 1\}\}$$

$$|C| = 1$$

**Which heuristic is better?** NN or SE?

$$NN > SE \rightarrow 26.765 > 26.142$$

- This is not always the case;
- If we start the NN in a different city the solution can change;
- When SE heuristic has equal cost edges we can get different solutions based on a random choice.
- **Is there any room for improvement?**

## Is there any room for improvement?

- How do we know that we should invest more time improving our heuristics?
- Is there a simple way to evaluate the solution;
- If we had a good lower bound for the solution then we could compare!

**The optimality gap** is the gap between a feasible solution value (upper bound) and a lower bound (relaxation).

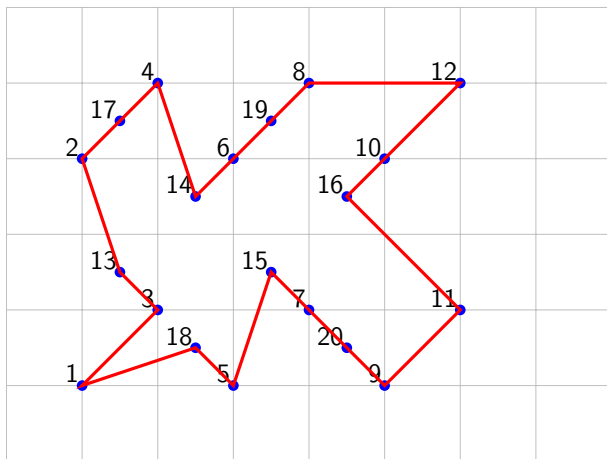
$$\frac{UB - LB}{LB} \times 100\%$$

The optimal solution is both an upper bound and a lower bound:

- A feasible solution cannot be lower than the optimal, otherwise the optimal solution would not be optimal;
- The optimal solution is a feasible solution, hence it is also an upper bound.

# Solution with B&C

Optimal cost = 22.4667



## Sorted edges

$$\frac{SE - opt}{opt} \times 100\%$$

$$\frac{26.142 - 22.467}{22.467} \times 100\% \approx 16.36\%$$

## Nearest Neighbor

$$\frac{NN - opt}{opt} \times 100\%$$

$$\frac{26.765 - 22.467}{22.467} \times 100\% \approx 19.13\%$$

Is a 16.36% gap a "**good**" solution?

- The size and complexity of the problem, as well as the time constraints, can make it necessary to **accept a suboptimal solution** with a larger gap.
- The cost implications of the suboptimal solution, such as high shipping costs, may require a more accurate solution with a smaller gap.
- The acceptable gap between solutions may vary depending on the level of service quality required or the specific needs of the problem at hand.

**Local Search algorithms** optimize the cost function by exploring the neighborhood of the current point in the solution space.

- A move is a change in the solution, e.g., swapping customers in a route.
- LS makes a move that improves the solution at each step.
- Once no improving predefined moves exist the algorithm stops (**Local optima**).



Let  $S$  be the set of feasible solutions and let  $f$  be the objective function of the problem.

- **Definition 1:** Let  $\mathcal{H}$  be a heuristic that defines for each solution  $w \in S$  a subset  $S_w \subseteq S$  of solutions “close” (to be defined by the user according to the problem of interest) to the solution  $w$ . The subset  $S_w$  is called the neighborhood of solution  $w$ .
- **Definition 2:** A solution  $w^* \in S$  is called a local optimum with respect to  $\mathcal{H}$  for the subset of feasible solutions  $S$  and the objective function  $f$  if  $f(w^*) \leq f(z)$  for all  $z \in S_{w^*}$ .
- **Definition 3:** The neighborhood structure  $\mathcal{H}$  is said to be exact if, for every local optimum with respect to  $\mathcal{H}$ ,  $w^* \in S$ ,  $w^*$  is also a global optimum of  $S$  and  $f$ .

# Local Search Algorithms

**Input** : Initial solution  $w$

**Output:** Best solution found

**while** *termination condition not met* **do**

    Generate a solution  $z$  from the neighborhood  $S_w$  of the current solution  $w$ ;

**if**  $f(z) < f(w)$  **then**  
        |  $w \leftarrow z$ ;

**end**

**if**  $f(z) \geq f(w)$  for all  $z \in S_w$  **then**  
        | **Terminate**;

**end**

**end**

**Algorithm 1:** Local Search

- The **k-opt** algorithm is a **local search** algorithm for the **Traveling Salesman Problem (TSP)**.
- The algorithm works by iteratively improving a **feasible solution** to the TSP by exploring its **neighborhood** using a **k-opt move**.
- The **k-opt move** involves removing **k** edges from the current solution and reconnecting the resulting fragments in a new way to obtain a new feasible solution.
- The value of **k** determines the size of the neighborhood and the complexity of the search.
- The algorithm terminates when no further improvement can be made, and the current solution is returned as the approximate solution to the TSP.

## 2-opt algorithm

**Input** : An initial tour  $T$  for the TSP

**Output:** An approximate solution to the TSP

Set  $T^* \leftarrow T$ ;

**while** *no improvement is made* **do**

**for**  $i \in T^*$  **do**

**for**  $j = i + 1$  **to**  $|T|$  **do**

**if**  $d_{i,i-1} + d_{j+1,j} > d_{i,j+1} + d_{i-1,j}$  **then**

$R \leftarrow$  Reverse the segment between cities  $i - 1$  and  $j + 1$ ;

$T^* \leftarrow R$ ;

**end**

**end**

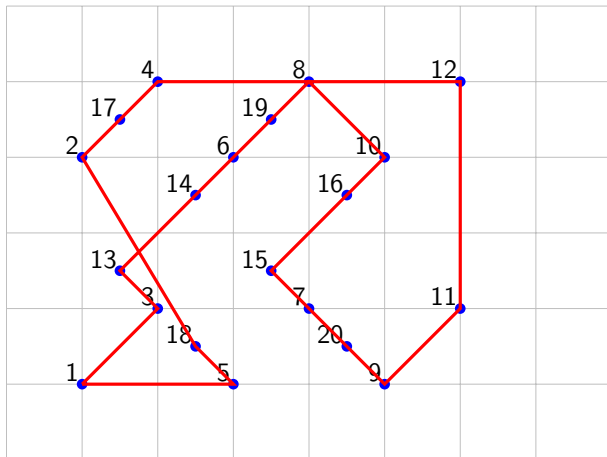
**end**

**end**

**Algorithm 2:** 2-opt algorithm for TSP

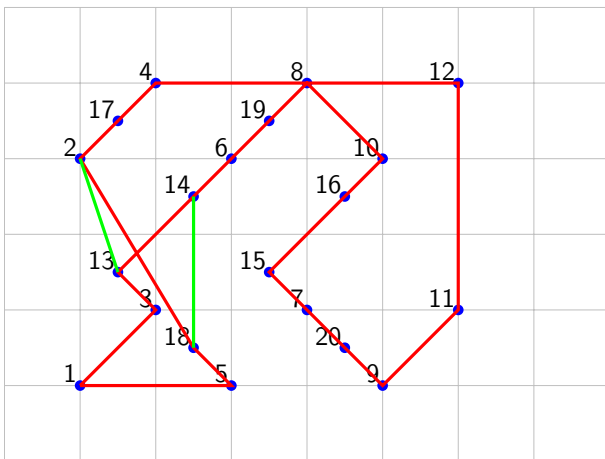
## 2-opt Example

Initial tour is the NN solution



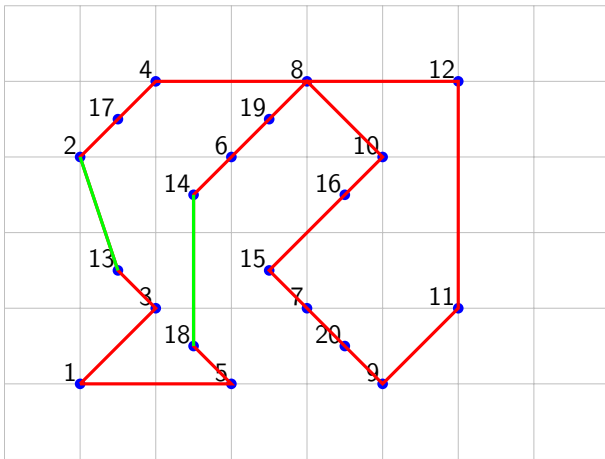
## 2-opt Example

A 2-opt move consists of replacing 2 edges with 2 new edges



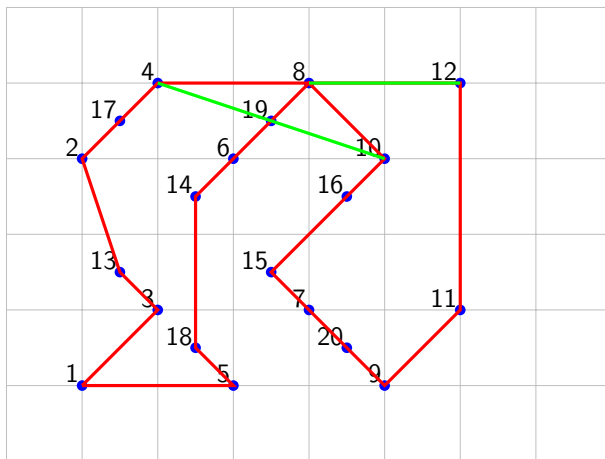
## 2-opt Example

Remove (13,14) and (2,18). Now add (2,13) and (14,18) to reconnect the tour.



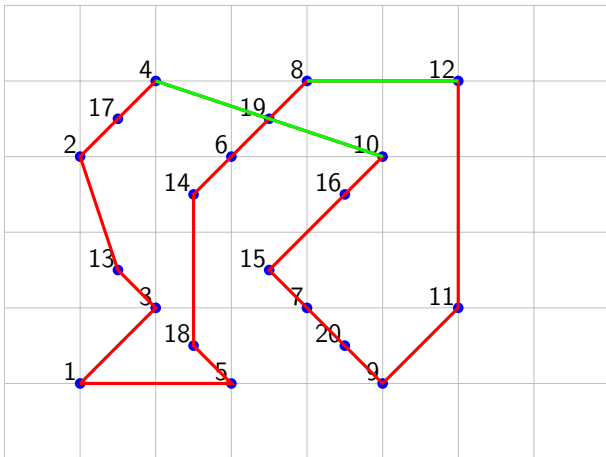
## 2-opt Example

2-opt continues only making moves that improve the objective function.



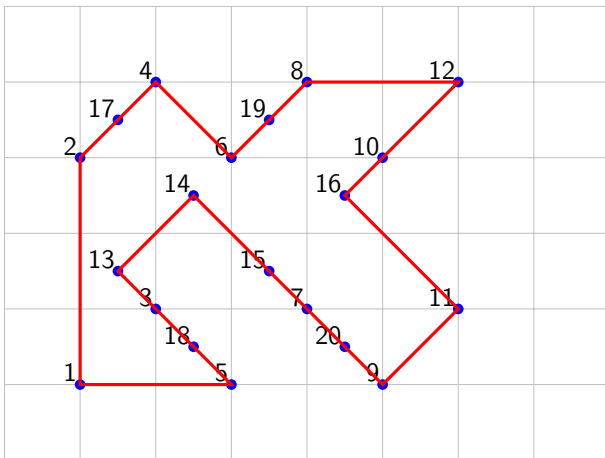


## 2-opt Example



## 2-opt Example

2-opt solution value = 23.97.    **Local optimum**



Is this solution better?

$$\frac{2_{opt} - opt}{opt} \times 100\%$$

$$\frac{23.97 - 22.467}{22.467} \times 100\% \approx 6.72\%$$

## Why not do 10-opt or more?

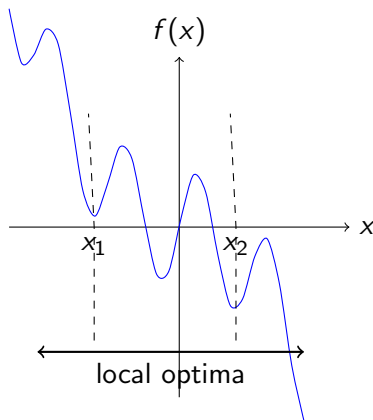
- The time complexity of each iteration for the k-opt algorithm is:

$$\mathcal{O}(n^k)$$

In general, the time complexity of the k-opt algorithm increases exponentially with the value of k, so it is often used with small values of k (such as 2 or 3) to balance between solution quality and computation time.

# Local Search

- In a local optimal point there is no descent direction. That is, there is no improving solution in the neighborhood around the current solution



- In Local search the local optima is defined by the heuristic (e.g., 2-opt heuristic).
- Once the heuristic cannot identify an improving move, then we say that the solution is a local optima (e.g., 2-opt local optima).
- Notice that if we change the heuristic (e.g, 4-opt) we could improve the solution to arrive at a new local optima for the new heuristic.
- **Disadvantage** local search is unable to escape local optima since it does not accept non improving solutions.

- 1 Heuristics
  - Greedy Heuristics
  - Local Search Heuristics
- 2 Metaheuristics
  - Simulated Annealing

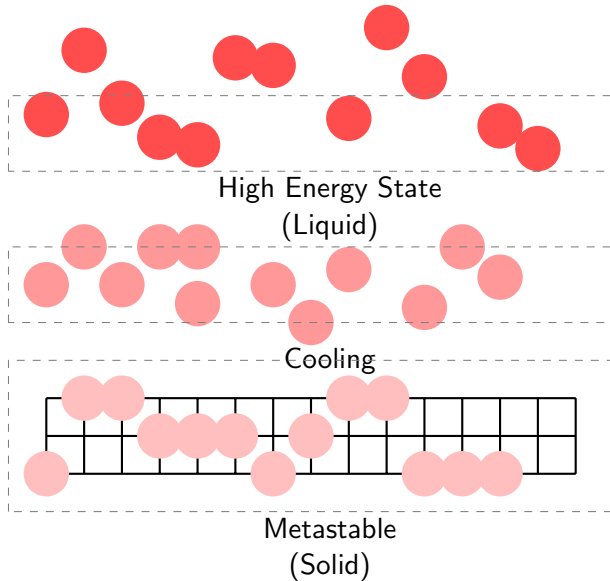
- The idea is to escape local optima by altering the solution in some way ( sometimes called kick, shake or destroy).
- Moves that cause an increase in the function can be accepted to escape local optima.
- Once the solution has been altered in some way that is sufficient to escape, local search is used again to improve the new solution.
- Generally, the procedure is repeated for some pre-specified number of iterations.



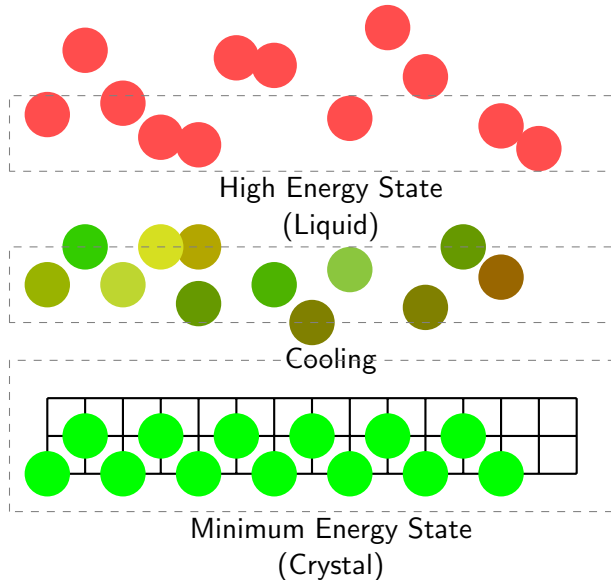
# Annealing

- Annealing is a process of heating and cooling a material to change its properties.
- It is commonly used in the manufacturing of metals and glass.
- The annealing process involves heating the material to a high temperature, then allowing it to **cool slowly**.
- This slow cooling allows the material's atoms to rearrange themselves into a more stable configuration.
- Annealing can result in changes to the physical and mechanical properties of a material, such as strength, ductility, and toughness.
- It can also be used to relieve stresses in a material that occur during processes such as welding or machining.

# Annealing: Rapid cooling



# Annealing: Slow gradual cooling



# Simulated Annealing(SA)

- SA is a metaheuristic optimization algorithm based on the physical process of annealing in materials.
- The SA process involves starting with an initial solution and gradually modifying it by changing some elements of the solution.
- At each step, the new solution is evaluated and accepted or rejected based on a **probability** that depends on the difference between the energy of the new solution and the energy of the current solution, as well as the current temperature.
- Initially, the temperature is set high so that the system is in a high-energy state, allowing for a more complete exploration of the solution space.
- As the temperature is gradually decreased, the system is encouraged to settle into a **lower-energy** state.
- If the temperature decreases too quickly, the system can become trapped in a **local optimum**, known as hardening.

# Simulated Annealing

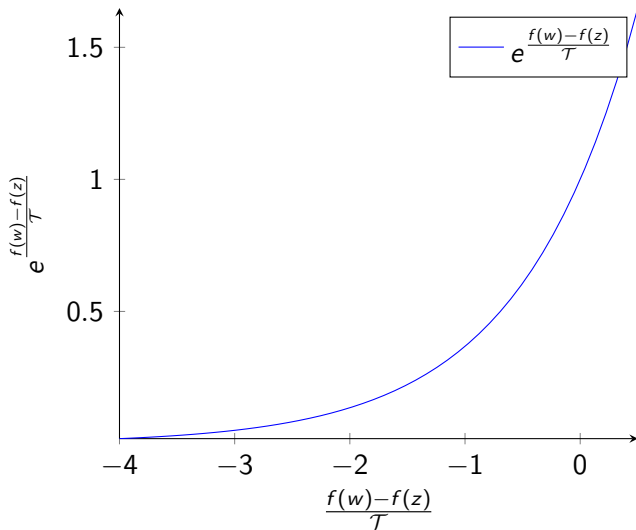
- Let the temperature be  $\mathcal{T}$ ;
- Let  $f(z)$  be the new solution's objective value;
- Let  $f(w)$  be the current solution's objective value;
- Let  $\Pr\{\text{accept } z\}$  be the probability of accepting  $z$  as a new current solution.

**Probability of accepting a new solution:**

$$e^{\frac{f(w)-f(z)}{\mathcal{T}}}$$

$$\Pr\{\text{accept } z\} = \begin{cases} 1 & \text{if } f(z) < f(w) \\ e^{\frac{f(w)-f(z)}{\mathcal{T}}} & \text{otherwise} \end{cases}$$

# Simulated Annealing



# Simulated Annealing

**Input** :  $w_{start}, \mathcal{T}_0, L_0$

**Output**: Best solution found

$w \leftarrow w_{start}, k \leftarrow 0, \mathcal{T}_k \leftarrow \mathcal{T}_0, L_k \leftarrow L_0;$

**while**  $\mathcal{T}_k \neq 0$  **do**

**for**  $l = 0$  **to**  $L_k$  **do**

        Generate a solution  $z$  from the neighborhood  $S_w$ ;

**if**  $f(z) < f(w)$  **then**

$w \leftarrow z$ ;

**end**

**else**

            Accept  $z$  as the current solution with probability  $e^{\frac{f(w)-f(z)}{\mathcal{T}_k}}$ ;

**end**

**end**

$k \leftarrow k + 1$ ;

    Compute  $(L_k, \mathcal{T}_k)$ ;

**end**

**return**  $w$ ;



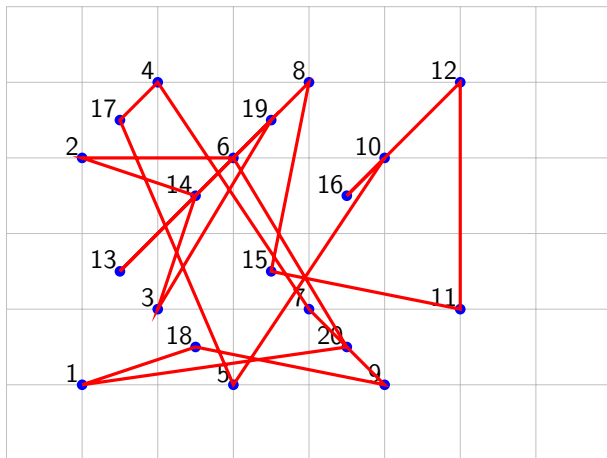
# Simulated Annealing

- At each iteration, the algorithm cools the temperature by a cooling factor of  $\alpha$ .

$$\mathcal{T}_{k+1} \leftarrow \alpha \times \mathcal{T}_k$$

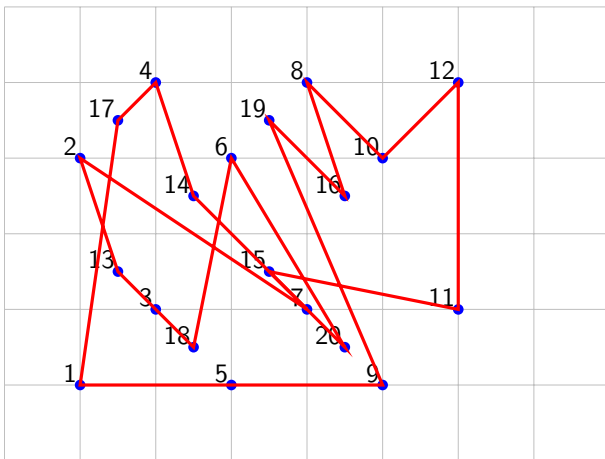
# SA Example

$$\mathcal{T}_0 = 1000, \alpha = 0.999, L = 5$$



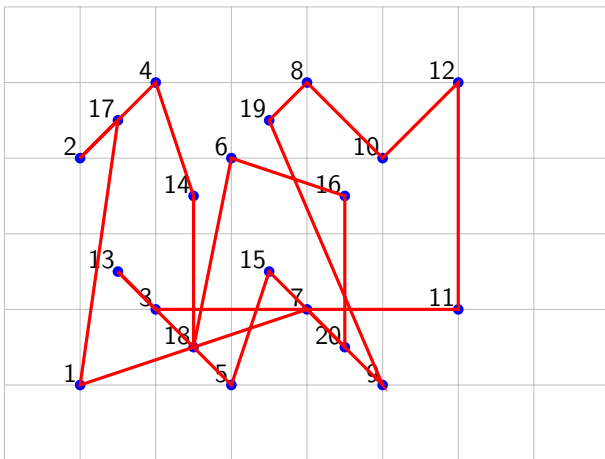
# SA Example

$$\mathcal{T}_k = 499.9$$



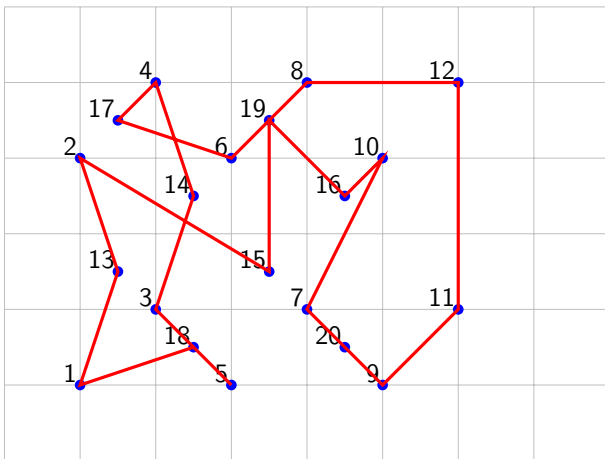
# SA Example

$$\mathcal{T}_k = 124.94$$



# SA Example

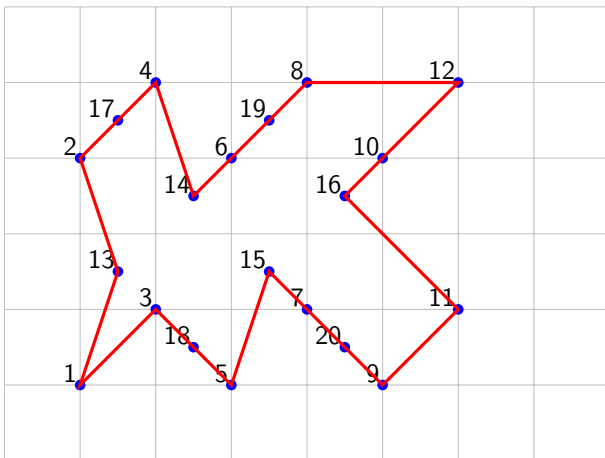
$$\mathcal{T}_k = 1.0$$



# SA Example

$$\mathcal{T}_k \simeq 0.0$$

Total cost 22.467



# SA calibrating parameters

## Cooling temperature

- If the cooling factor “ $\alpha$ ” is too small the temperature will cool down too quickly. The solution might get stuck in local optima;
- If it is too high the algorithm will take longer to converge to a solution.

## Temperature

- If the initial temperature is too low, SA can fail to escape local optima.
- If the temperature is too high, SA will take a long time to converge to a solution.

## Stopping criteria

- If the stopping criteria is too high, SA will stop before final improvements can be explored.
- If it is too low it might take a long time for the algorithm to stop, even after finding a good solution. There will be no improvements.

**The cooling:**

$$\mathcal{T}_{k+1} = \alpha \times \mathcal{T}_k$$

**The algorithm will stop when:  $\mathcal{T}_k \leq stop$  therefore:**

$$\alpha^{k^*} \mathcal{T}_0 = stop$$

Where  $k^*$  is the total number of iterations and stop is the stopping value for the temperature.



# SA calibrating parameters

To solve for  $\alpha$ :

$$\alpha = \left( \frac{\text{stop}}{\mathcal{T}_0} \right)^{\frac{1}{k^*}}$$

To solve for  $k^*$ :

$$k^* = \frac{\ln \left( \frac{\text{stop}}{\mathcal{T}_0} \right)}{\ln(\alpha)}$$

To solve for  $\mathcal{T}_0$ :

$$\mathcal{T}_0 = \frac{\text{stop}}{\alpha^{k^*}}$$

To solve for stop:

$$\text{stop} = \alpha^k \cdot \mathcal{T}_0$$

# SA: Initial Temperature

- ① Set stopping criteria “stop” equal to a small number, e.g.,  $stop = 10^{(-8)}$ .
- ② Set temperature to a value that is not too small or too large.
  - ① Run 20 iterations of the algorithm with the temperature equal to 0, i.e.,  $T_0 = 0$ ;
  - ② Determine the absolute value of the difference from the current solution to the new solution;
  - ③ Set the initial temperature to the average of the difference.

# SA: Iterations and cooling rate

It depends how long you have to generate solutions.

- 1 Determine the time that you have to run the algorithm, e.g., 1 minute, 20 minutes, etc.
- 2 Determine how long it takes for each iteration and set iterations to the amount of time you have.
- 3 Set the cooling rate accordingly:

$$\alpha = \left( \frac{\text{stop}}{\mathcal{T}_0} \right)^{\frac{1}{k^*}}$$

# SA: conclusion

- Simulated Annealing is a stochastic algorithm.
- Stochastic algorithms depend on random parameters that change every time we run the algorithm.
- Stochastic algorithms can provide different solutions every time we run.
- It can be necessary to run a few times to get the best solution.

# Stochastic vs Deterministic

**Stochastic algorithms** are algorithms that use random elements in their search process. They can be thought of as probabilistic algorithms that make use of randomness to explore the search space.

**Deterministic algorithms** are algorithms that produce the same output for a given input every time they are run. They do not use any randomness in their search process.

# Summary

- Heuristics are used for the following reasons:
  - When **time is of essence**;
  - The **exact** solution might be **difficult or impossible** to find in a reasonable amount of time;
- It is necessary to have a lower bound to compare the quality of the solutions provided by the heuristic.
  - Heuristic solutions can be arbitrarily bad, while exact methods provide the best solution, heuristics do not.
  - The need to compare and improve the heuristics is always present.

# Summary

## Heuristics can be classified in different ways

- **Constructive heuristics:** Build solution from scratch.
  - Nearest Neighbor heuristic.
  - Sorted edges heuristic.
- **Local search**
  - k-opt
- **Deterministic:** if the algorithm converges to the same solution
  - Nearest Neighbor; if we start with a predefined city;
  - Sorted edges
  - k-opt
- **Stochastic:** if the solution depends on some random parameters.
  - Simulated annealing.
  - Nearest Neighbor, if we start in a random city.
- **Greedy:** always choosing the best step:
  - Nearest Neighbor heuristic.
  - Sorted edges heuristic.

# Summary

- Constructive heuristics are fast and can build initial solutions quickly;
- Local search algorithms improve solutions by making specific “moves” that search the neighborhood of the solution;
- Local search heuristics define the neighborhood by the moves;
- Local search heuristics get stuck in local optima and cannot escape;
- To find better solutions we need to escape local optima by momentarily accepting bad solutions that worsen the objective value.
- Simulated Annealing uses a stochastic approach to accept solutions so that there exists a high probability of moving to a worse solution.
- The probability of acceptance of bad solutions decreases as the temperature cools.



- Wolsey, L. A. (1998). *Integer programming*. Wiley.
- Gendreau, M., & Potvin, J.-Y. (Eds.). (2010). *Handbook of metaheuristics*. Springer.