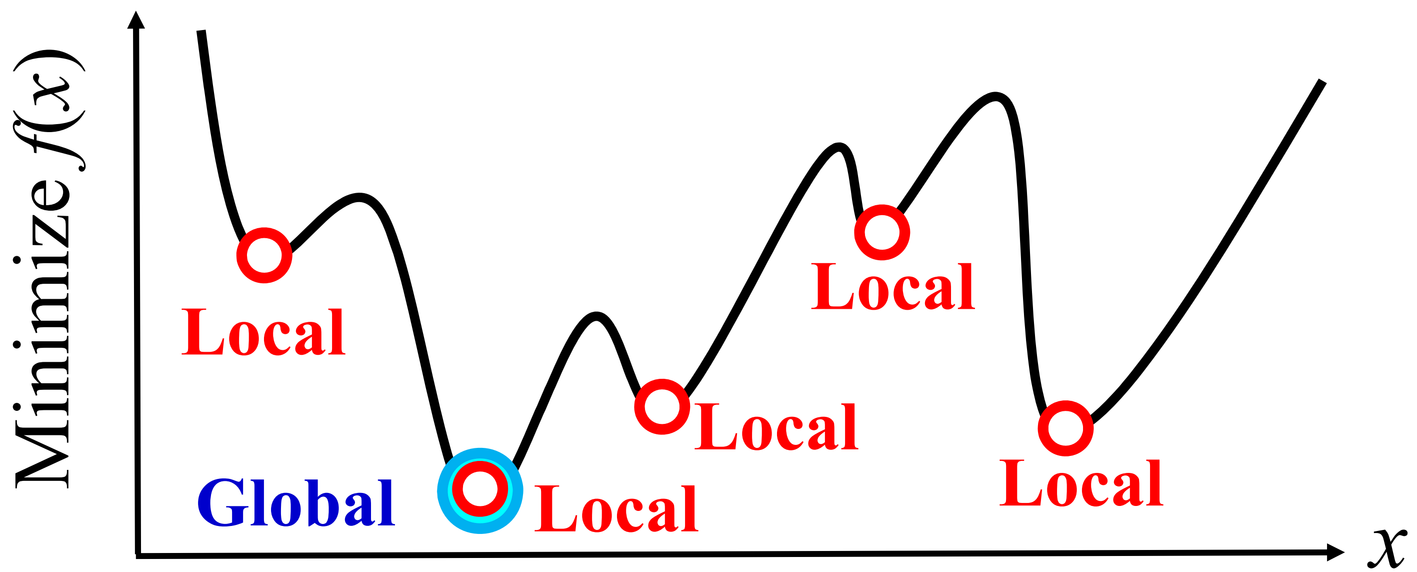
## Terminology

**Local solution, Locally optimal solution**

A solution $\boldsymbol{x}$ is referred to as a locally optimal solution (local solution) if and only if $\boldsymbol{x}$ has no better solution in its neighborhood.

**Global solution, Globally optimal solution, Optimal solution**

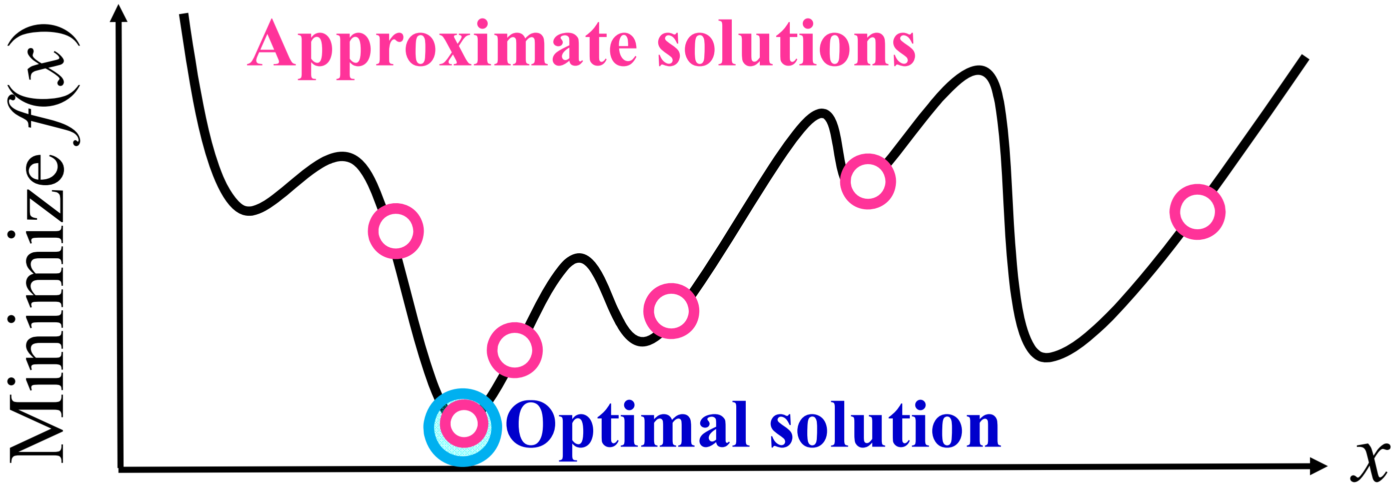
A solution $\boldsymbol{x}$ is referred to as a globally optimal solution (global solution, optimal solution) if and only if $\boldsymbol{x}$ is the best solution among all feasible solutions.



## Terminology

### Approximate solution

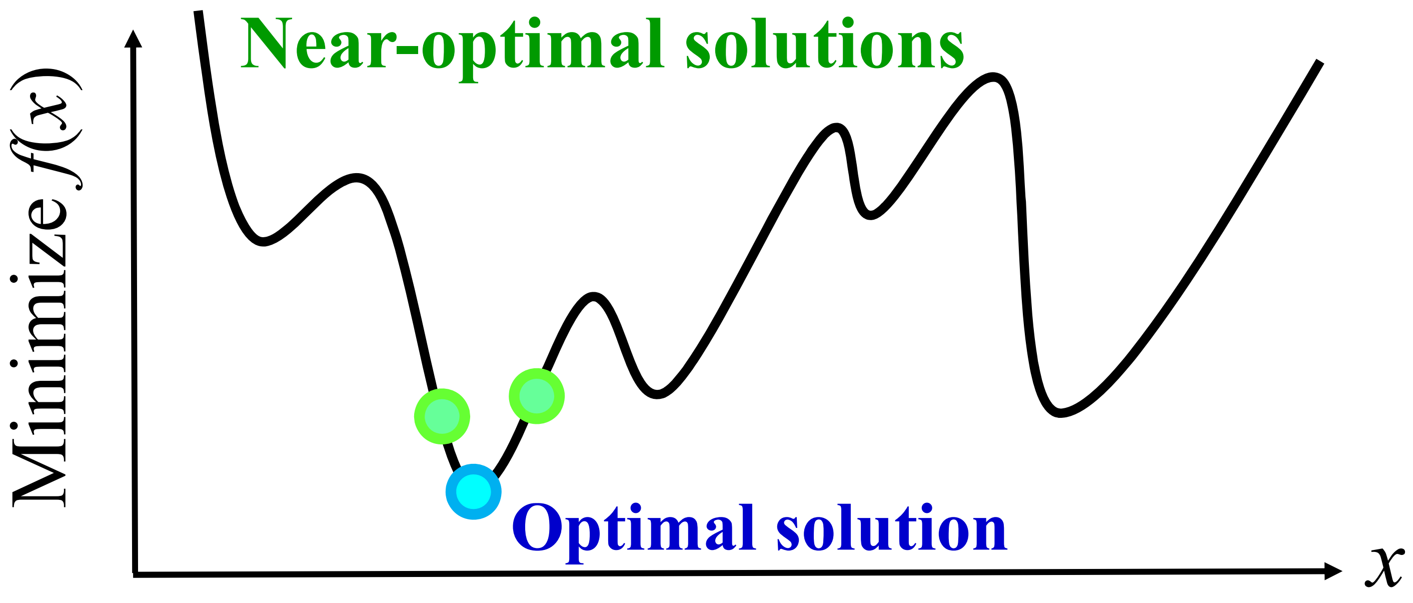
A solution ***x*** is referred to as an approximate solution when ***x*** is obtained by an approximation algorithm. An approximate solution ***x*** can be very bad, very good, and optimal. However, the approximation algorithm does not know whether ***x*** is optimal or not. For example, if we randomly generate a solution, it is a randomly generated solution (not an approximate solution). If we select the best solution from randomly generated 100 solutions, the selected solution is an approximate solution. It can be very bad and very good (optimal).



## Terminology

**Near-optimal solution**

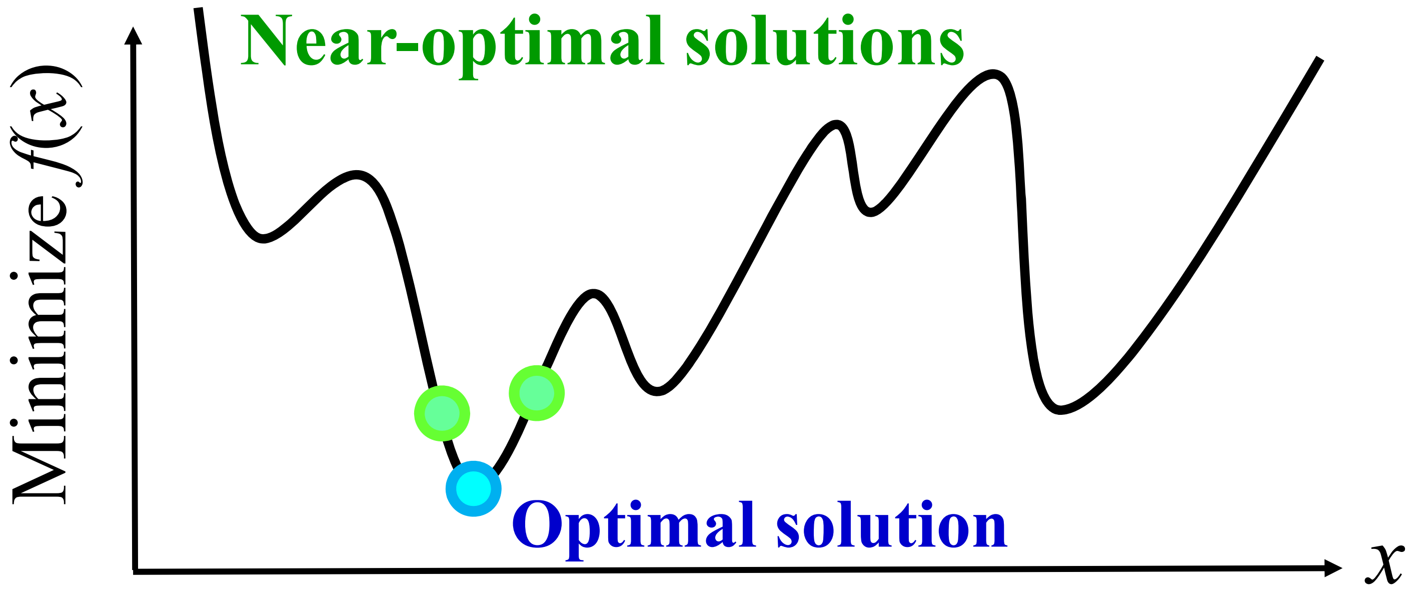
A near-optimal solution is a solution which is close to the optimal solution. It seems to me that there is no clear mathematical definition of near-optimal solutions (i.e., there is no clear mathematical definition of “near”).



## Terminology

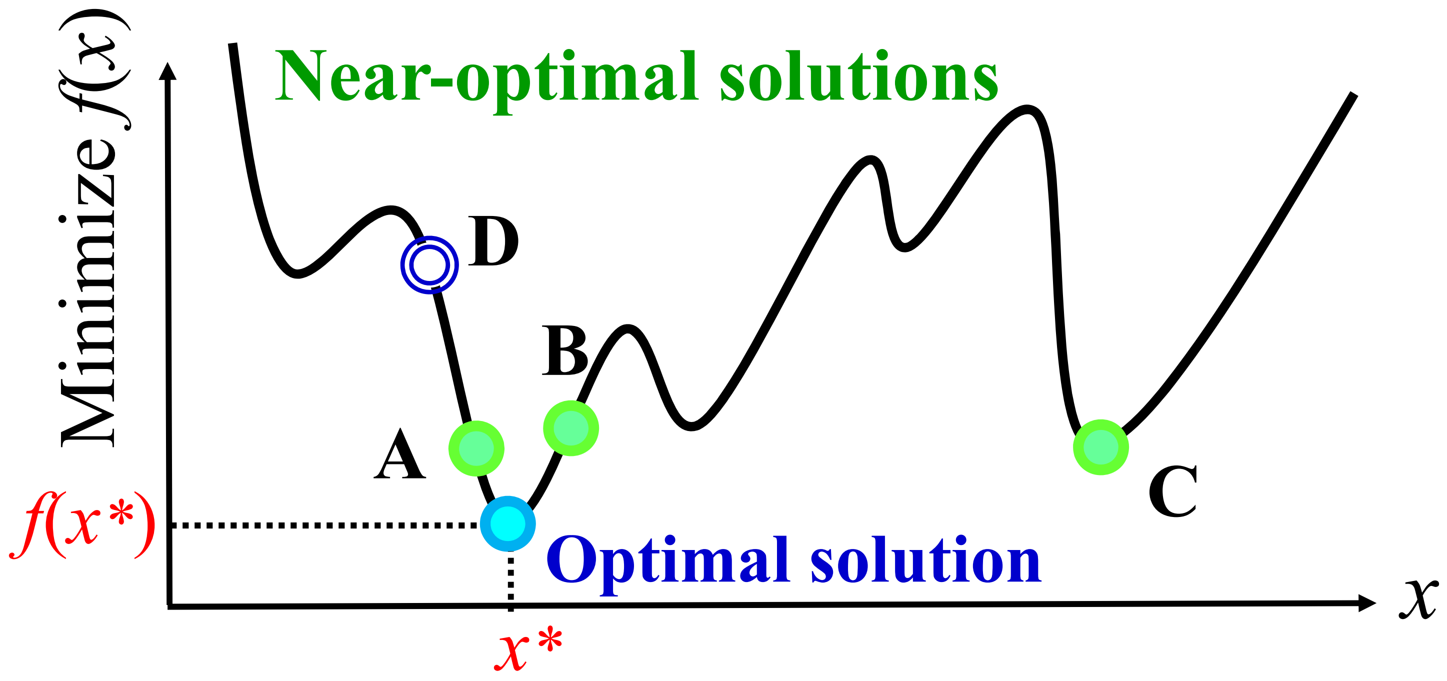
**Near-optimal solution**

A near-optimal solution is a solution which is close to the optimal solution. It seems to me that there is no clear mathematical definition of near-optimal solutions (i.e., there is no clear mathematical definition of "near"). Near-optimal solutions usually means good or very good solutions.



## Terminology

**Near-optimal solution**

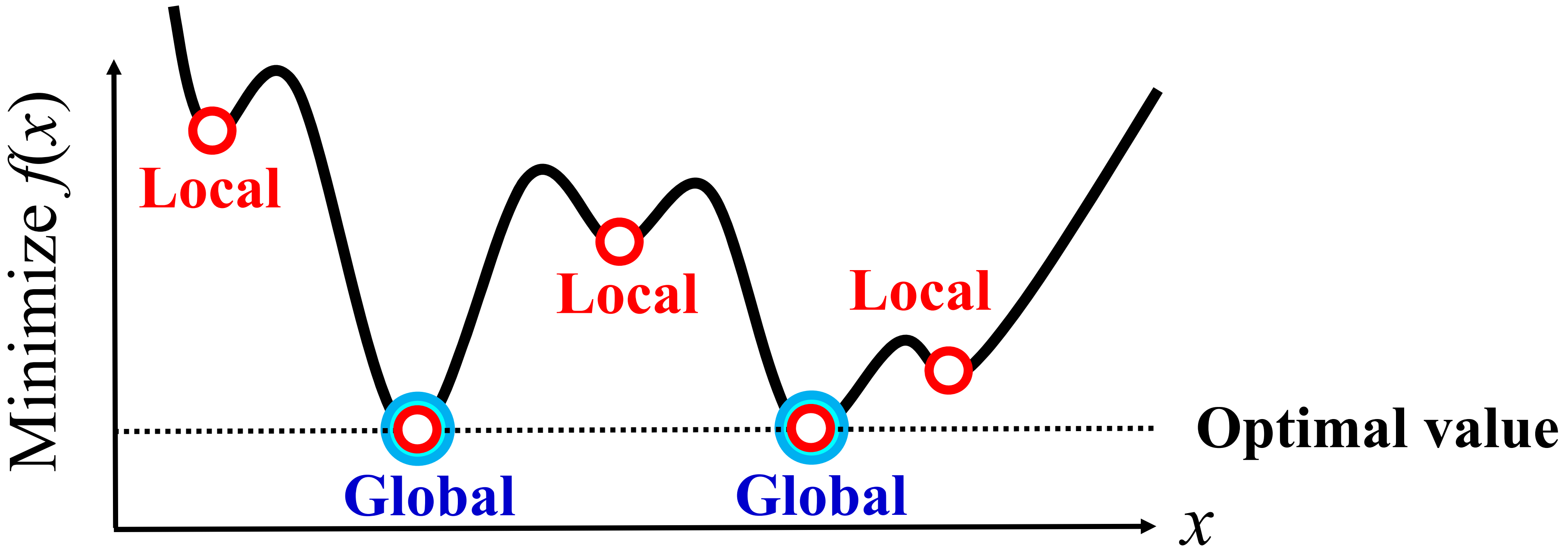


A and B are near optimal since they are close to the optimal solution (i.e., to both $f(x^*)$ and $x^*$). C is near-optimal since its objective value is close to $f(x^*)$ whereas it is not close to $x^*$. D is not near-optimal since its objective value is not close to $f(x^*)$ whereas it is close to $x^*$.

## Note 1:

## Globally optimal solution:

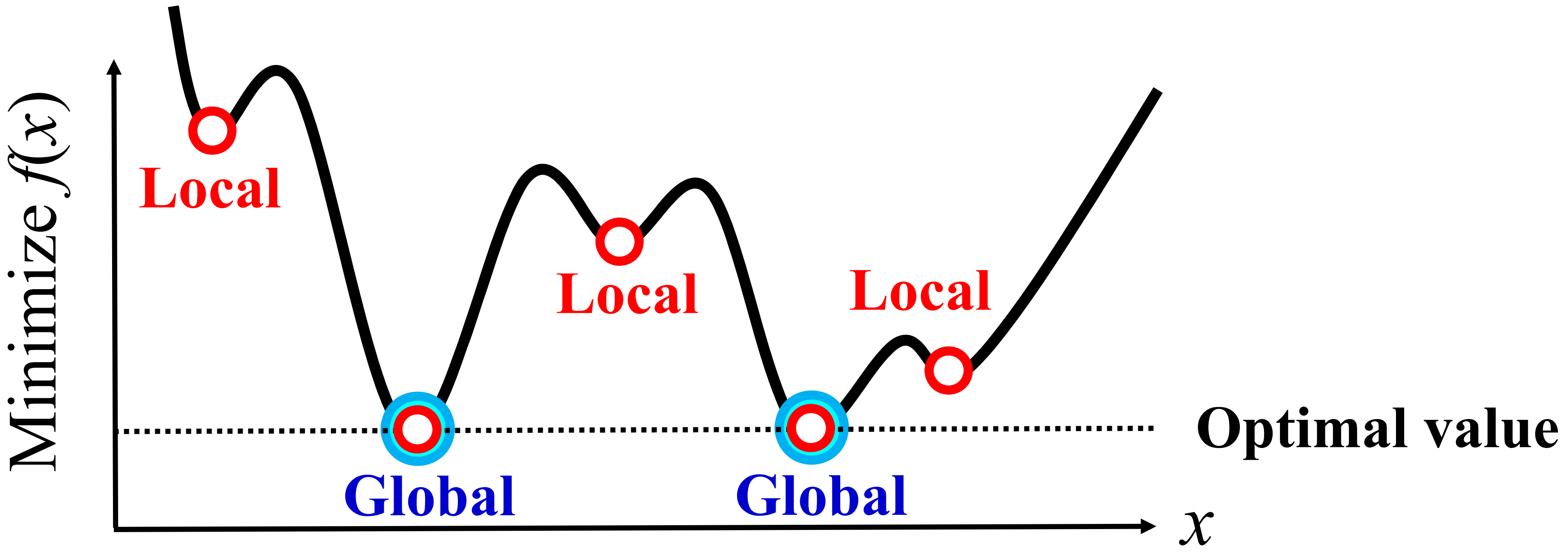
**An optimization problem can have multiple global solutions.**



## Note 1:

## Globally optimal solution:

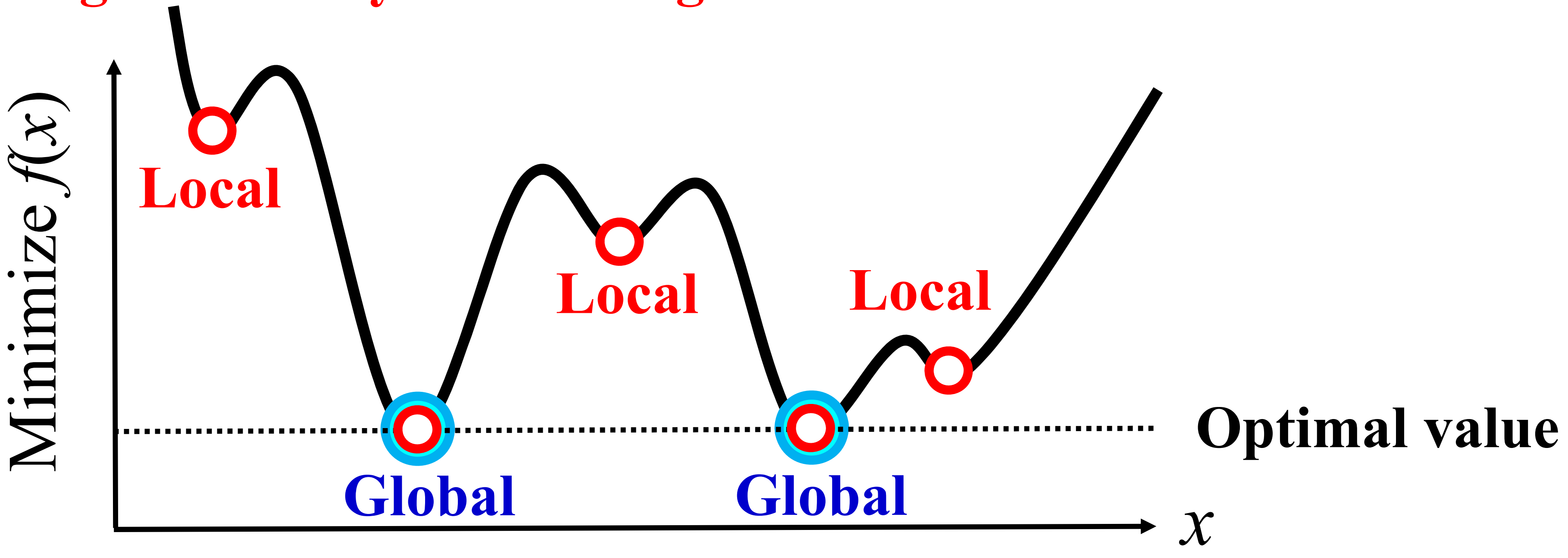
**An optimization problem can have multiple global solutions. Such an optimization problem is often called a multi-modal optimization problem where the task of an optimization algorithm is to find all global solutions.**



## Note 1:

## Globally optimal solution:

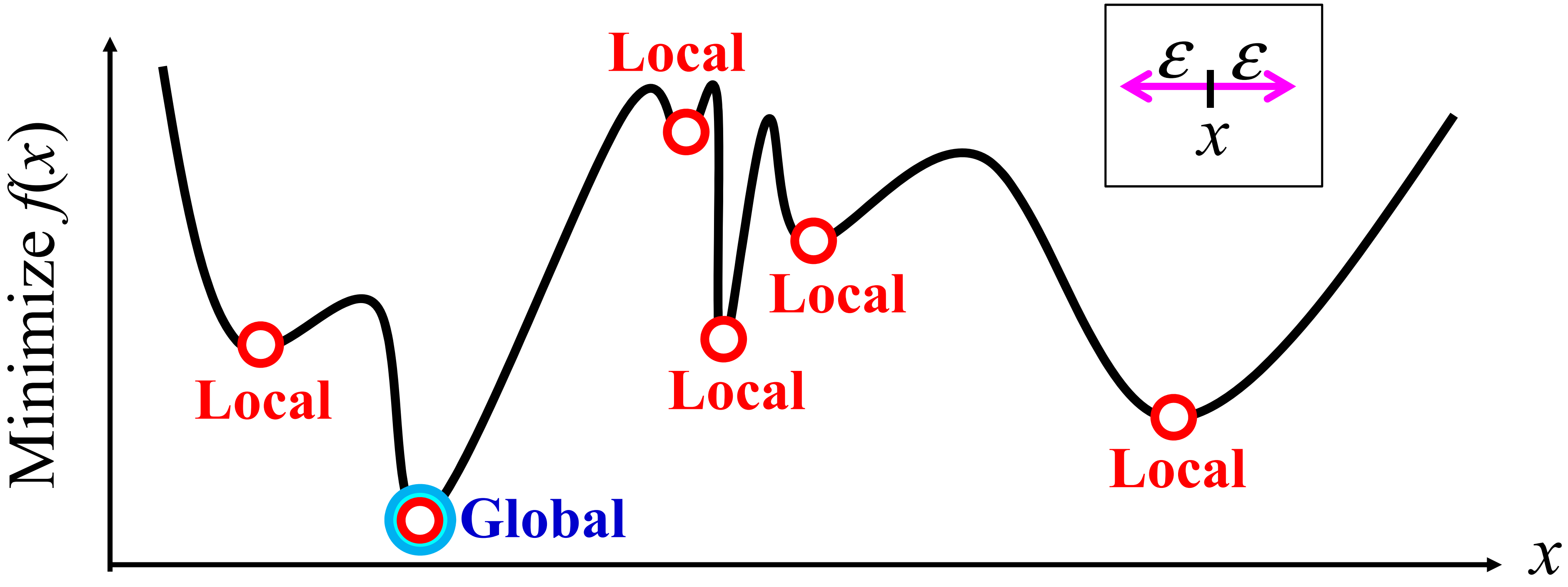
**An optimization problem can have multiple global solutions. Such an optimization problem is often called a multi-modal optimization problem where the task of an optimization algorithm is to find all global solutions. Some multi-modal algorithms try to find all global and local solutions.**



## Note 2:

## Locally optimal solution:

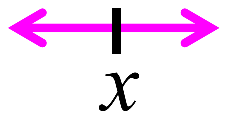
**The definition of local solutions depends on the definition of the neighborhood structure (especially in the case of combinatorial optimization). In continuous optimization, we usually define local solutions by assuming a very small neighborhood size $[x-\varepsilon, x+\varepsilon]$, $\varepsilon \to 0$.**



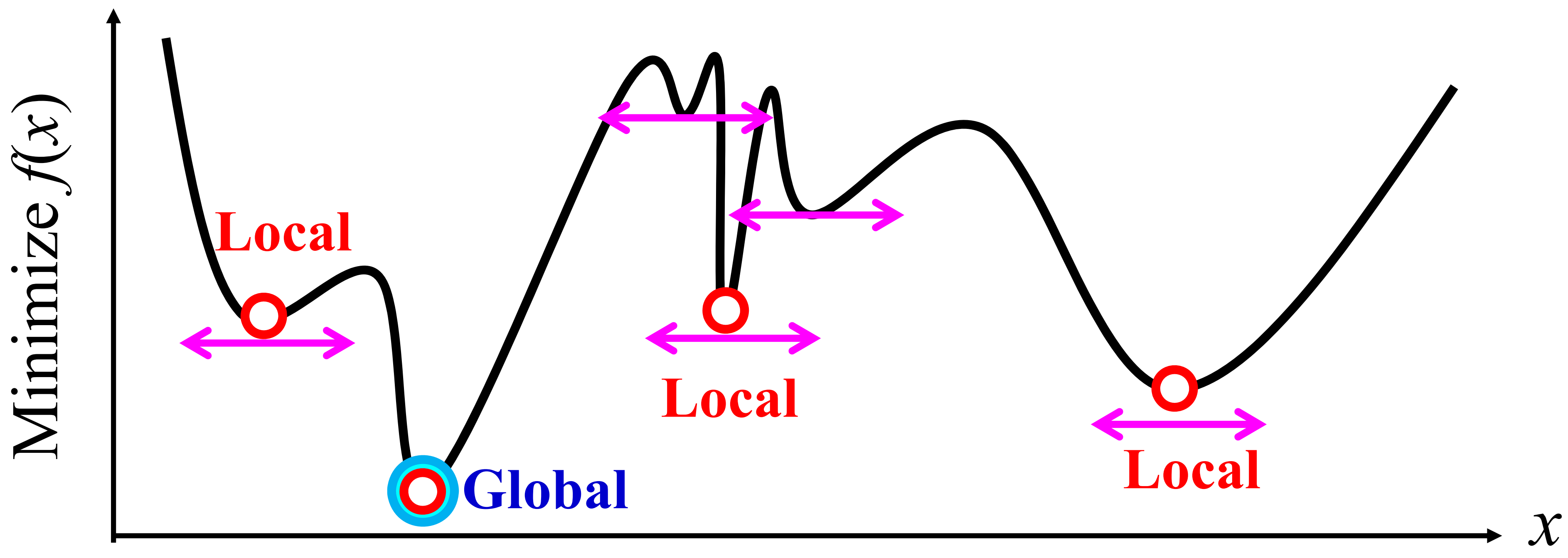
## Note 2:

## Locally optimal solution:

**If we use a large neighborhood, some solutions are not local solutions. For example, if the neighborhood size is**



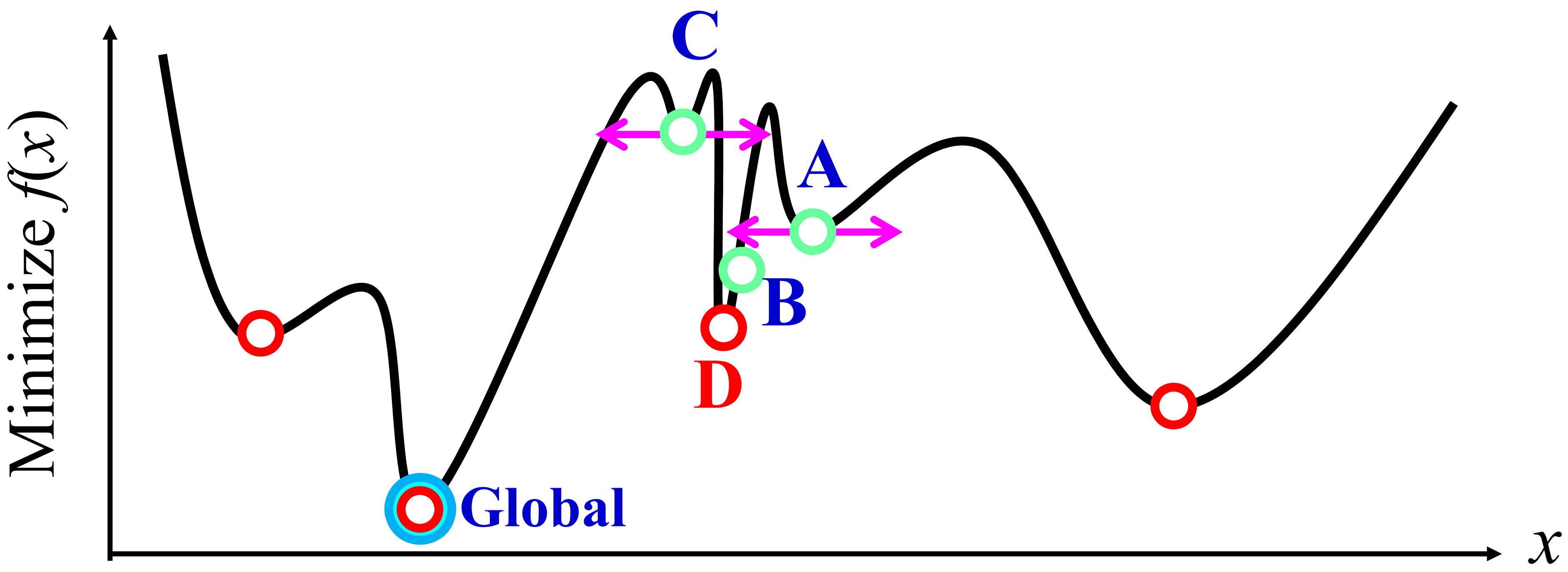
**the following are local solutions.**



## Note 2:

## Locally optimal solution:

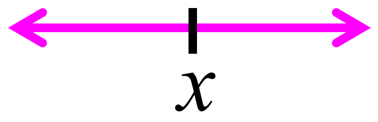
**If we use a large neighborhood, some solutions are not local solutions. For example, if the neighborhood size is $\underset{x}{\longleftrightarrow}$ the following are local solutions.**



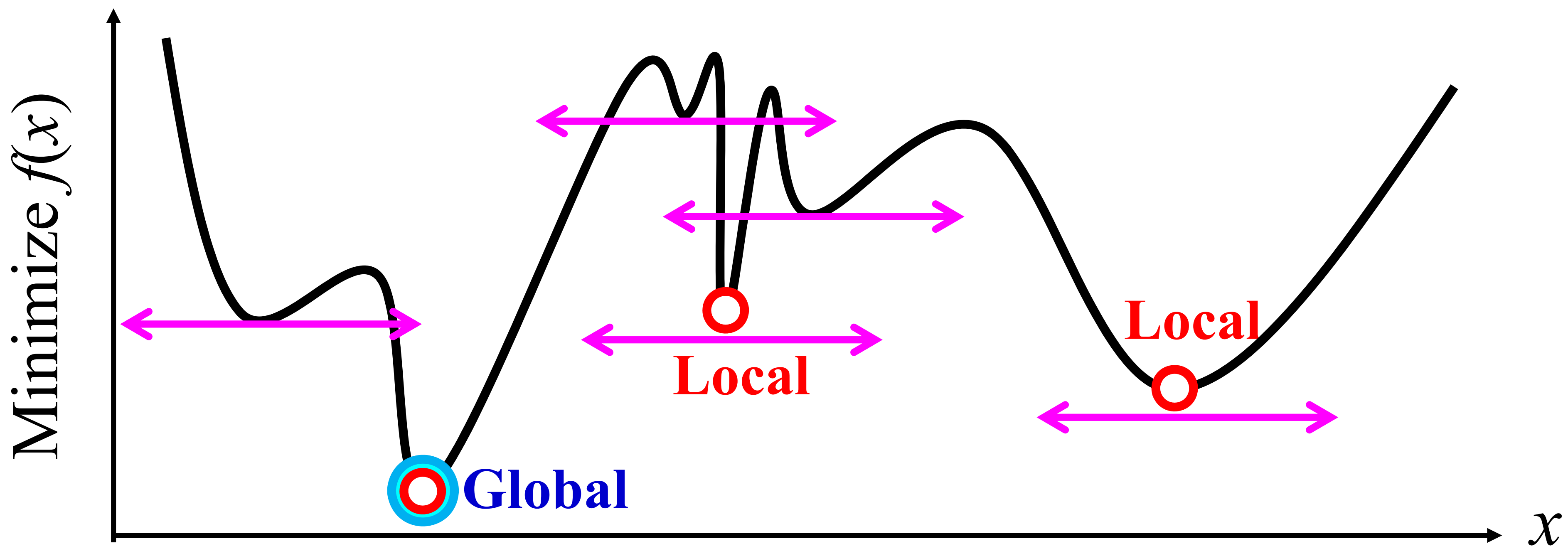
**Note 2:**

## Locally optimal solution:

**If we use a large neighborhood, some solutions are not local solutions. For example, if the neighborhood size is**



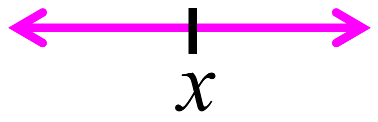
**the following are local solutions.**



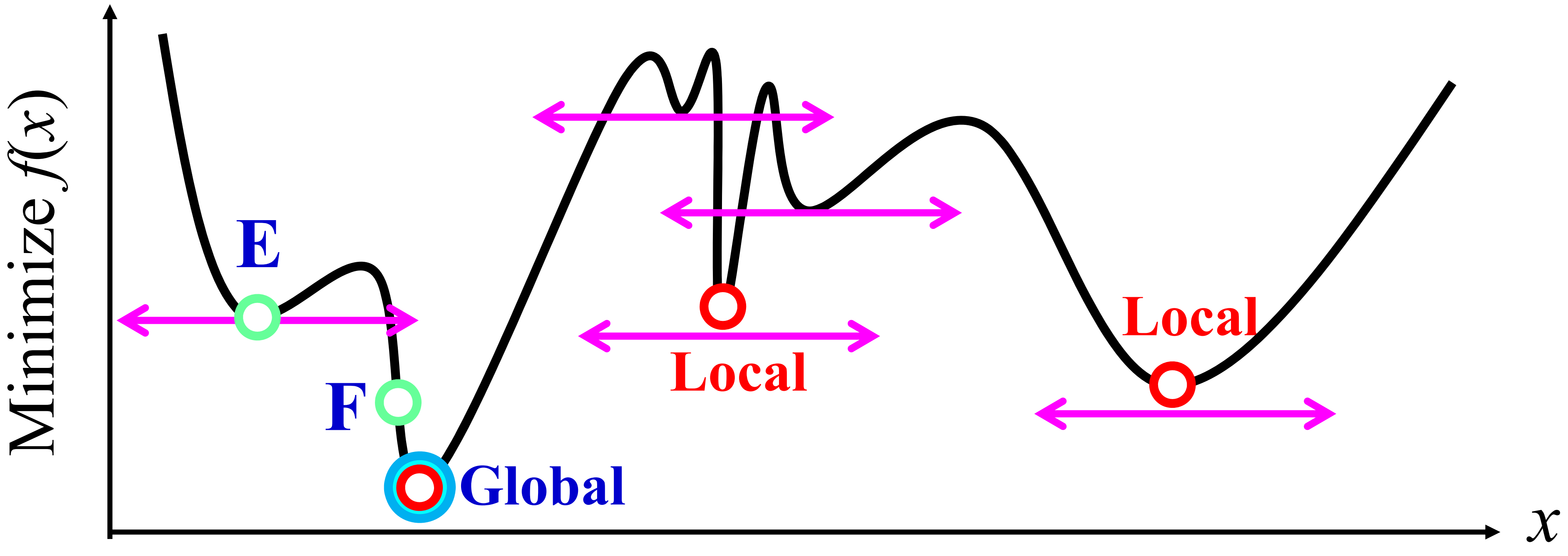
## Note 2:

## Locally optimal solution:

**If we use a large neighborhood, some solutions are not local solutions. For example, if the neighborhood size is**



**the following are local solutions.**



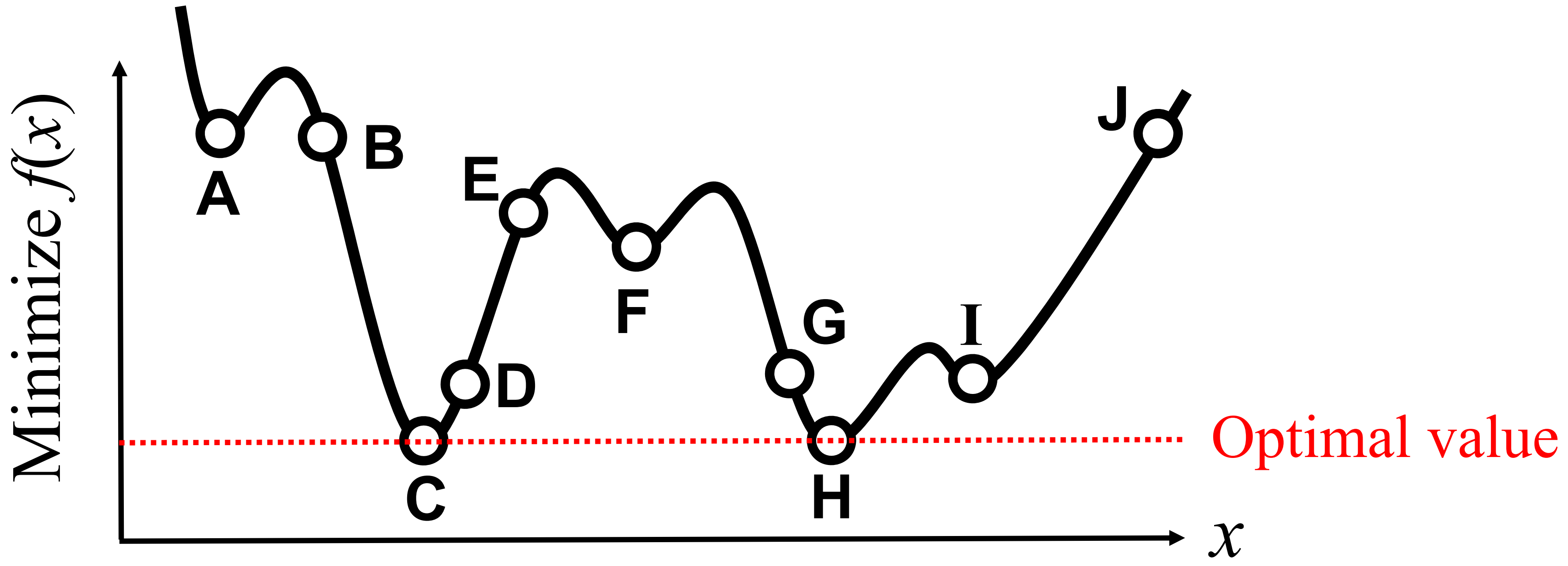
**Q1**. Which are globally optimal solutions ?

**Q2**. Which are locally optimal solutions ?

**Q3.** Which are near-optimal solutions ?

**Q4.** Which can be approximate solutions ?

Your answer: **Q1: ???, Q2: ???, Q3: ???. Q4: ???**



## Optimization Methods

## Knapsack Problem

**Input:** Item set: $n$ items

Value of each item: $v_i$ ($i = 1, 2, ..., n$)

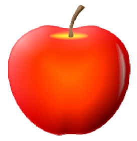
Weight of each item: $w_i$ ($i = 1, 2, ..., n$)

Weight capacity of the knapsack: $W$

**Objective:** Maximize the total value of the selected items

**Output:** Selected item set: $S \subset \{1, 2, ..., n\}$

**Constraint:** $\sum_{i \in S} w_i \leq W$



| Item ($i$) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| value ($v_i$) | 4 $ | 5 $ | 12 $ | 14 $ |
| weight ($w_i$) | 2 kg | 3 kg | 5 kg | 6 kg |

$W = 9$ kg

## Knapsack Problem: Formulation

$$\text{Maximize } V = \sum_{i \in S} v_i \quad \text{subject to} \quad \sum_{i \in S} w_i \le W$$

$$\text{Maximize } f(\boldsymbol{x}) = \sum_{i=1}^{n} v_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \le W \quad \text{and} \quad x_i \in \{0, 1\},\ i = 1, 2, \dots, n$$

**$W$ = 9 kg**

| Item ($i$) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| value ($v_i$) | 4 \$ | 5 \$ | 12 \$ | 14 \$ |
| weight ($w_i$) | 2 kg | 3 kg | 5 kg | 6 kg |

## Knapsack Problem: Formulation

$$\text{Maximize } V = \sum_{i \in S} v_i \quad \text{subject to} \quad \sum_{i \in S} w_i \le W$$

$$\text{Maximize } f(\boldsymbol{x}) = \sum_{i=1}^{n} v_i x_i$$

$$\text{subject to } \sum_{i=1}^{n} w_i x_i \le W \quad \text{and} \quad x_i \in \{0, 1\},\ i = 1, 2, \ldots, n$$

***W* = 9 kg**

| Item ($i$) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| value ($v_i$) | 4 $ | 5 $ | 12 $ | 14 $ |
| weight ($w_i$) | 2 kg | 3 kg | 5 kg | 6 kg |

**What is the optimal solution ?**

**Selected Item Set S* = {______}**

## Greedy Algorithm

**Q1.** How to design a greedy algorithm?

**A.** Choose good items one by one.

**Q2.** How to evaluate each item ? Which is the best item ?

**$W$ = 9 kg**

| Item ($i$) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| value ($v_i$) | 4 $ | 5 $ | 12 $ | 14 $ |
| weight ($w_i$) | 2 kg | 3 kg | 5 kg | 6 kg |

## Greedy Algorithm

**Q1.** How to design a greedy algorithm?

**A.** Choose good items one by one.

**Q2.** How to evaluate each item ? **Which is the best item ?**

**Your Answer: Item ?**

***W* = 9 kg**

| Item ($i$) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| value ($v_i$) | 4 \$ | 5 \$ | 12 \$ | 14 \$ |
| weight ($w_i$) | 2 kg | 3 kg | 5 kg | 6 kg |

## Greedy Algorithm

**Q1.** How to design a greedy algorithm?

**A.** Choose good items one by one.

**Q2.** How to evaluate each item ? Which is the best item ?

**Good item:** large value & small weight

**Bad item:** small value & large weight

**$W$ = 9 kg**

| Item ($i$) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| value ($v_i$) | 4 \$ | 5 \$ | 12 \$ | 14 \$ |
| weight ($w_i$) | 2 kg | 3 kg | 5 kg | 6 kg |

## Greedy Algorithm

Q1. How to design a greedy algorithm?

A. Choose good items one by one.

Q2. How to evaluate each item ? Which is the best item ?

**Select items in a decreasing order of value/weight ($v_i/w_i$).**

[Point] Examine all items without stopping the algorithm.

$W = 9$ kg

|  | **(3)** | **(4)** | **(1)** | **(2)** |
|---|---|---|---|---|
| Item ($i$) | 1 | 2 | 3 | 4 |
| value ($v_i$) | 4 $ | 5 $ | 12 $ | 14 $ |
| weight ($w_i$) | 2 kg | 3 kg | 5 kg | 6 kg |
| value/weight | 2 | 1.67 | 2.4 | 2.33 |

## Greedy Algorithm

Q1. How to design a greedy algorithm?

A. Choose good items one by one.

Q2. How to evaluate each item ? Which is the best item ?

**Select items in a decreasing order of value/weight ($v_i/w_i$).**

[Point] Examine all items without stopping the algorithm.

(1) Item 3: 5kg (OK) 12$ (Item 3)

(2) Item 4: 5kg + 6kg (NG) 12$ (Item 3)

(3) Item 1: 5kg + 2kg (OK) 16$ (Item 1 & Item 3)

(4) Item 2: 5kg + 2kg + 3kg (NG) 16$ (Item 1 & Item 3)

$W = 9$ **kg**

| Item ($i$) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| value ($v_i$) | 4 $ | 5 $ | 12 $ | 14 $ |
| weight ($w_i$) | 2 kg | 3 kg | 5 kg | 6 kg |
| value/weight | 2 | 1.67 | 2.4 | 2.33 |

## Greedy Algorithm

Select items in a decreasing order of value/weight ($v_i/w_i$).

(1) Item 3: 5kg (OK)    12$ (Item 3)

(2) Item 4: 5kg + 6kg (OK)   26$ (Item 3 & Item 4)

***W* = 11 kg**

| Item ($i$) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| value ($v_i$) | 4 $ | 5 $ | 12 $ | 14 $ |
| weight ($w_i$) | 2 kg | 3 kg | 5 kg | 6 kg |
| value/weight | 2 | 1.67 | 2.4 | 2.33 |

**One Clear Feature of the Greedy Algorithm:**
If the total weight of the selected items is the same as the weight capacity *W* with no item skip, the greedy solution is the optimal solution.

## Knapsack Problem: Formulation

$$\text{Maximize } f(\boldsymbol{x}) = \sum_{i=1}^{n} v_i x_i$$
$$\text{Subject to } \sum_{i=1}^{n} w_i x_i \le W \quad \text{and} \quad x_i \in \{0, 1\},\ i = 1, 2, \ldots, n$$

**Greedy Algorithm**

Select items in a decreasing order of value/weight ($v_i/w_i$).

**Question:** How can we generalize this algorithm to the case of multiple constraint conditions?

$$\text{Subject to } \sum_{i=1}^{n} w_{ij} x_i \le W_j,\ j = 1, 2, \ldots, m,$$
$$\text{and} \quad x_i \in \{0, 1\},\ i = 1, 2, \ldots, n$$

**Example:** weight and size (volume)

Constraint 1: Total weight (weight constraint)
Constraint 2: Total volume (size constraint)

## Knapsack Problem with Multiple Constraint Conditions

$$\text{Maximize } f(\boldsymbol{x}) = \sum_{i=1}^{n} v_i x_i$$

$$\text{Subject to } \sum_{i=1}^{n} w_{ij} x_i \le W_j,\ j = 1, 2, \ldots, m,$$

$$\text{and } x_i \in \{0, 1\},\ i = 1, 2, \ldots, n$$

**Greedy Algorithm**

Select items in a decreasing order of ____**?**____ .

## Knapsack Problem with Multiple Constraint Conditions

$$\text{Maximize} \quad f(\boldsymbol{x}) = \sum_{i=1}^{n} v_i x_i$$

$$\text{Subject to} \sum_{i=1}^{n} w_{ij} x_i \le W_j, \; j = 1, 2, \ldots, m,$$

$$\text{and} \quad x_i \in \{0, 1\}, \; i = 1, 2, \ldots, n$$

**Greedy Algorithm**

Select items in a decreasing order of ___**?**___ .

**Q.** How to evaluate each item.

$$\boxed{\textbf{?}}$$

## Knapsack Problem with Multiple Constraint Conditions

$$\text{Maximize } f(\boldsymbol{x}) = \sum_{i=1}^{n} v_i x_i$$

$$\text{Subject to } \sum_{i=1}^{n} w_{ij} x_i \le W_j,\ j = 1, 2, \ldots, m,$$

$$\text{and } x_i \in \{0, 1\},\ i = 1, 2, \ldots, n$$

**Greedy Algorithm**

Select items in a decreasing order of ___?___ .

**Q.** How to evaluate each item.

$$\frac{v_i}{\sum_{j=1}^{m} w_{ij}}, \quad \operatorname*{Max}_{j=1,\ldots,m} \frac{v_i}{w_{ij}}, \quad \operatorname*{Min}_{j=1,\ldots,m} \frac{v_i}{w_{ij}}, \quad \operatorname*{Average}_{j=1,\ldots,m} \frac{v_i}{w_{ij}}$$

## LP Relaxation

$$\text{Maximize } g(\boldsymbol{x}) = \sum_{i=1}^{n} x_i v_i$$

$$\text{Subject to } \sum_{i=1}^{n} x_i w_i \leq W \quad \text{and} \quad 0 \leq x_i \leq 1,\ i = 1,\ 2,\ \ldots,\ n$$

## LP Solution: 21.33$

(Usually this is not a feasible solution)

Item 3: 5kg 12$

Item 4: 4kg (4/6 of 6kg) 9.33$ (4/6 of 14$)

$W = 9$ kg

| Item ($i$) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| value ($v_i$) | 4 $ | 5 $ | 12 $ | 14 $ |
| weight ($w_i$) | 2 kg | 3 kg | 5 kg | 6 kg |
| value/weight | 2 | 1.67 | 2.4 | 2.33 |

## LP Relaxation: Upper Bound of Knapsack Problem

Maximize $g(\boldsymbol{x}) = \sum_{i=1}^{n} x_i v_i$

Subject to $\sum_{i=1}^{n} x_i w_i \le W$ and $0 \le x_i \le 1,\ i = 1, 2, \ldots, n$

$\boldsymbol{x}^{\mathrm{LP}}$: Optimal solution of the LP relaxation problem

$g(\boldsymbol{x}^{\mathrm{LP}})$: Optimal value of the LP relaxation problem

$f(\boldsymbol{x}^*)$: Optimal value of the knapsack problem $\boxed{f(\boldsymbol{x}^*) \le g(\boldsymbol{x}^{\mathrm{LP}})}$

### How to create a feasible solution from $x^{\mathrm{LP}}$:

The *i*th item is selected if and only if $x_i^{\mathrm{LP}} = 1$:

$$y_i = \begin{cases} 1, \text{ if } x_i^{\mathrm{LP}} = 1, \\ 0, \text{ if } x_i^{\mathrm{LP}} < 1, \end{cases} \quad i = 1, 2, \ldots, n.$$

$$\boxed{f(\boldsymbol{y}) \le f(\boldsymbol{x}^*) \le g(\boldsymbol{x}^{\mathrm{LP}})}$$

## LP Relaxation: Upper Bound of Knapsack Problem

Maximize $g(\boldsymbol{x}) = \sum_{i=1}^{n} x_i v_i$

Subject to $\sum_{i=1}^{n} x_i w_i \le W$ and $0 \le x_i \le 1,\ i = 1, 2, \ldots, n$

$\boldsymbol{x}^{\mathrm{LP}}$: Optimal solution of the LP relaxation problem

$g(\boldsymbol{x}^{\mathrm{LP}})$: Optimal value of the LP relaxation problem

$f(\boldsymbol{x}^*)$: Optimal value of the knapsack problem $\boxed{f(\boldsymbol{x}^*) \le g(\boldsymbol{x}^{\mathrm{LP}})}$

### How to create a feasible solution from $x^{\mathrm{LP}}$:

The $i$th item is selected if and only if $x_i^{\mathrm{LP}} = 1$:

$$y_i = \begin{cases} 1, \text{ if } x_i^{\mathrm{LP}} = 1, \\ 0, \text{ if } x_i^{\mathrm{LP}} < 1, \end{cases} \quad i = 1, 2, \ldots, n.$$

$$\boxed{f(\boldsymbol{y}) \le f(\boldsymbol{x}^*) \le g(\boldsymbol{x}^{\mathrm{LP}})}$$

$\boxed{\text{If } \boldsymbol{y} = \boldsymbol{x}^{\mathrm{LP}}, \text{ then } \boldsymbol{y} \text{ is optimal}}$

## Relation between the greedy solution $\boldsymbol{x}$ and the feasible solution $\mathbf{y}$ created from the LP solution $\boldsymbol{y}^{\mathrm{LP}}$

If we terminate the iteration here (i.e., at the first NG), the greedy solution $\boldsymbol{x}$ is always the same as $\boldsymbol{y}$ from LP.

**Greedy Algorithm**

Select items in a decreasing order of value/weight ($v_i/w_i$).

[Point] Examine all items without stopping the algorithm.

(1) Item 3: 5kg (OK)  12$ (Item 3)
(2) Item 4: 5kg + **6kg (NG)**  12$ (Item 3)
(3) Item 1: 5kg + 2kg (OK)  16$ (Item 1 & Item 3)
(4) Item 2: 5kg + 2kg + 3kg (NG)  16$ (Item 1 & Item 3)

$W = 9$ **kg**

| Item ($i$) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| value ($v_i$) | 4 $ | 5 $ | 12 $ | 14 $ |
| weight ($w_i$) | 2 kg | 3 kg | 5 kg | 6 kg |
| value/weight | 2 | 1.67 | 2.4 | 2.33 |

## Knapsack Problem with Multiple Constraint Conditions

$$\text{Maximize } f(\boldsymbol{x}) = \sum_{i=1}^{n} v_i x_i$$

$$\text{Subject to } \sum_{i=1}^{n} w_{ij} x_i \le W_j,\ j = 1, 2, \ldots, m,$$

$$\text{and } x_i \in \{0, 1\},\ i = 1, 2, \ldots, n$$

## LP-Relaxation Problem

$$\text{Maximize } f(\boldsymbol{x}) = \sum_{i=1}^{n} v_i x_i$$

$$\text{Subject to } \sum_{i=1}^{n} w_{ij} x_i \le W_j,\ j = 1, 2, \ldots, m,$$

$$\text{and } 0 \le x_i \le 1,\ i = 1, 2, \ldots, n$$

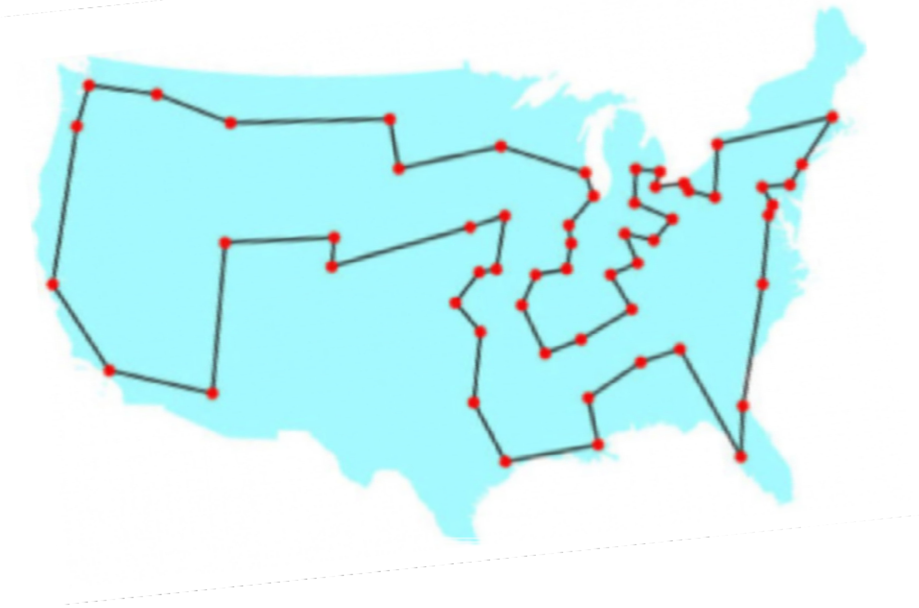
**Travelling Salesman Problem (TSP): To find the shortest tour to visit all cities and return to the start city.**

**Input:** City set: $n$ cities ($i = 1, 2, ..., n$)

Distance between each pair of cities $(i, j)$: $d_{ij} = d_{ji}$

**Objective:** Minimization of a tour length starting from a city, visiting all cities and returning to the start city.

**Output:** Tour with the minimum distance



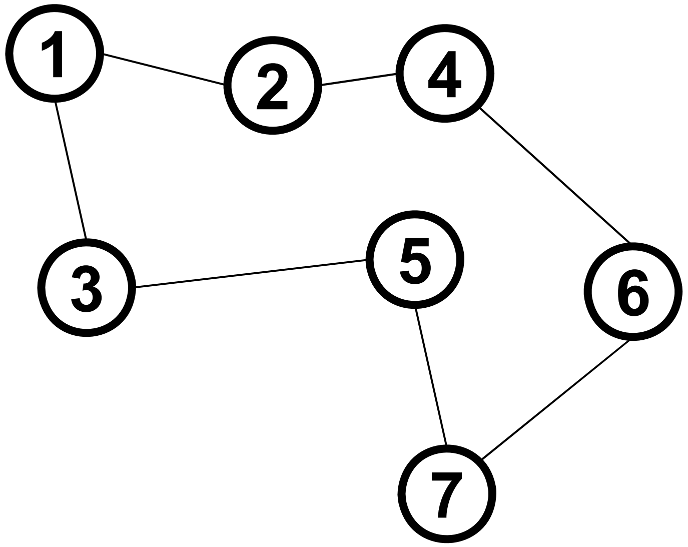
**Travelling Salesman Problem (TSP): To find the shortest tour to visit all cities and return to the start city.**

**Problem Size:** The number of cities $n$ (e.g., 100 cities)

**Search Space Size:** The number of possible combinations

The total number of permutations of $n$ cities:

$$n! = n(n - 1)(n - 2) \cdots 1$$

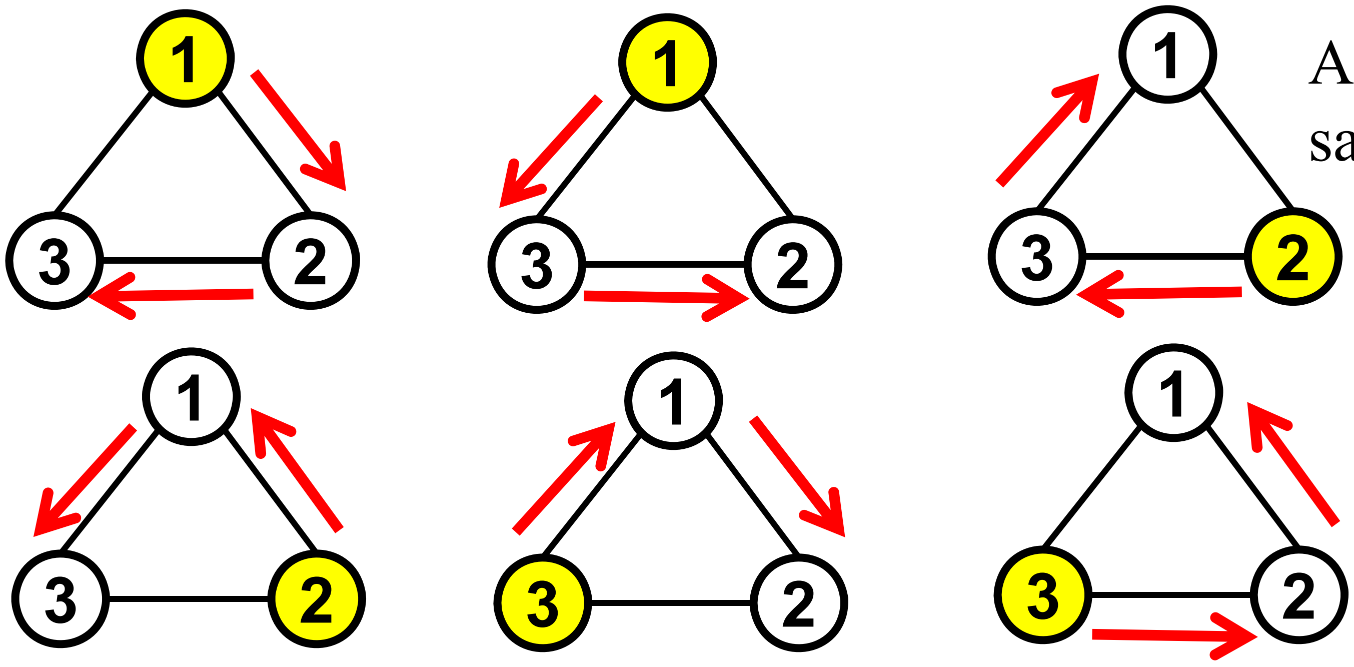


**Travelling Salesman Problem (TSP): To find the shortest tour to visit all cities and return to the start city.**

**Problem Size:** The number of cities $n$ (e.g., 100 cities)

**Search Space Size:** The number of possible combinations

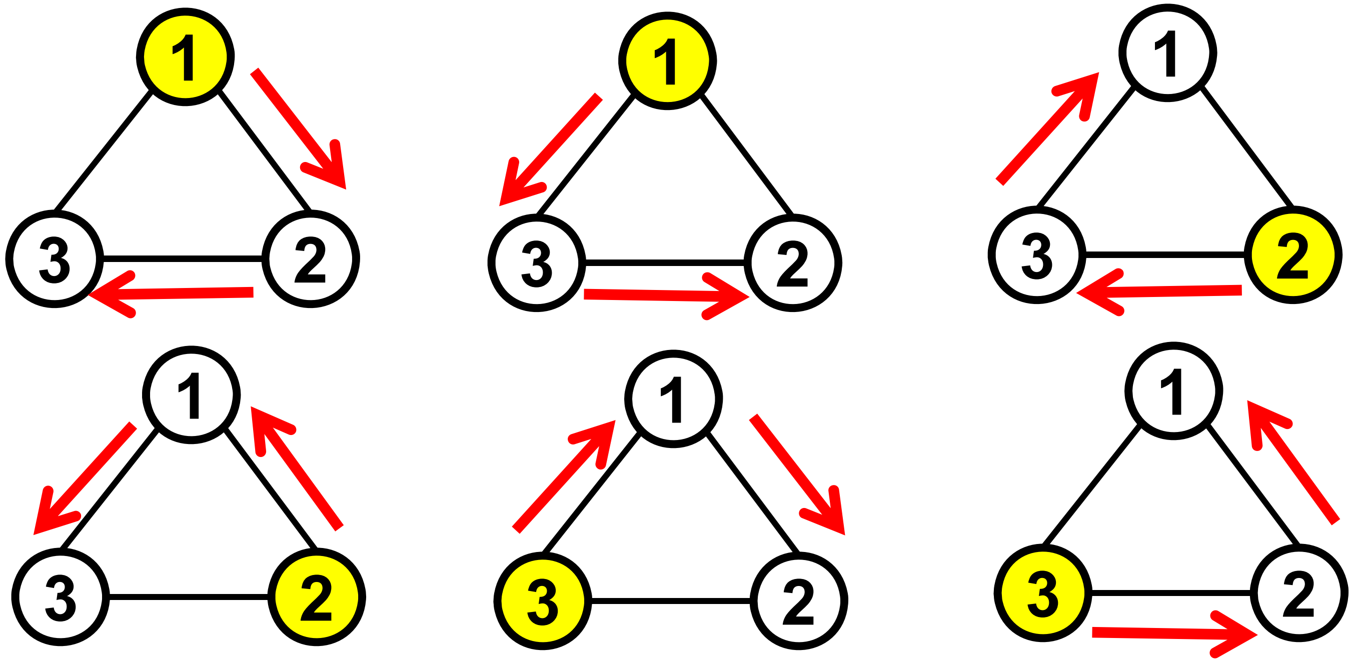
**When $n = 3$ (3 cities):** The number of permutations of 3 cities is $n! = 6$



All of them are the same solution:
**A single tour.**

When $n = 3$ (3 cities): $n! = 6$ $\Rightarrow$ **One tour.**

- We can specify one city as the start city:
  permutations of $(n - 1)$ cities. $n!$ $\Rightarrow$ $(n - 1)!$
- One permutation has the same tour length as its reverse order:
  $Length(1231) = Length(1321)$: $(n - 1)!$ $\Rightarrow$ $(n - 1)!/2$
- The search space size: **$(n - 1)!/2$**



## How large is $(n-1)!/2$ ?

$n = 3$: $(n-1)!/2 = 1$

$n = 4$: $(n-1)!/2 = 3$ (It is easy to manually examine all tours)

$n = 5$: $(n-1)!/2 = 15$ (It is easy to examine all tours)

$n = 10$: $(n-1)!/2 = 181{,}440$ (It is possible to examine all tours)

$n = 20$: $(n-1)!/2 = 6.082 \times 10^{16}$ (19,500 years by 1 million/second)

$n = 50$: $(n-1)!/2 = 3.041 \times 10^{62}$

$n = 100$: $(n-1)!/2 = 4.666 \times 10^{155}$

$n = 1{,}000$: $(n-1)!/2 =$ ...

$n = 10{,}000$: $(n-1)!/2 =$ ...

$n = 100{,}000$: $(n-1)!/2 =$ ...

Totally different algorithms are needed for $n = 10$ and $n = 100{,}000$.

## Mona Lisa TSP Challenge 100,000 Cities

http://www.math.uwaterloo.ca/tsp/data/ml/monalisa.html

Robert Bosch, February 2009
mona-lisa100K.tsp

**The current best known results:**
**Tour: 5,757,191 Gap: 107 (0.0019%)**

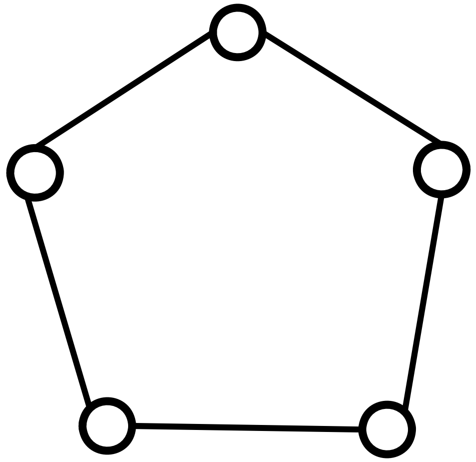
**The tour was found on March 17, 2009, by Yuichi Nagata. The bound gives a value B such that no tour has length less than B; this bound was found on July 27, 2012, with the Concorde code. The Gap number is the gap in our knowledge, that is, the difference between the length of the tour and the value of the bound.**

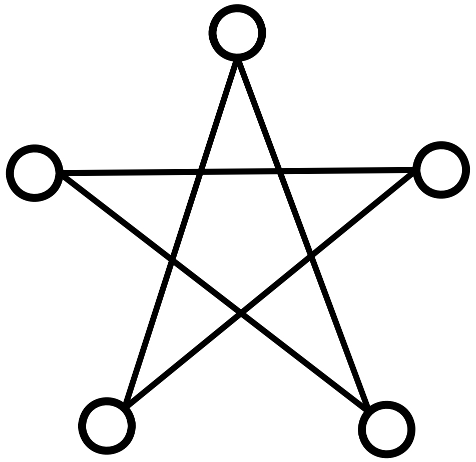
**It has been over three years since Yuichi Nagata produced the best-known tour. To help perk up interest in searching for an even better solution, we are offering a $1,000 prize to the first person to find a tour shorter than 5,757,191.**

## TSP Problem Analysis:

The worst tour among all the $(n - 1)!/2$ possible tours.

**Question:** How bad is the worst tour length among the $(n - 1)!/2$ possible tours in comparison with the optimal tour length? (e.g., 2 times, 5 times, 10 times ?)





$$\boxed{\textbf{Solution Quality Index} = \frac{\textbf{Tour Length}}{\textbf{Optimal Tour Length}}}$$

**Question:** How bad can the solution quality index be for an *n*-city problem (when $n \to \infty$) ?

$$\boxed{?}$$

$$\boxed{\textbf{Solution Quality Index} = \frac{\textbf{Tour Length}}{\textbf{Optimal Tour Length}}}$$

**Question:** How bad can the solution quality index be for an *n*-city problem (when $n \to \infty$) ?

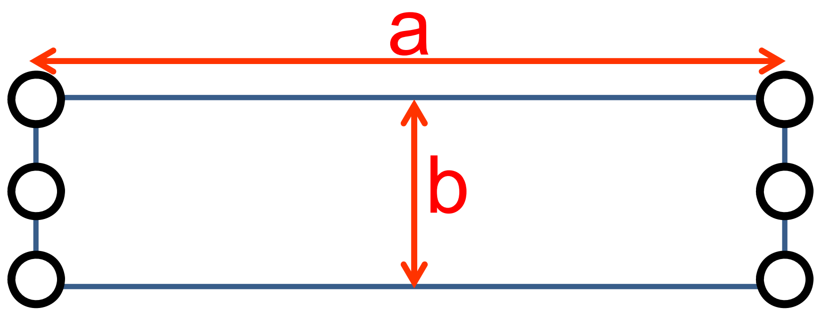
$$\boxed{\ ?\ }$$

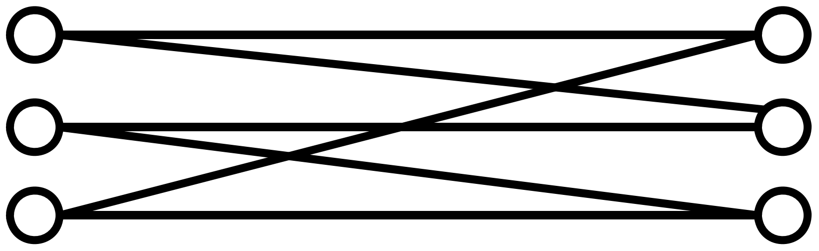
**Easier Question:** How bad can the solution quality index be for a 6-city problem?

$$\boxed{\textbf{Solution Quality Index} = \frac{\textbf{Tour Length}}{\textbf{Optimal Tour Length}}}$$

**Question:** How bad can the solution quality index be for an *n*-city problem (when $n \to \infty$) ?

Solution Quality Index $\to \infty$ when $n \to \infty$ in the worst case.





$$\text{Tour length} = 2a + 2b$$
$$\cong 2a \ \ (\text{if } b << a)$$

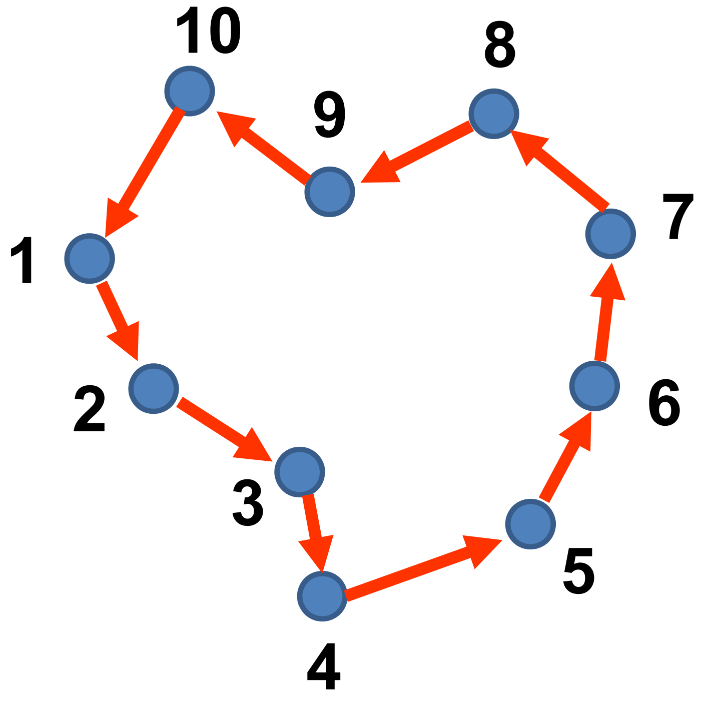
$$\text{Tour length} \cong 6a = na$$
$$\text{Solution Quality Index} \cong n/2$$

This means that a randomly generated solution can be very bad by increasing the number of cities.

$$\boxed{\textbf{Solution Quality Index} = \frac{\textbf{Tour Length}}{\textbf{Optimal Tour Length}}}$$

## Simple Nearest Neighbor Greedy Algorithm

1. Arbitrarily select a start city.
2. Move from the current city to the nearest city among the remaining cities until all cities are visited.
3. Go back to the start city.



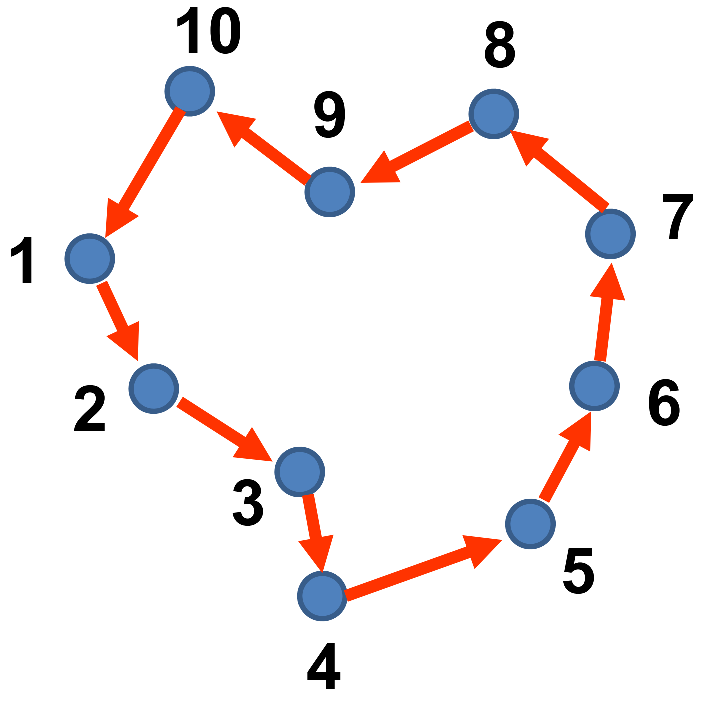
## Simple Nearest Neighbor Greedy Algorithm

1. Arbitrarily select a start city.

This looks very simple !

2. Move from the current city to the nearest city among the remaining cities until all cities are visited.

3. Go back to the start city.



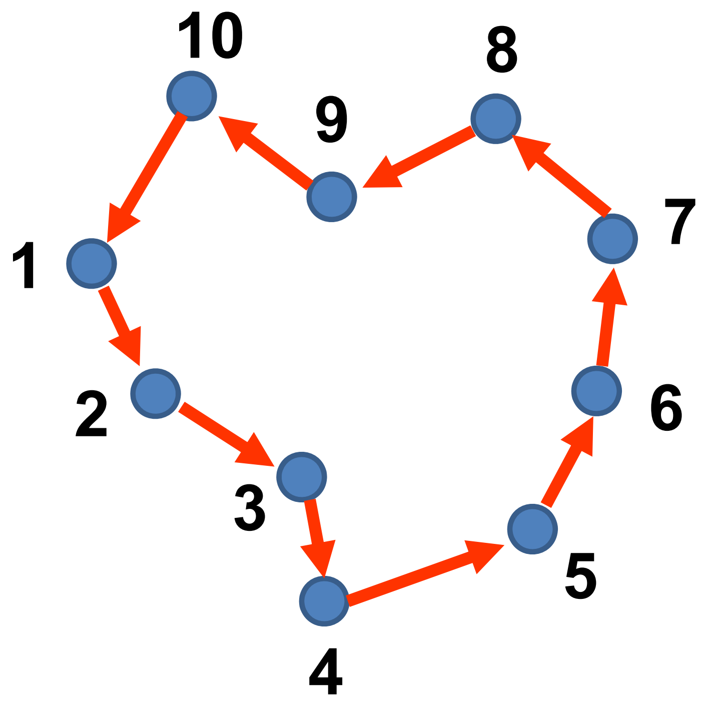
# Simple Nearest Neighbor Greedy Algorithm

1. Arbitrarily select a start city.

This looks very simple !

2. Move from the current city to the nearest city among the remaining cities until all cities are visited.

3. Go back to the start city.



Some Implementation Issues:

(a) How to choose a start city?
(random, by a rule, examine all cities?)

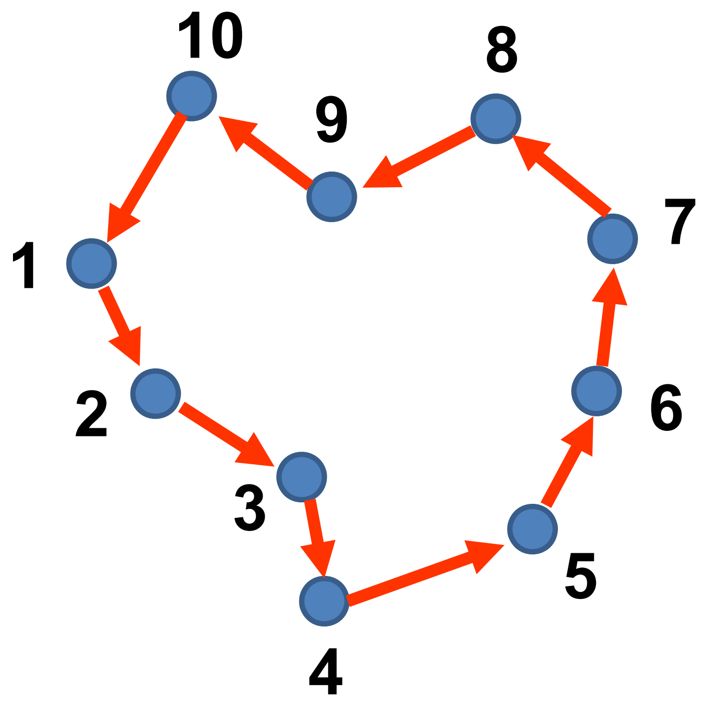
# Simple Nearest Neighbor Greedy Algorithm

1. Arbitrarily select a start city.

This looks very simple !

2. Move from the current city to the nearest city among the remaining cities until all cities are visited.

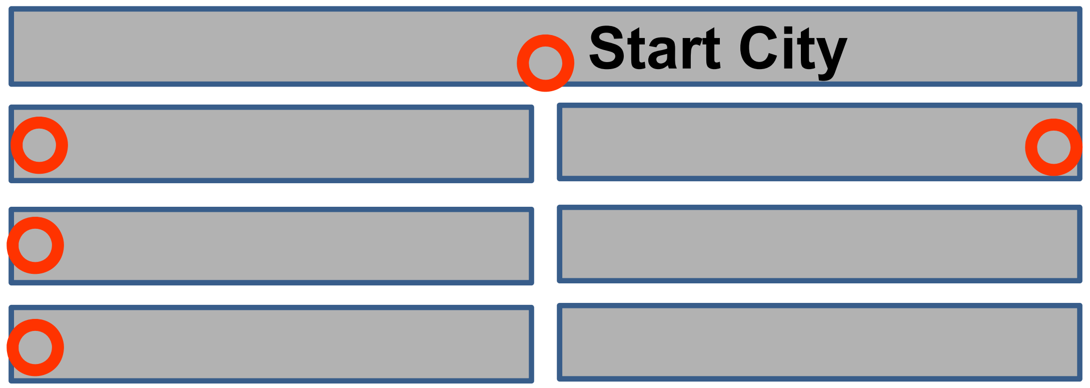
3. Go back to the start city.



Some Implementation Issues:

(a) How to choose a start city?
(random, by a rule, examine all cities?)

(b) How to choose one city if multiple cities have the same minimum distance?



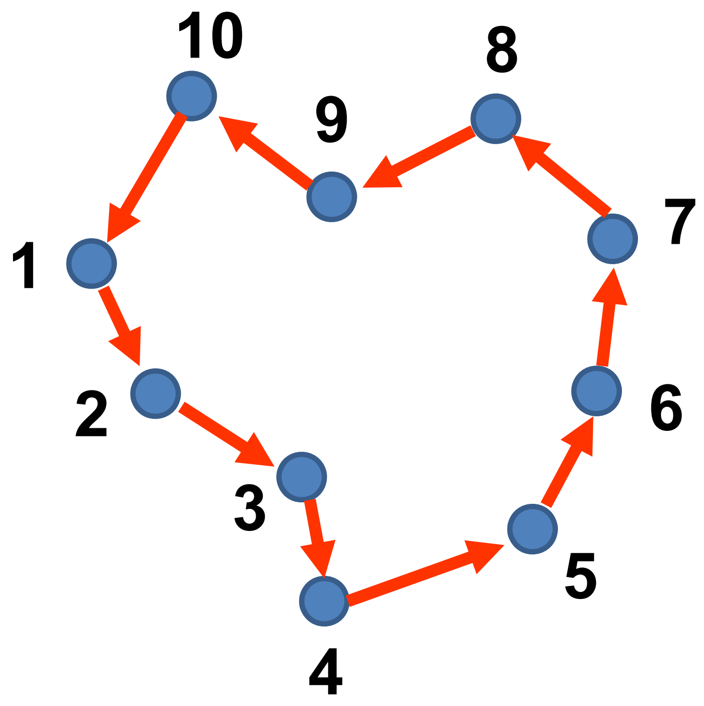
## Simple Nearest Neighbor Greedy Algorithm

1. Arbitrarily select a start city.

This looks very simple !

2. Move from the current city to the nearest city among the remaining cities until all cities are visited.

3. Go back to the start city.



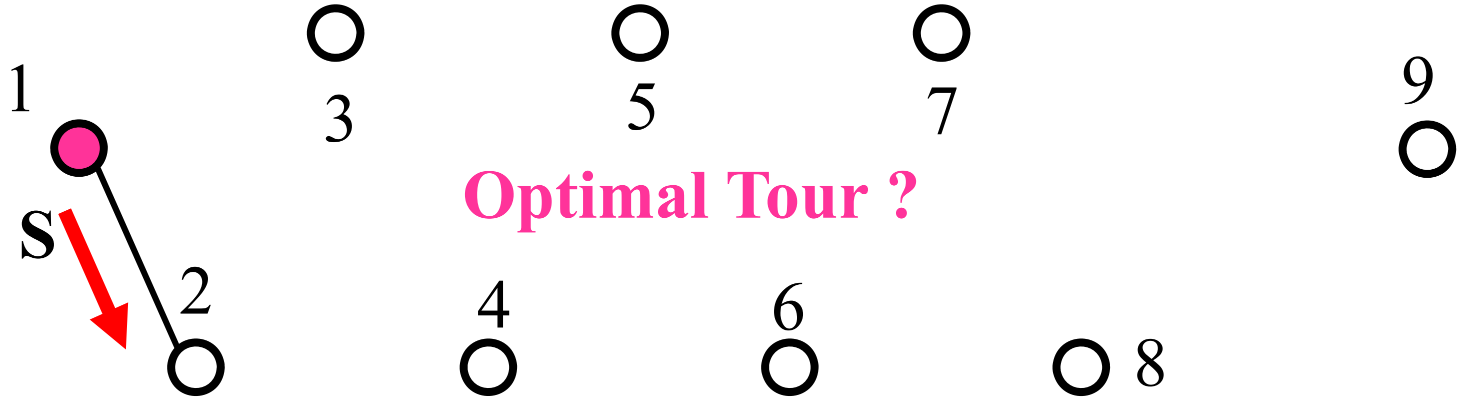
Another Issue:

How to evaluate this algorithm ?

- Average result over all start cities?
- Best result over all start cities?
- Worst result over all start cities?

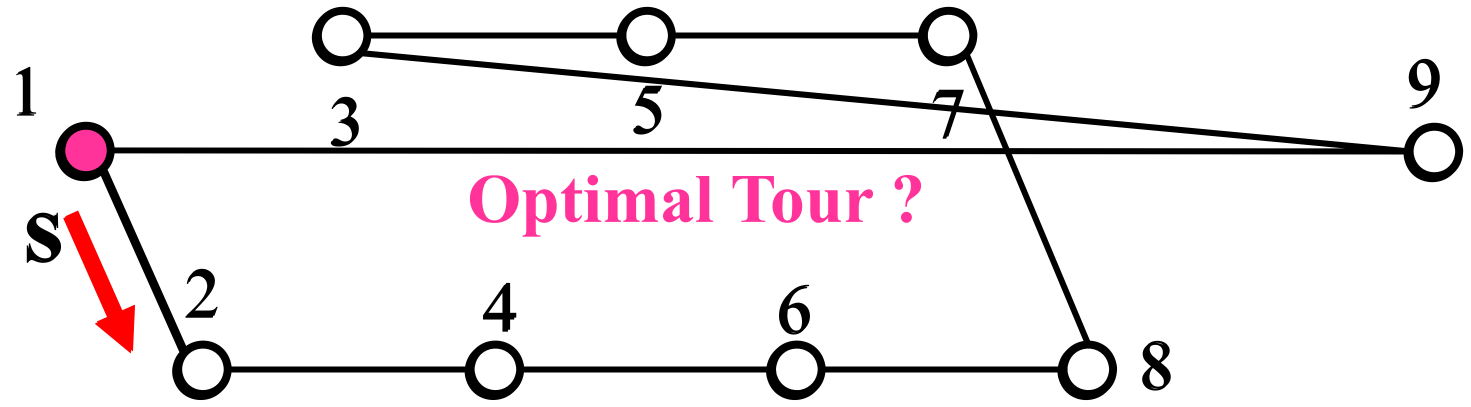
## Simple Examples

Create a greedy solution from City S (City 1).



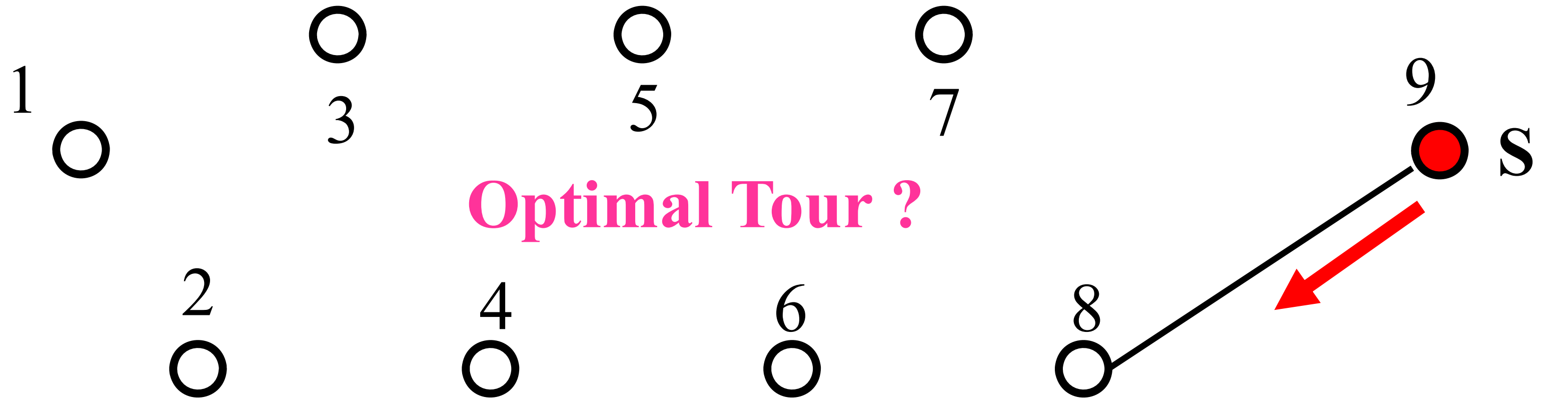
## Simple Examples

Create a greedy solution from City S (City 1).



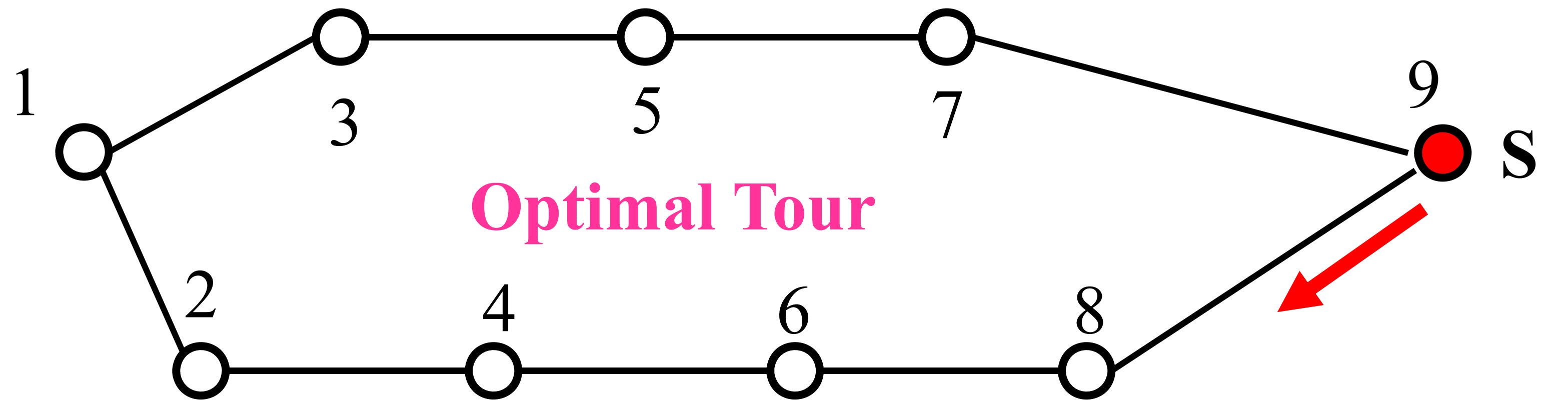
## Simple Examples

Create a greedy solution from City S (City 9).



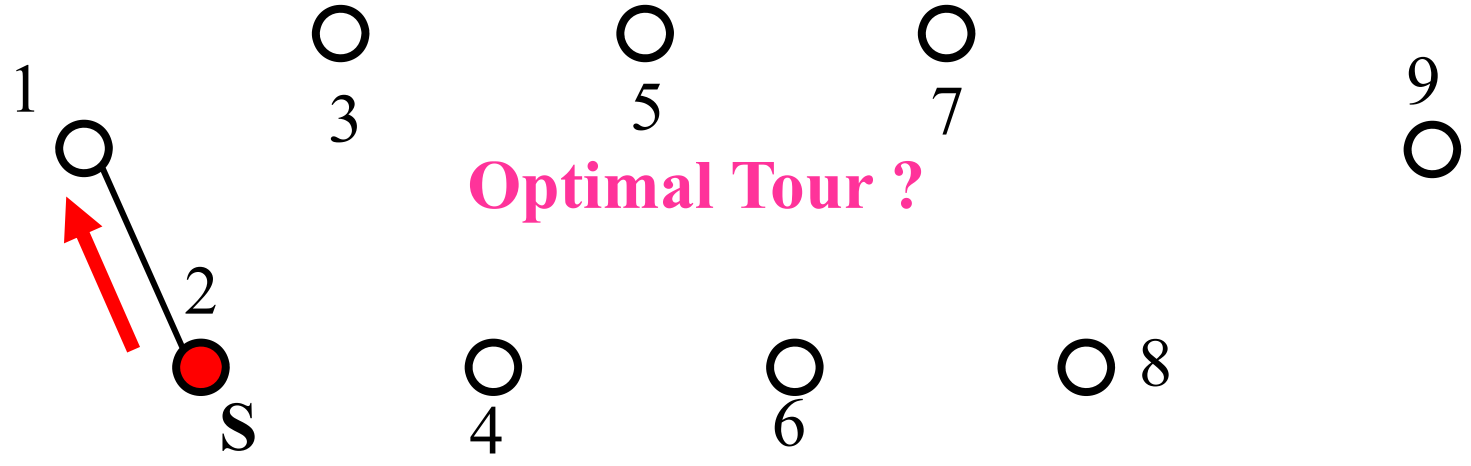
Optimal Tour ?

## Simple Examples



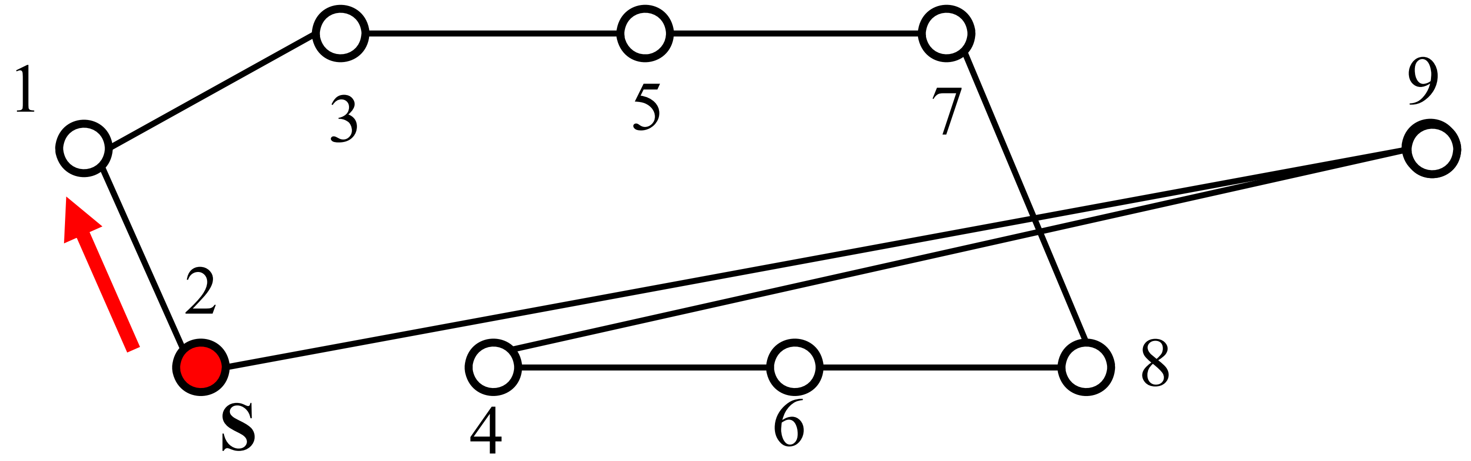
## Simple Examples

Create a greedy solution from City S (City 2).

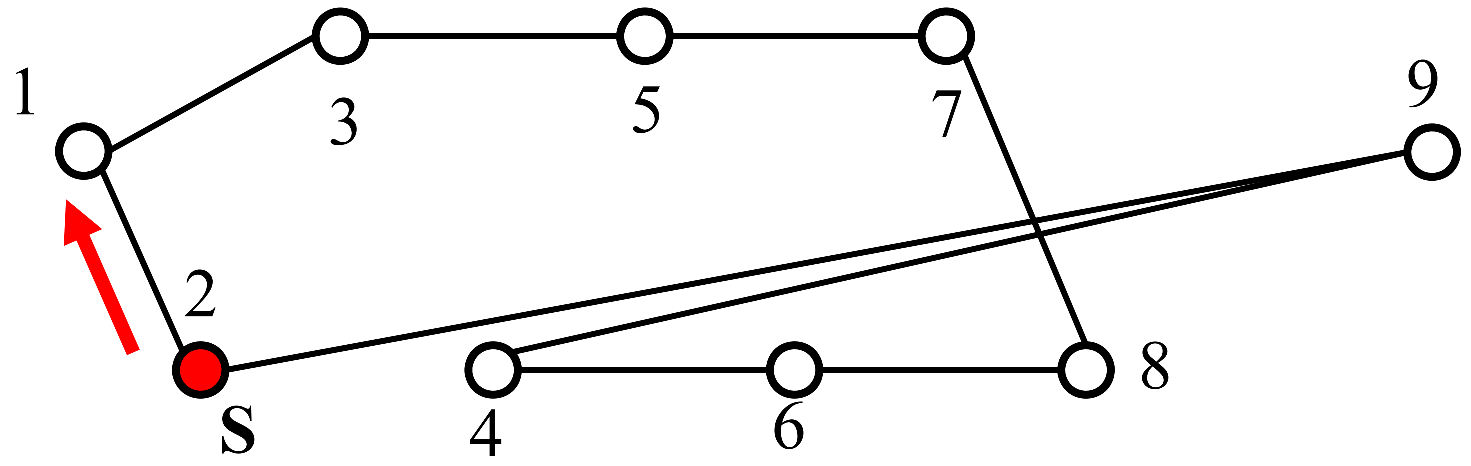


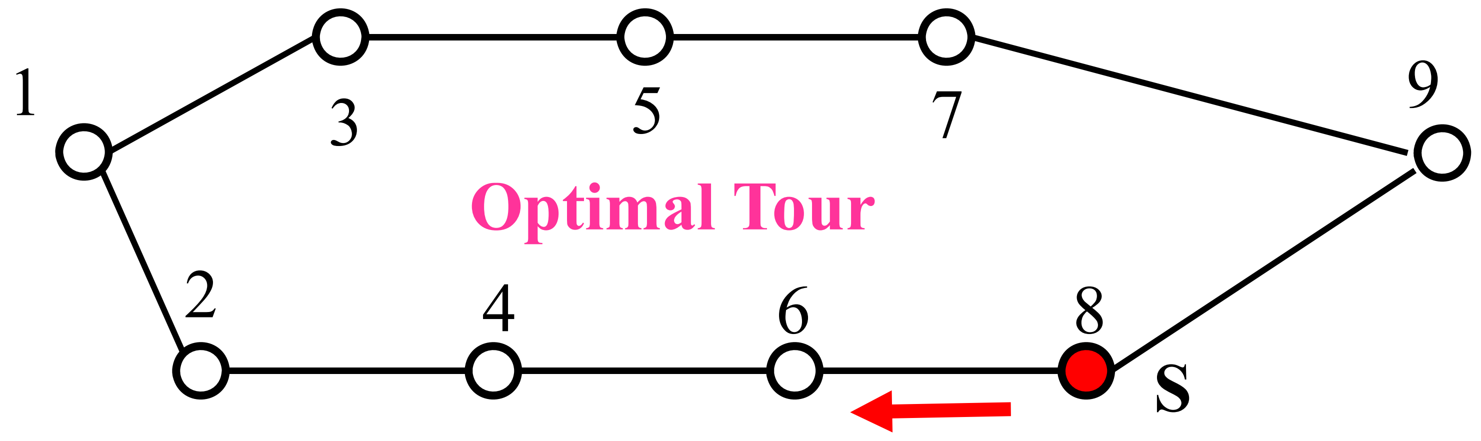
## Simple Examples

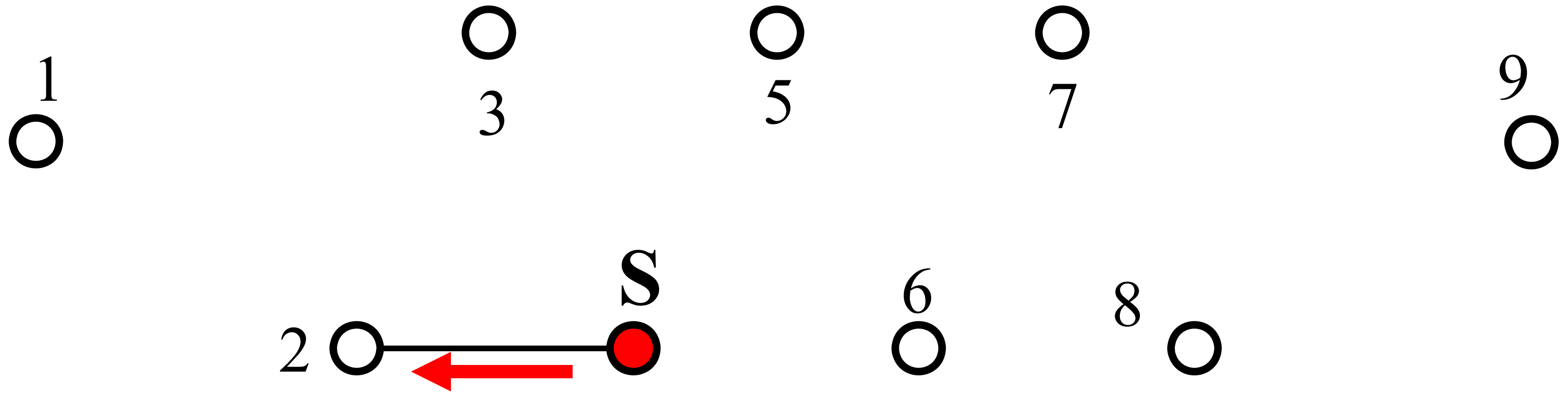
Create a greedy solution from City S (City 2).



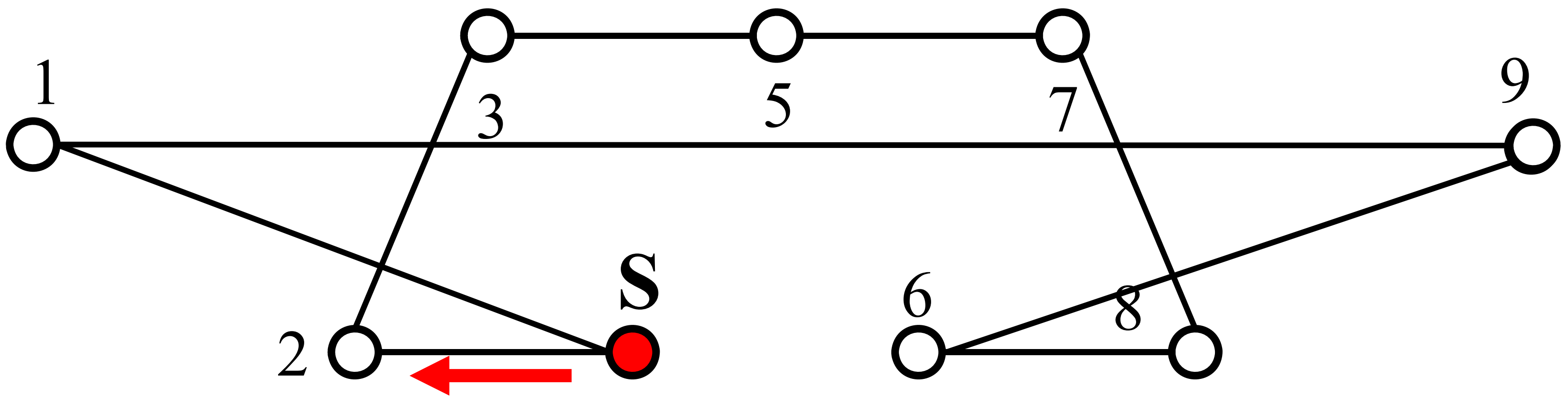
## Simple Examples





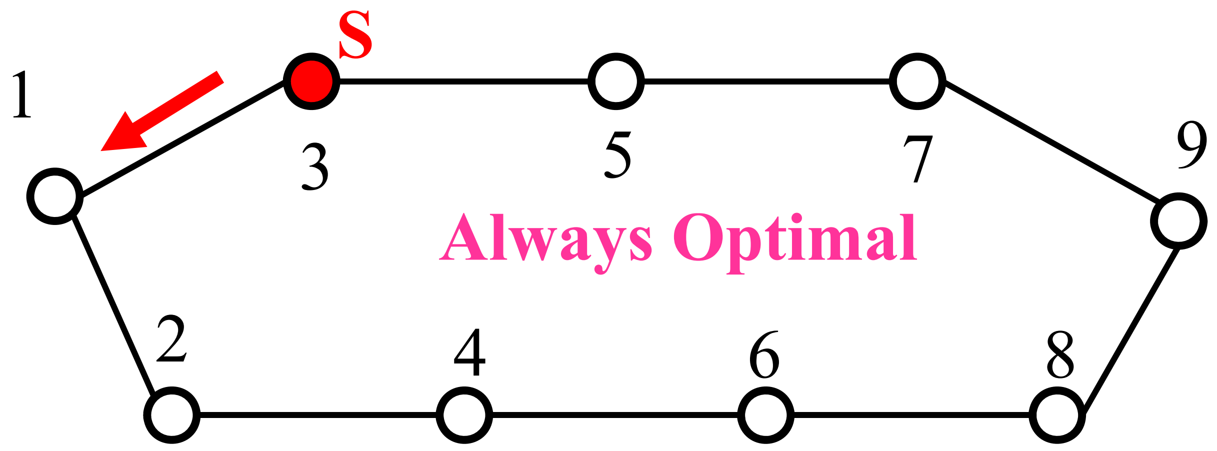


Create a greedy solution from City S.

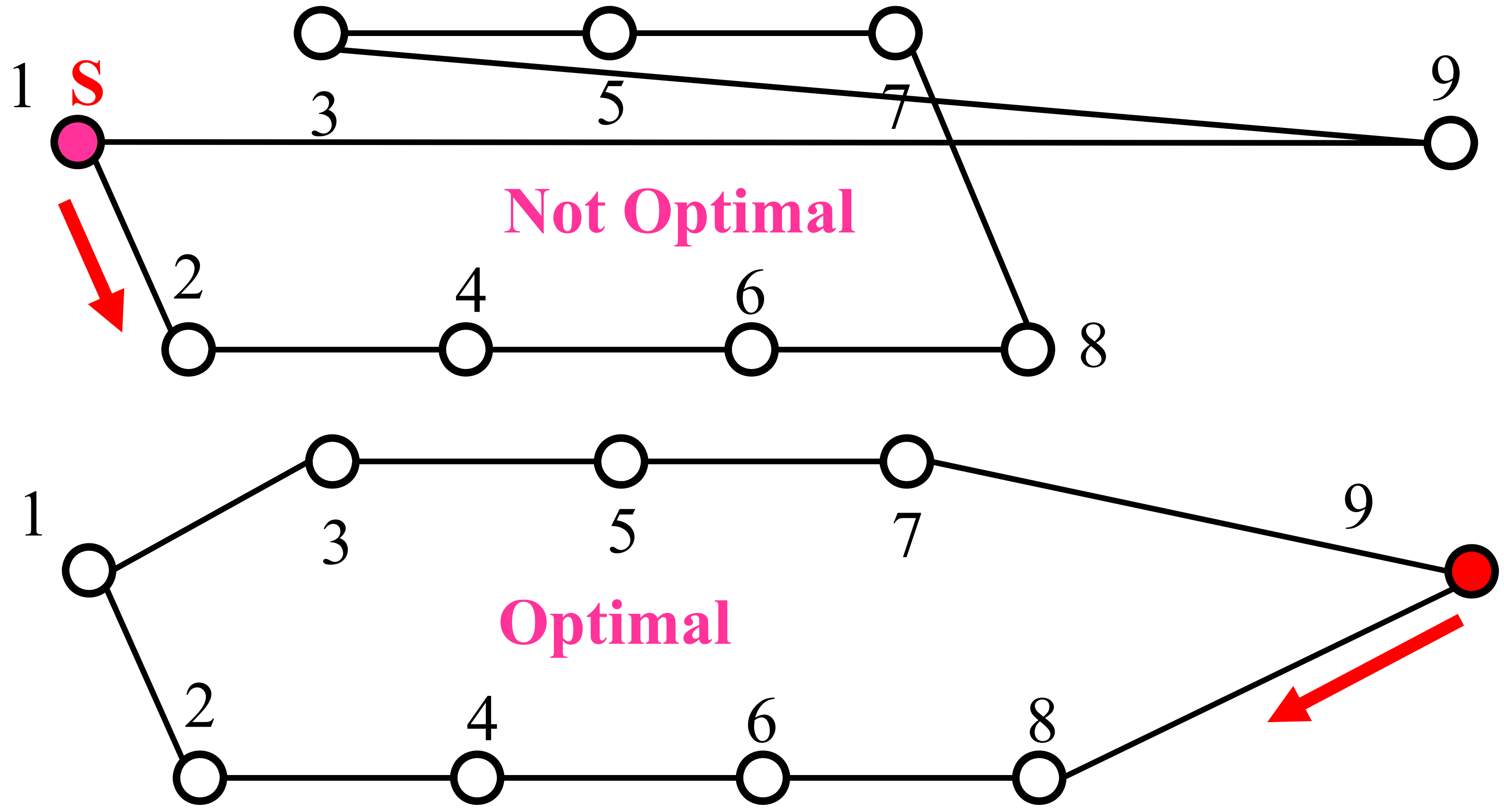


Create a greedy solution from City S.

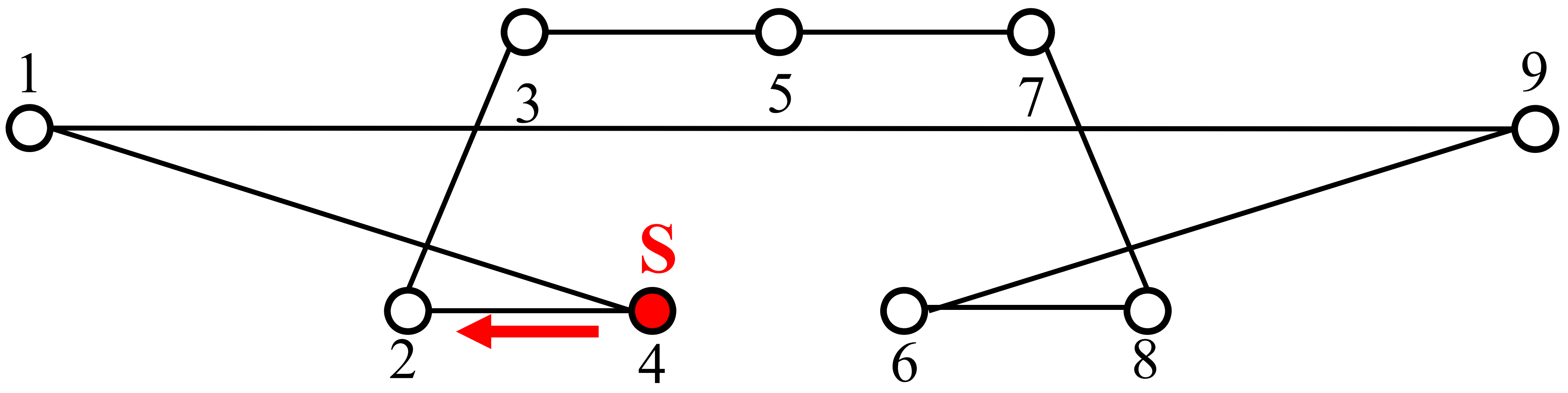
One circle:



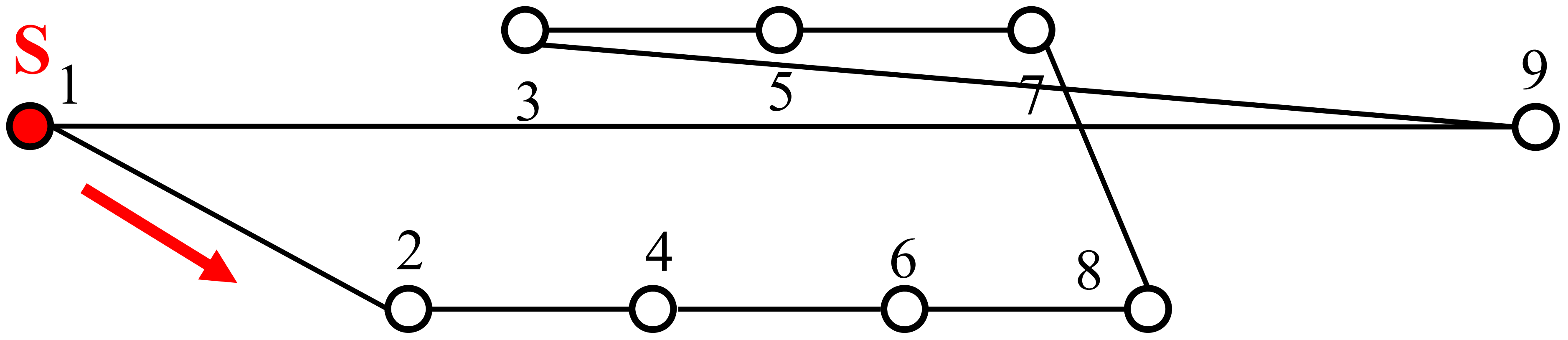
One isolated city:



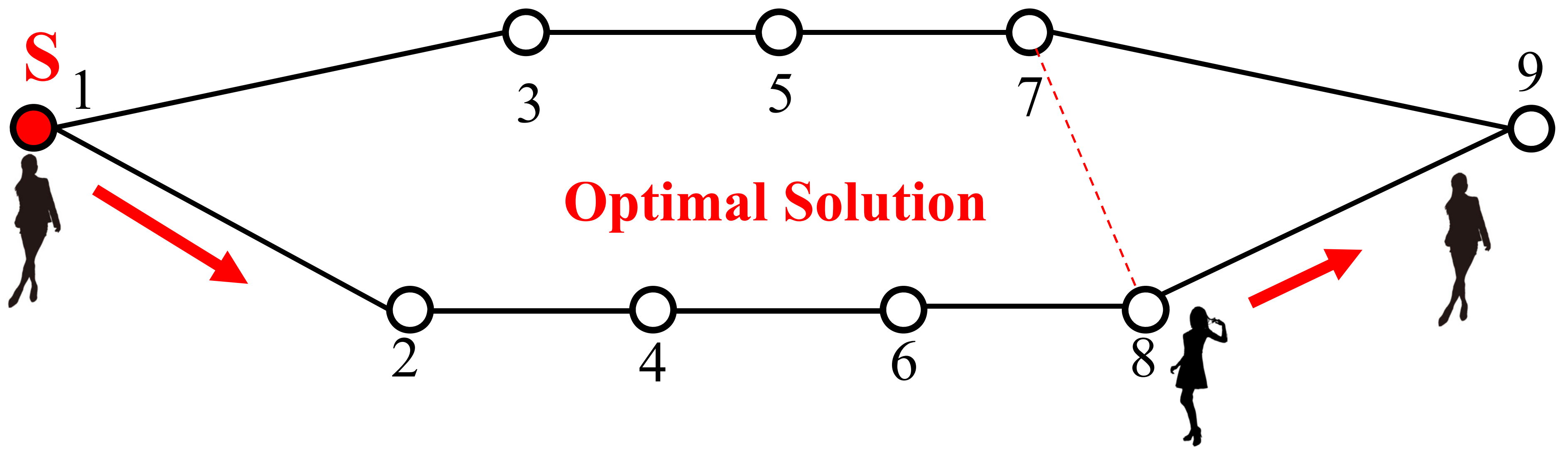
## Two isolated cities:

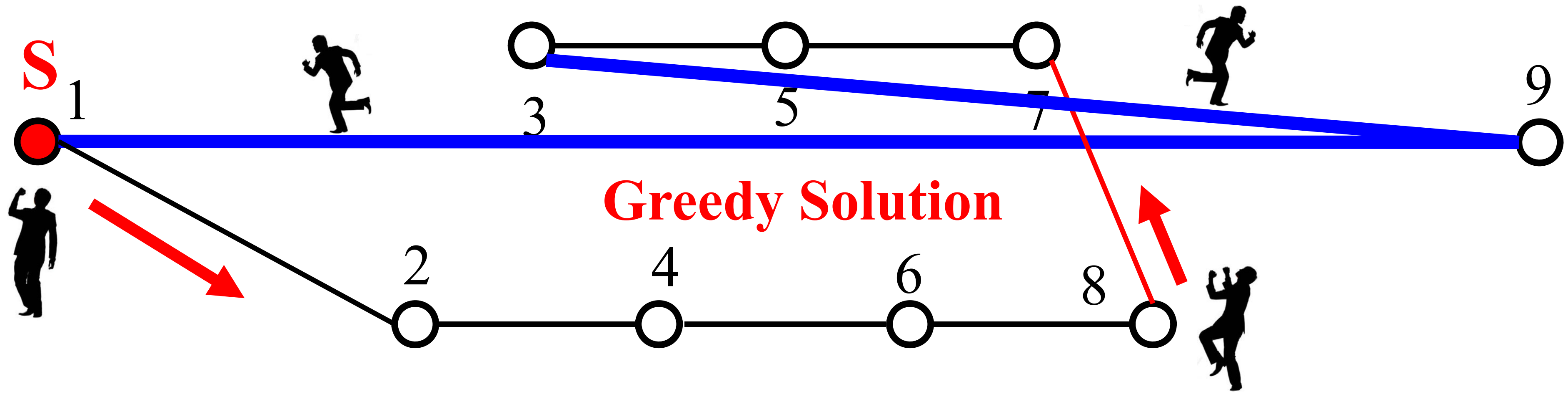


**Always Not Optimal**



## Greedy Algorithm looks poor (not intelligent):





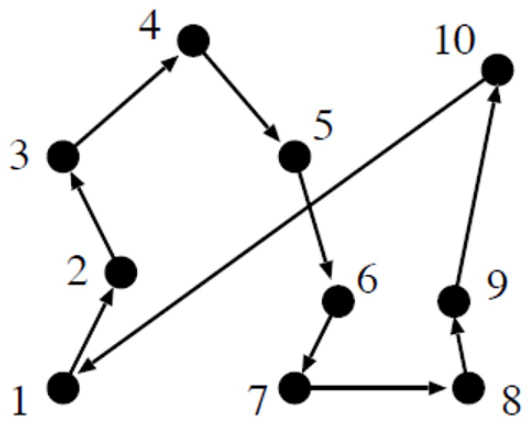
Question: How bad ?

## Worst Case Analysis for the Greedy Algorithm:

What is the worst case performance of the simple nearest neighbor greedy algorithm ?

**Q.** What is the worst possible value (i.e., the largest possible value) of the solution quality index when the tour is obtained by the simple nearest neighbor greedy algorithm?

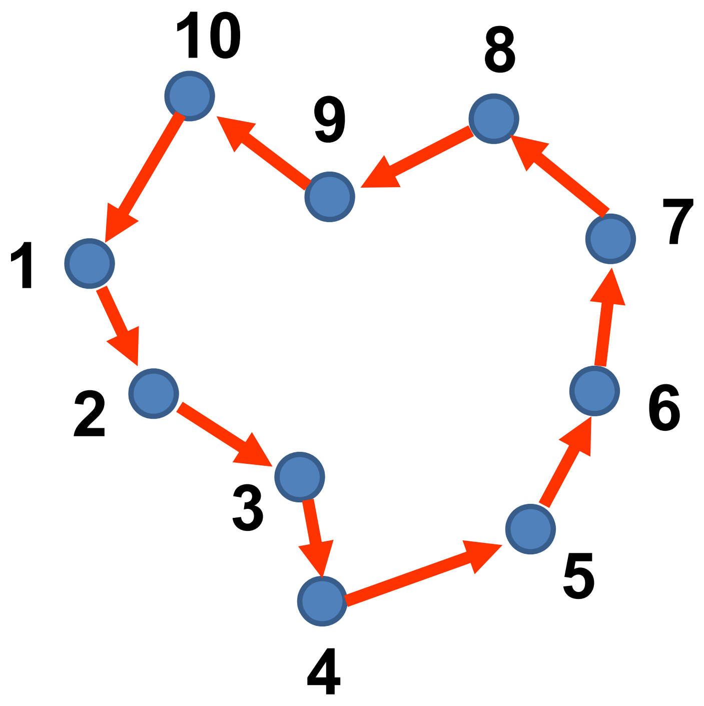
The worst case of the greedy algorithm (with the worst choice of a start city) is much better than the worst solution (*n*/2).



$$\boxed{\textbf{Solution Quality Index} = \frac{\textbf{Tour Length}}{\textbf{Optimal Tour Length}}}$$

## Simple Nearest Neighbor Greedy Algorithm

1. Arbitrarily select a start city.
2. Move from the current city to the nearest city among the remaining cities until all cities are visited.
3. Go back to the start city.



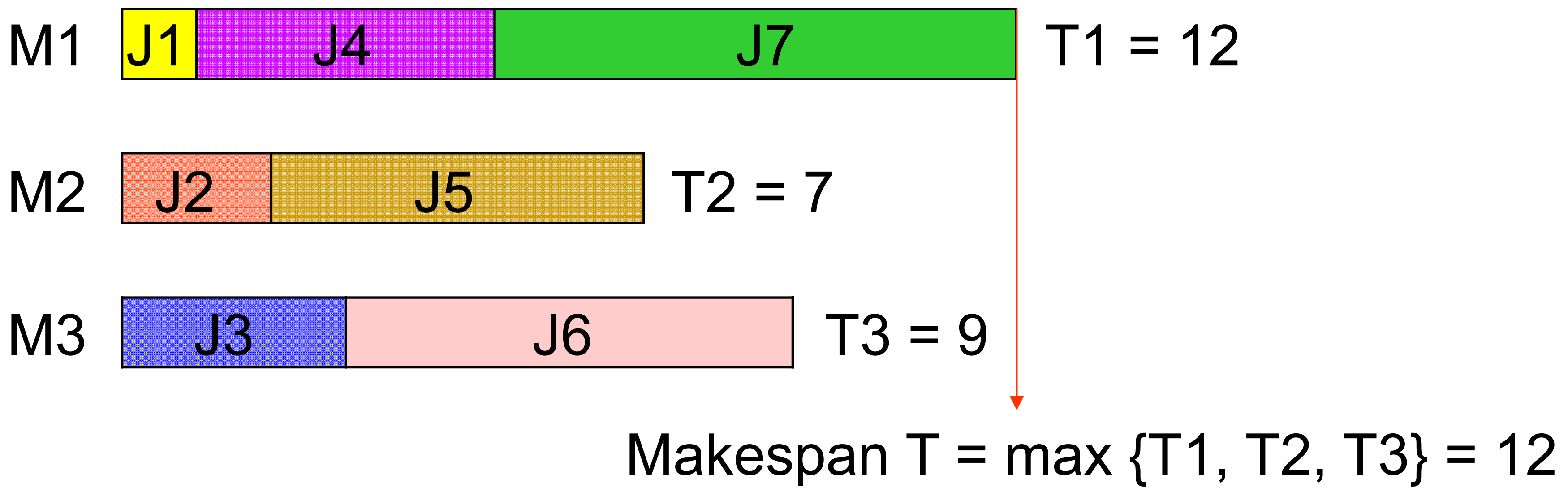
## Load Balancing Problem

**Input:** $m$ identical machines: M1, M2, ..., M$m$

$n$ jobs: J1, J2, ..., J$n$

Processing time of each job: $t_j$ ($j$ = 1, 2, ..., $n$)

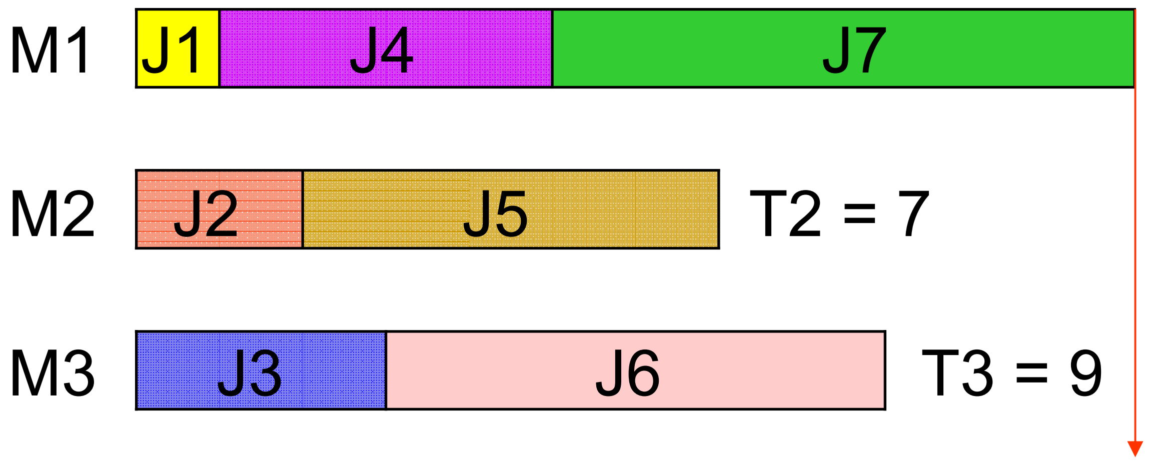
Example: 3 machines and 7 jobs ($t_j$ = 1, 2, 3, 4, 5, 6, 7)



**Q. What is the best assignment ?**

**A. The assignment with the minimum makespan.**

Example: 3 machines and 7 jobs ($t_j$ = 1, 2, 3, 4, 5, 6, 7)



Makespan T = max {T1, T2, T3} = 12

**Optimal Solution:**

M1: ??, ...

M2: ??, ...

M3: ??, ...

Optimal Makespan $T^* = \underline{??}$

## Greedy Algorithm

**Assign a job to the machine with the smallest load in an arbitrary order of jobs.**

### Simple Example: Two Machines and Three Jobs

$\boxed{\text{J1: 20}}$ $\boxed{\text{J2: 20}}$ $\boxed{\text{J3: 40}}$

If the three jobs are assigned in the order of J1, J2, J3:

Makespan = ??

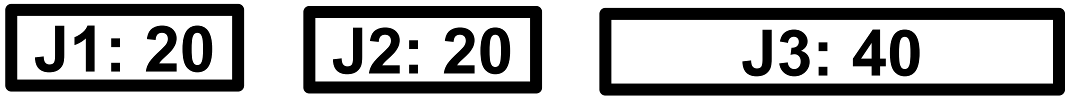
If the three jobs are assigned in the order of J3, J2, J1:

Makespan = ??

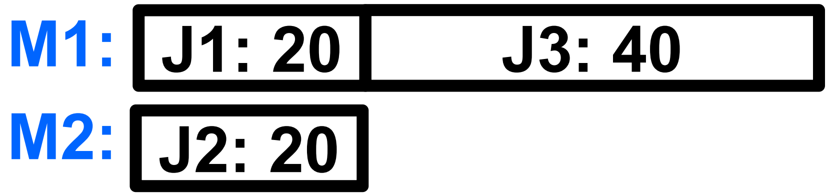
## Greedy Algorithm

**Assign a job to the machine with the smallest load in an arbitrary order of jobs.**

### Simple Example: Two Machines and Three Jobs

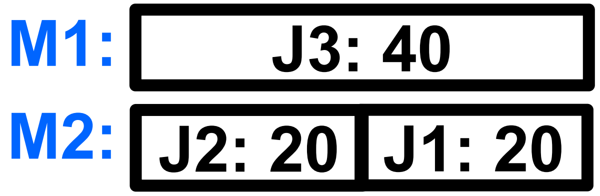


If the three jobs are assigned in the order of J1, J2, J3:



Makespan = 60

If the three jobs are assigned in the order of J3, J2, J1:



Makespan = 40

**Question: How to specify the order of jobs ?**

**Today’s Lab Session Task 1:**

Using an example of the knapsack problem, explain that the greedy solution can be much worse than the optimal solution (i.e., the total profit of a greedy solution can be smaller than 1/100 of the total profit of the optimal solution).

**Today’s Lab Session Task 2:**

Using an example of the TSP problem, explain that the greedy tour length can be about two times of the optimal tour length.

**Today’s Lab Session Task 3:**

Using an example of the load balancing problem, explain that the greedy makespan can be about two times of the optimal makespan.