Yuan Ze University

The Traveling Salesman Problem (TSP) and its solving algorithm
旅行推銷員問題與其解法

2015暑期





Global Logistics Lab.

ver. 1.0.4

Outlines

- Measuring Computational Efficiency
- Traveling Salesman Problem (TSP)
- Construction Heuristics
- Local Search Algorithms

Measuring Computational Efficiency

Consider the following algorithm

```
for(i=0; i<n; i++) {
  for(j=0;j<m;j++) {
      c[i][j] = a[i][j] + b[i][j];
  }
}
```

Total number of operations:

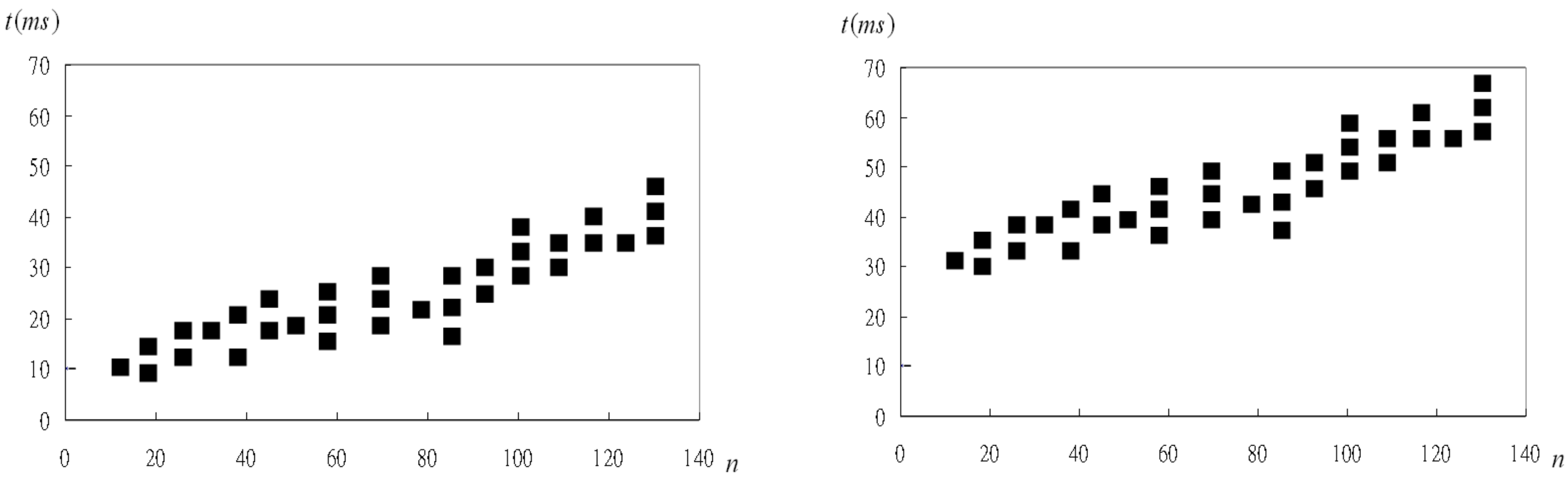
Addition: **(+)** m*n + **(++)** m*n + **(++)** n => **$(2m+1)*n*C_1$**

Assignments: **(=)** m*n + **(=)** n + **(=)** 1 => **$(m+1)*n + 1*C_2$**

Comparisons: **(<)** m*n + **(<)** n => **$(m+1)*n*C_3$**

Measuring Computational Efficiency

Which one is faster?

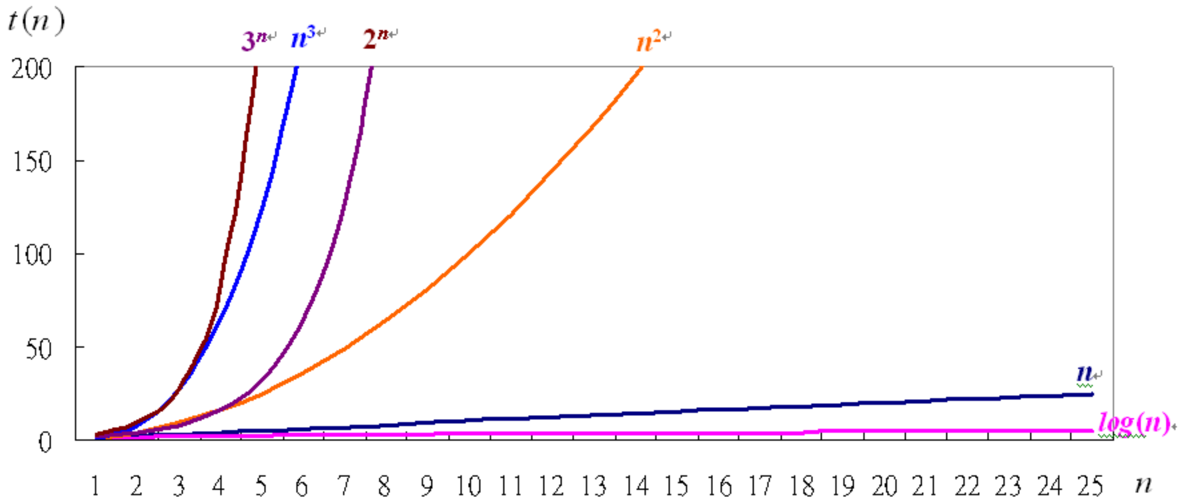


(a) (b)

Measuring Computational Efficiency

- Running Time

$$\underbrace{\log(n) < n < n^2 < n^3}_{\text{polynomial time}} \le \underbrace{2^n < 3^n < n!}_{\text{exponential}}$$



Measuring Computational Efficiency

- Big-O notation

  $f(n)$ is O(g(n)) **:** if there is a real number $c > 0$ and an integer constant $n_0 \geq 1$, such that $f(n) \leq cg(n)$ for every integer $n \geq n_0$.

- Examples

  $7n$-2 is O(n)

  $20n^3+10n\log n+5$ is O(n^3)

  2^{100} is O(1)

Measuring Computational Efficiency

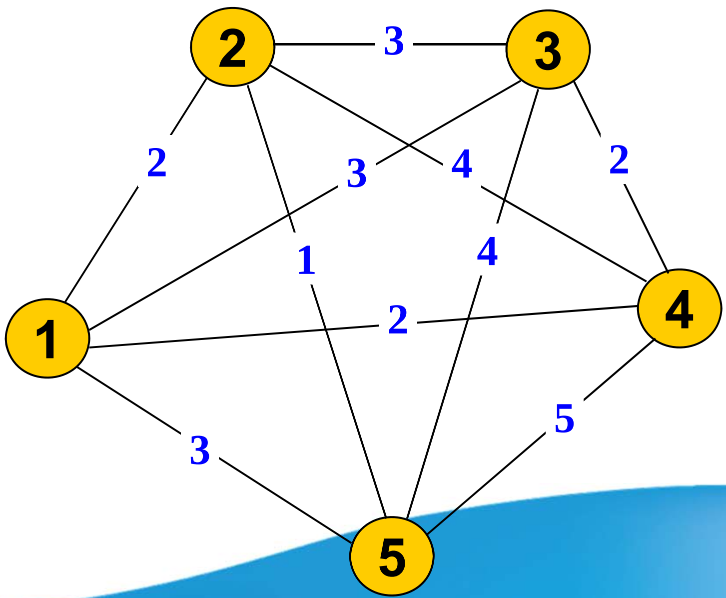
- Big-O notation

O(log(n)) < O(n) < O(n log(n)) <O(n^2) < O(n^3) <O(2^n) <O(3^n)

logarithmic	*linear*	*polynomial*	*exponential*
$O(\log n)$	$O(n)$	$O(n^k),\ k \geq 1$	$O(a^n),\ a \geq 1$

Traveling Salesman Problem (TSP)

- The TSP can be described as the problem of finding the minimum distance route that begins at a given node of the network, visits all the members of a specified set of nodes exactly once, and returns eventually to the initial node.



Standard Formulation

- Dantzig, Fulkerson, Johnson (1954) :

Suppose there exists n cities, x_{ij} is a link in tour, $i, j \in \{1, 2, \ldots, n\}$.

Minimise: $$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

subject to:

$$\left.\begin{array}{ll} \sum_{i=1}^{n} x_{ij} = 1 & \forall\, j \\ \sum_{j=1}^{n} x_{ij} = 1 & \forall\, i \end{array}\right\} \text{assignment}$$

$$\left.\sum_{i,j\in S} x_{ij} \le |S| - 1 \quad \forall\, S \subseteq \{2, 3, \ldots, n\}\right\} \text{subtour elimination}$$

the general form

- Interpreting the general form of constrains
  - General form

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j$$

  - Expression form (n = 4)
    - Expansion Summation:

$$x_{1j} + x_{2j} + x_{3j} + x_{4j} = 1 \qquad \forall j$$

    - Expansion Constraints:

$$x_{11} + x_{21} + x_{31} + x_{41} = 1$$
$$x_{12} + x_{22} + x_{32} + x_{42} = 1$$
$$x_{13} + x_{23} + x_{33} + x_{43} = 1$$
$$x_{14} + x_{24} + x_{34} + x_{44} = 1$$

Subtour

- Assignment constraints:

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall\, j$$

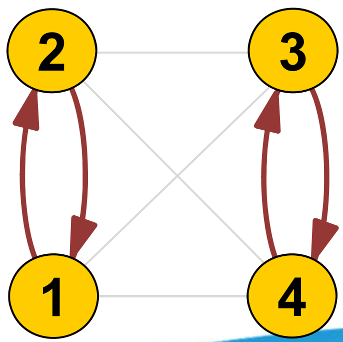
Salesman travels to node *j* from exactly one node *i*.

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i$$

Salesman travels from node *i* to exactly one node *j*

- Summation of each column (or row) is equal to 1.
- However, the subtour may occur:

$$\begin{array}{lcl} & & \text{sum} \\ x_{ij} = & \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} & \begin{matrix} 1 \\ 1 \\ 1 \\ 1 \end{matrix} \\ \text{sum} & \begin{matrix} 1 & 1 & 1 & 1 \end{matrix} & \end{array}$$

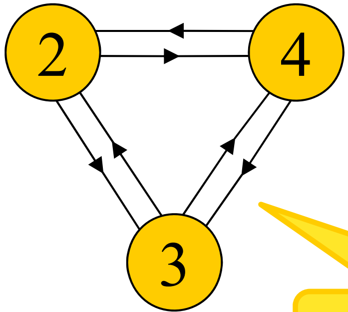


Subtour Elimination

$$\sum_{i,j \in S} x_{ij} \le |S| - 1 \qquad \forall\, S \subseteq \{2, 3, \ldots, n\}$$

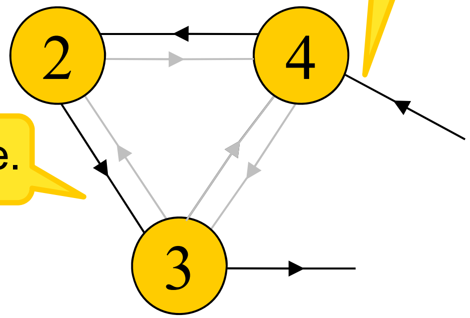
The subtour elimination forces the subset of nodes to connect to other nodes.

Example: $x_{32} + x_{24} + x_{43} + x_{23} + x_{42} + x_{43} \le 2$



Only two arcs can be used.

This is infeasible.



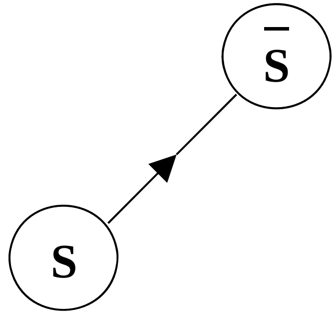
O(2^n) *Constraints* $= (2^{n-1} + n - 2)$

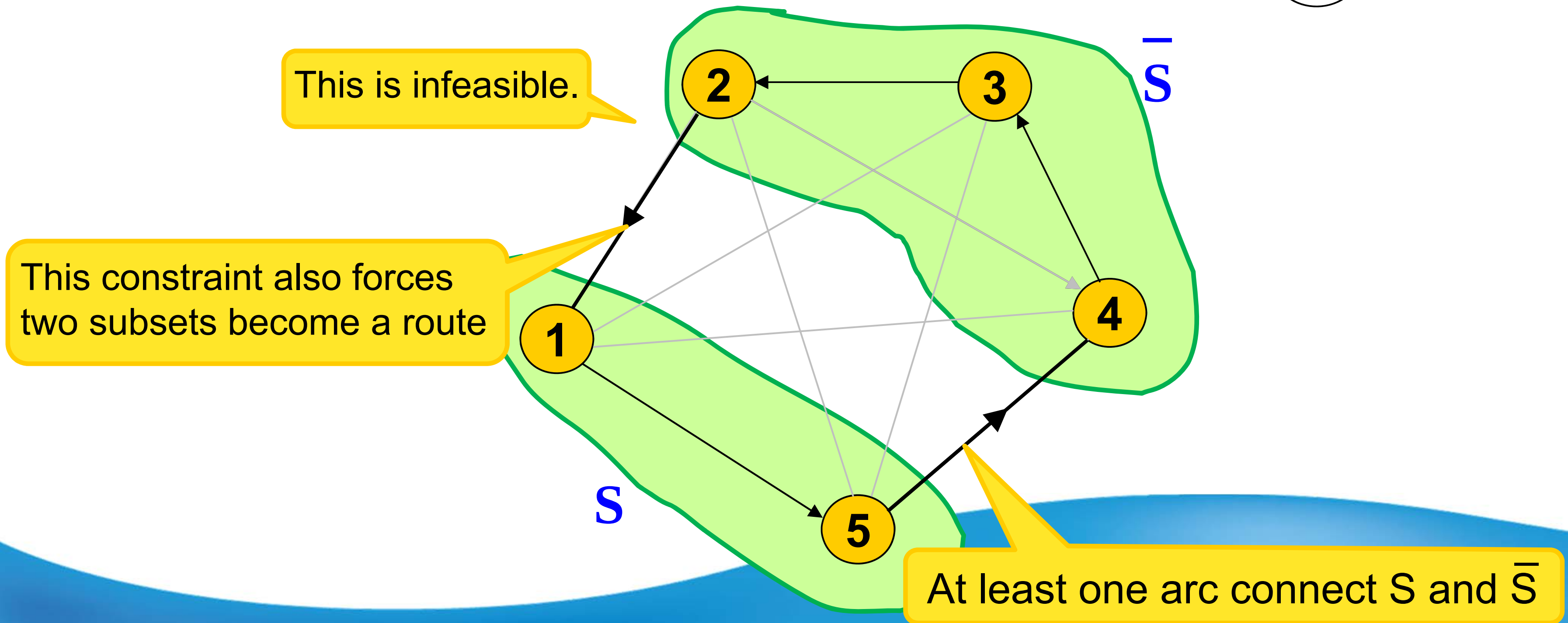
O(n^2) *Variables* $= n(n-1)$

Subtour Elimination (Equivalent Formulation)

- Replace subtour elimination constraints with

$$\sum_{i \in S,\, j \in \overline{S}} x_{ij} \geq 1 \quad \forall\ S \subseteq \{2, 3, \ldots, n\}$$





MTZ Formulation

- Miller, Tucker, Zemlin (1960):

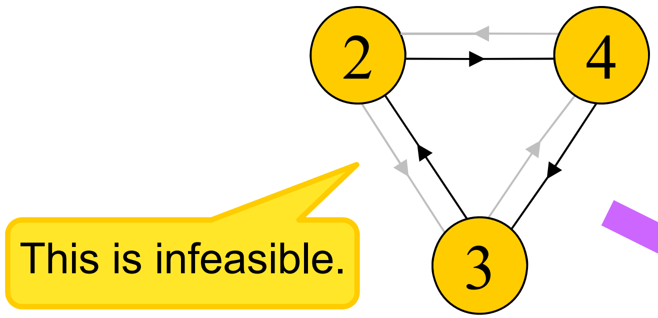
u_i = Sequence Number in which city i visited for $i = \{2, 3, \ldots, n\}$

Subtour elimination constraints replaced by

$$u_i - u_j + n x_{ij} \leq n - 1 \quad \forall i, j = \{2, 3, \ldots, n\}$$

MTZ Formulation

- **Avoids subtours but allows total tours (containing city 1)**



$$u_2 - u_4 + nx_{24} \le n\text{-}1$$

$$u_4 - u_3 + nx_{43} \le n\text{-}1$$

$$u_3 - u_2 + nx_{32} \le n\text{-}1$$

$$3n \le 3(n - 1)$$

0(n2) Constraints = (n2 – n + 2)

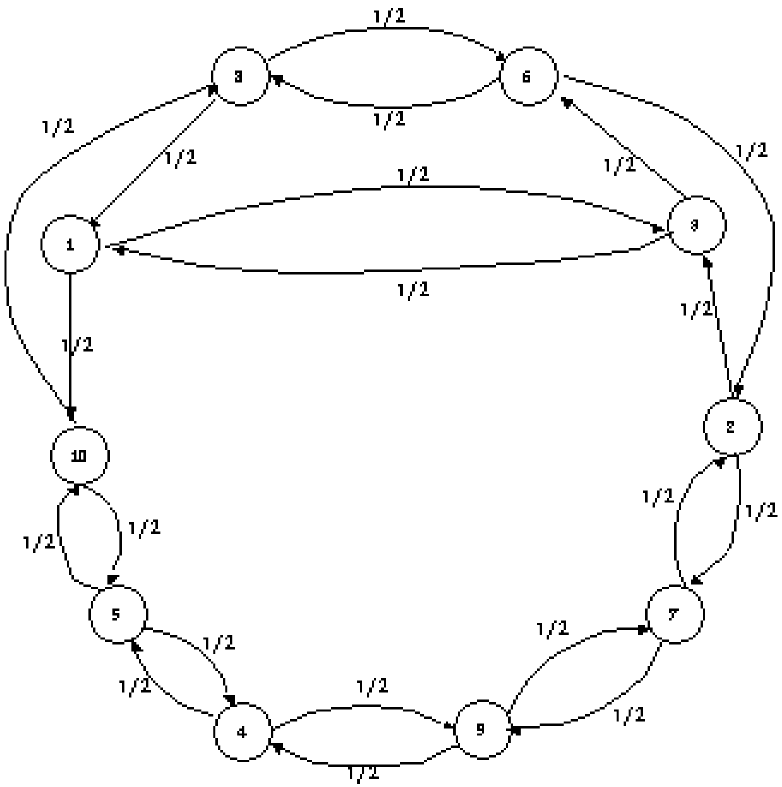
0(n2) Variables = (n – 1) (n + 1)

Weak, but can add "Logic Cuts"

e.g. $u_k \ge 1 + x_{ij} + x_{jk} + x_{1j}$

Standard Formulation

Lower Bound (LP Relaxation)

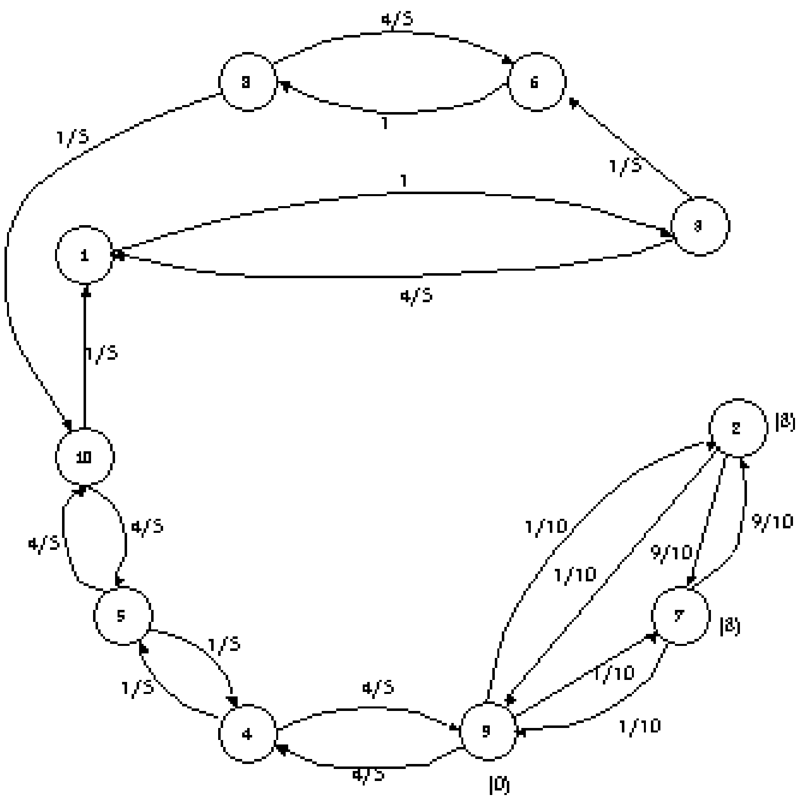


LP Relaxation Cost = 878

(Optimal Cost = 881)

MTZ Formulation

Lower Bound (LP Relaxation)



Subtour Constraints Violated : e.g.

$$\boldsymbol{x}_{27} + \boldsymbol{x}_{72} \not\leq 1$$

Logic Cuts Violated: e.g.

$$u_{9} \not\geq 1 + x_{27} + x_{79} - x_{17}$$

LP Relaxation Cost = 773 $^3/_5$

(Optimal Cost = 881)

Construction Heuristics

- Greedy Algorithms:
  - Using an index to fix the priority for solving the problem
  - Less flexibility to reach optimal solution
  - Constructing an initial solution for improvement algorithms
- Example:
  - Northwest corner and minimum cost matrix for transportation problem

Construction Heuristics

- Nearest neighbor procedure – $O(n^2)$
- Nearest insertion – $O(n^2)$
- Furthest insertion – $O(n^2)$
- Cheapest insertion – $O(n^3)$

  or – $O(n^2 \log n)$ (using heap)

Heuristic - Nearest Neighbor (NN)

Nearest neighbor for TSP

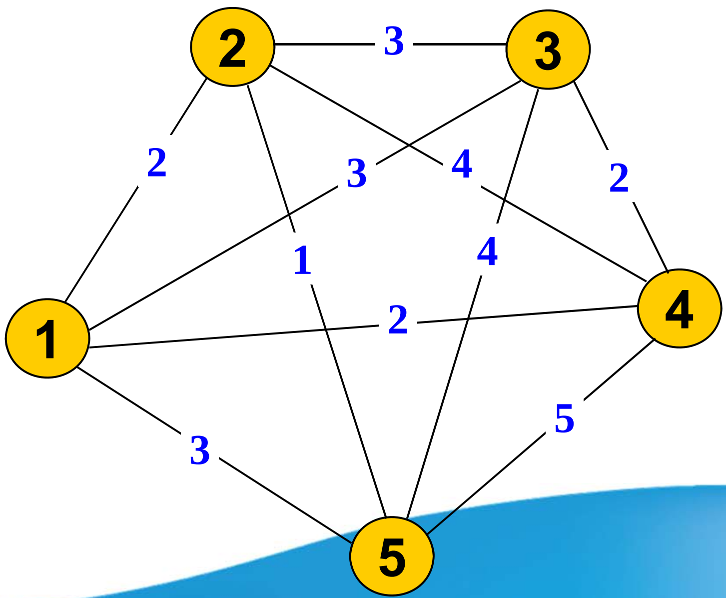
1. Start with an arbitrary node i as the beginning of a path.
2. Find a unvisited node k closest (minimum c_{jk}) to the last node at current path. Add node k to the path.
3. Label node k as visited node.
4. Repeat Step 2 and 3 until all nodes are contained in the path. Then join the first and last nodes

Heuristic - Nearest Neighbor (NN)

Step 1: Suppose node 1 is chose as beginning.

Step 2: The node 4 is selected such that the path has minimal increase cost c_{14}.

Step 3:



Heuristic - Nearest Neighbor (NN)

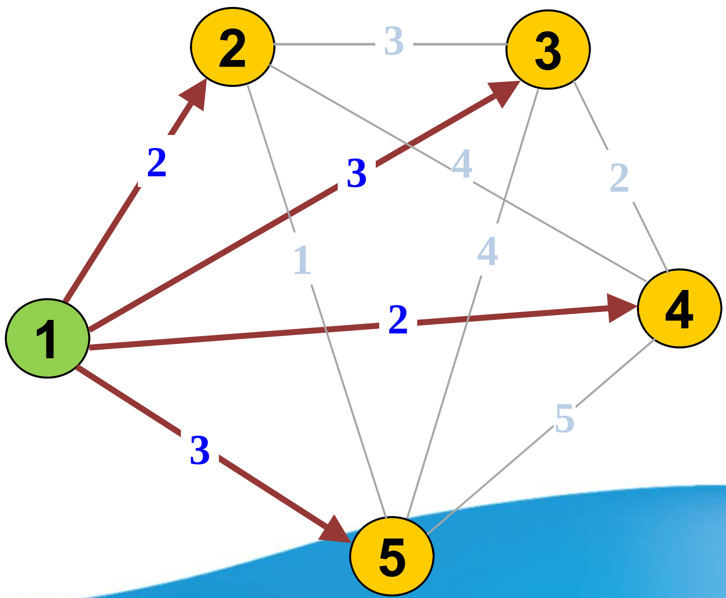
Step 1: Suppose node 1 is chose as beginning.

Step 2: The node 4 is selected such that the path has minimal increase cost c_{14}.

Step 3:

arbitrary choose one

1-2: 2 ⬅
1-3: 3
1-4: 2 ⬅
1-5: 3



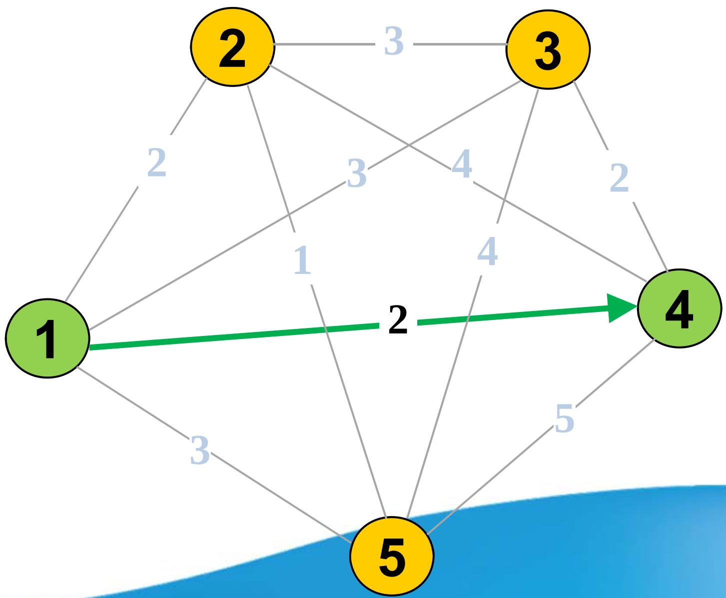
Heuristic - Nearest Neighbor (NN)

Step 1: Suppose node 1 is chose as beginning.

Step 2: The node 4 is selected such that the path has minimal increase cost c_{14}.

Step 3: Node 4 is selected and labeled as visited node.

1-2: 2
1-3: 3
1-4: 2
1-5: 3

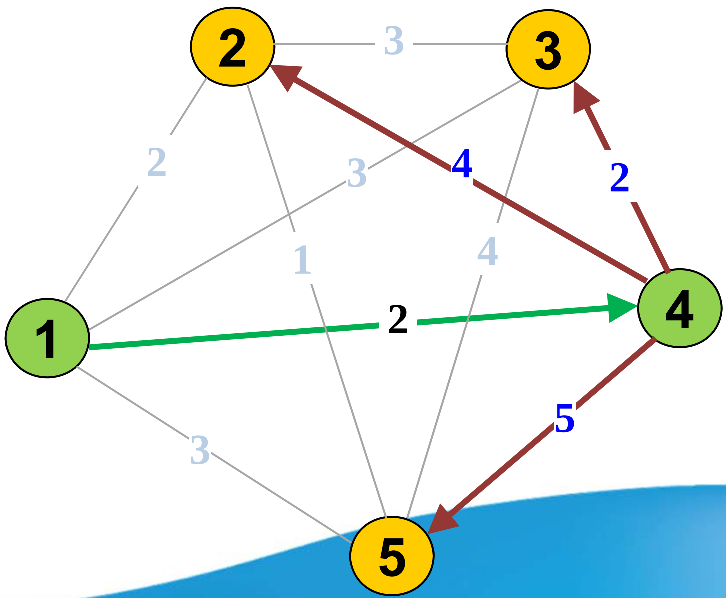


Heuristic - Nearest Neighbor (NN)

Step 2: The node 3 is selected such that the path has minimal increase cost c_{43}.

Step 3:

4-2: 4
4-3: 2
4-5: 5

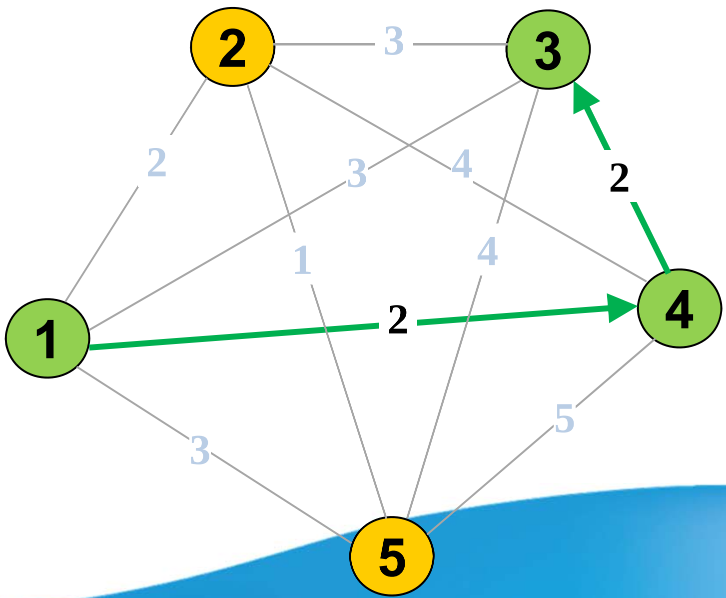


Heuristic - Nearest Neighbor (NN)

Step 2: The node 3 is selected such that the path has minimal increase cost c_{43}.

Step 3: Node 3 is selected and labeled as visited node.

4-2: 4
4-3: 2
4-5: 5

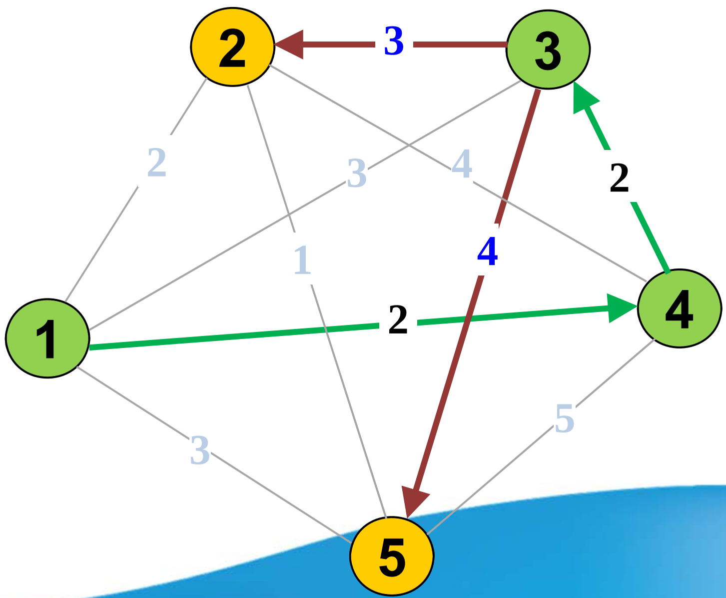


Heuristic - Nearest Neighbor (NN)

Step 2: The node 2 is selected such that that the path has minimal increase cost c_{32}.

Step 3:

3-2: 3
3-5: 4



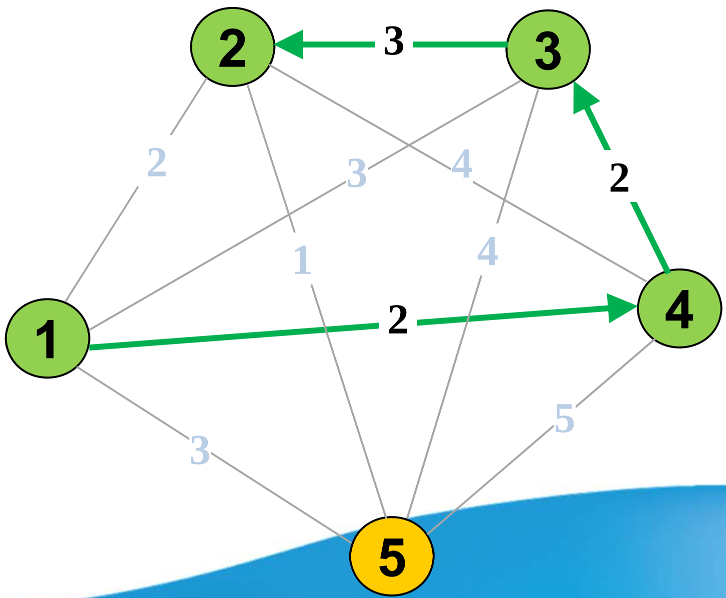
Heuristic - Nearest Neighbor (NN)

Step 2: The node 2 is selected such that that the path has minimal increase cost c_{32}.

Step 3: Node 2 is selected and labeled as visited node.

3-2: 3

3-5: 4



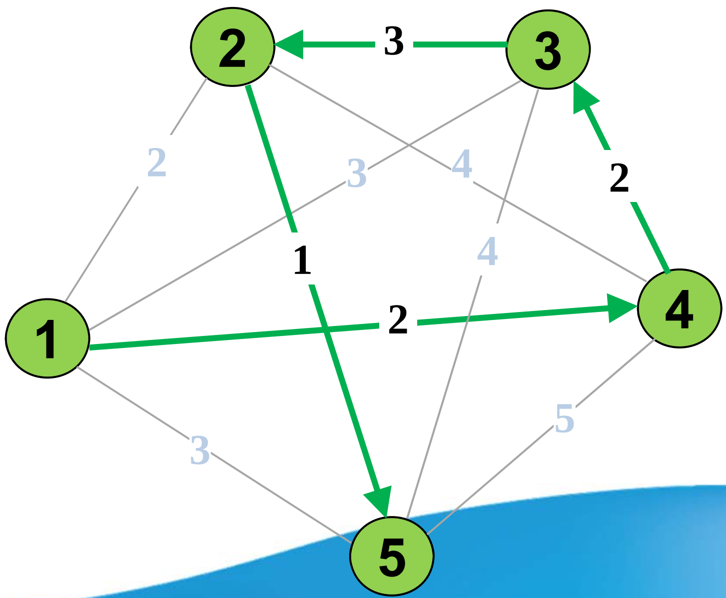
Heuristic - Nearest Neighbor (NN)

Step 2: Add the only unvisited node 5 to the path.

Step 3: Node 5 is selected and labeled as visited node.

Step 4:

2-5: 1



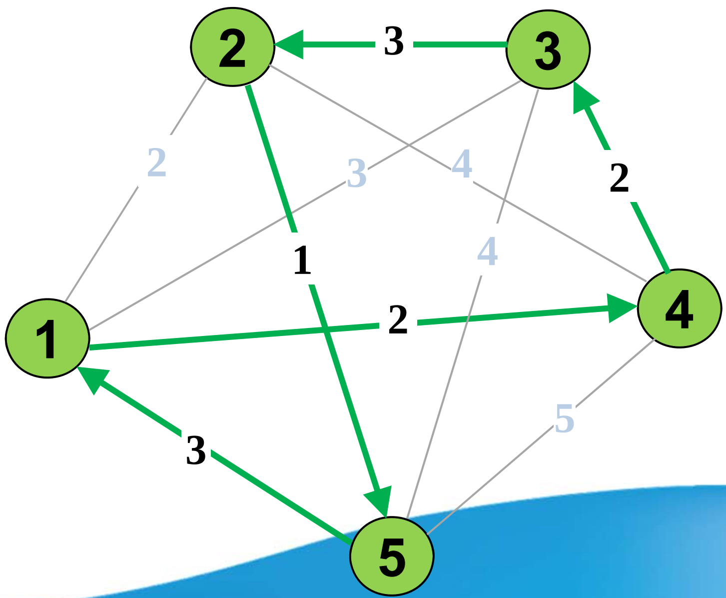
Heuristic - Nearest Neighbor (NN)

Step 2: Add the only unvisited node 5 to the path.

Step 3: Node 5 is selected and labeled as visited node.

Step 4: Link node 5 and node 1 to form a TSP tour.

2-5: 1



Heuristic - Nearest insertion (NI)

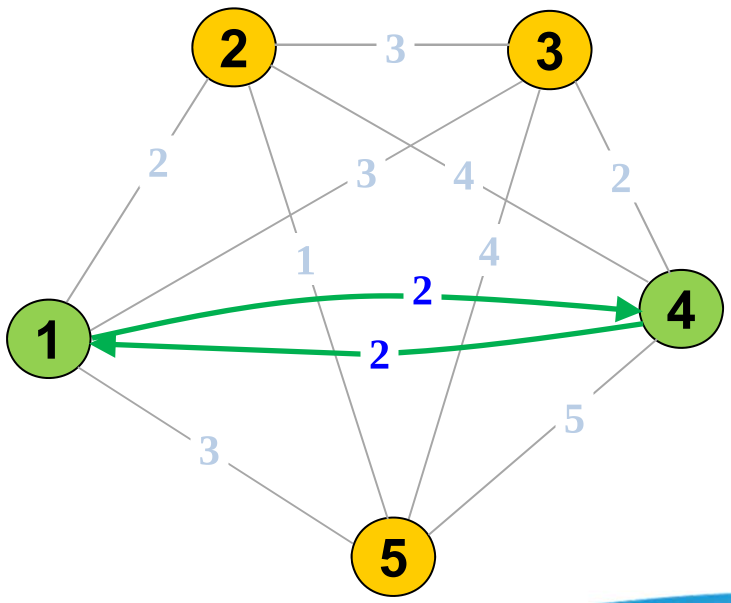
Nearest insertion for TSP

1. Start with a subgraph consisting of node i only.
2. Find node k such that c_{ik} is minimal and form the subtour i-k-i
3. (Selection) Given a subtour, find node k not in the subtour closest to any node in the tour.
4. (Insertion) Find the $arc(i, j)$ in the subtour which minimizes $c_{ik}+c_{kj}-c_{ij}$ Insert k between i and j.
5. Go to step3 unless we have a Hamiltonian cycle.

Heuristic - Nearest insertion (NI)

Step 1: Suppose node 1 is chose as beginning.

Step 2: The node 4 is selected such that subtour with minimal cost $2c_{14}$



Heuristic - Nearest insertion (NI)

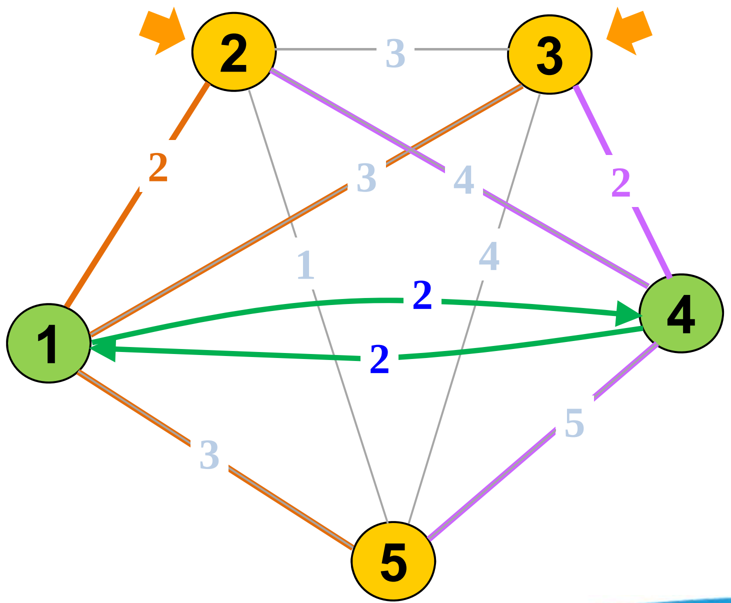
Step 3: Node 3 and 2 are closest to node 1 and 4 respectively. Node 3 is selected arbitrarily.

(Selection)

arbitrary choose one

1-2: 2
1-3: 3
1-5: 3

4-2: 4
4-3: 2
4-5: 5



Heuristic - Nearest insertion (NI)

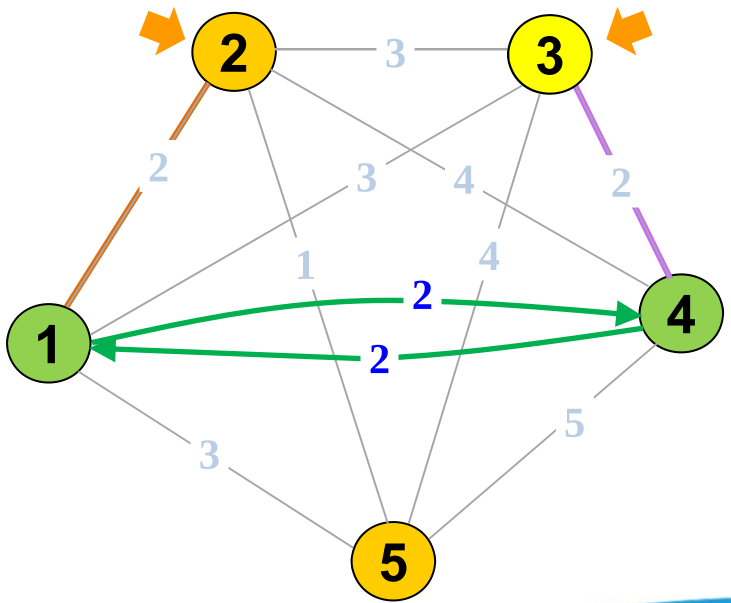
Step 3: Node 3 and 2 are closest to node 1 and 4 respectively. Node 3 is selected arbitrarily.

(Selection)

arbitrary choose one

1-2: 2
1-3: 3
1-5: 3

4-2: 4
4-3: 2
4-5: 5



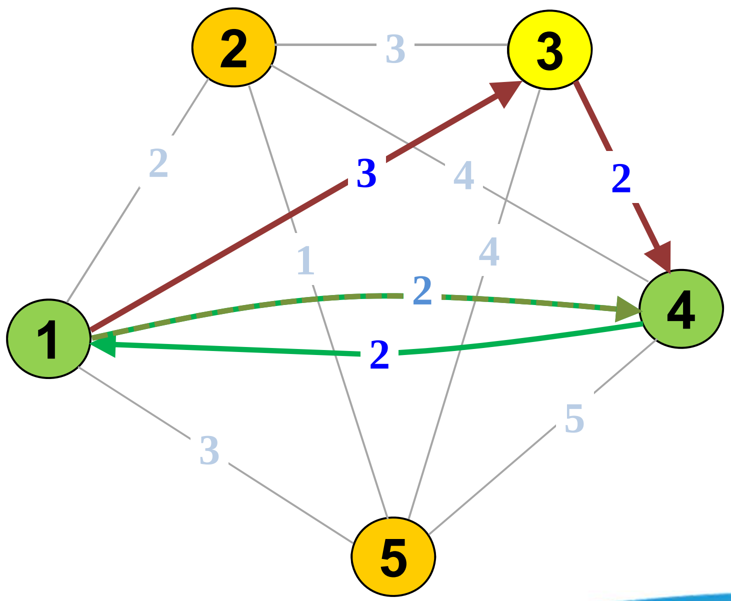
Heuristic - Nearest insertion (NI)

Step 3: Node 3 and 2 are closest to node 1 and 4 respectively. Node 3 is selected arbitrarily.

(Selection)

1-2: 2
1-3: 3
1-5: 3

4-2: 4
4-3: 2
4-5: 5



(Insertion)

1-3-4: 3+2-2=3

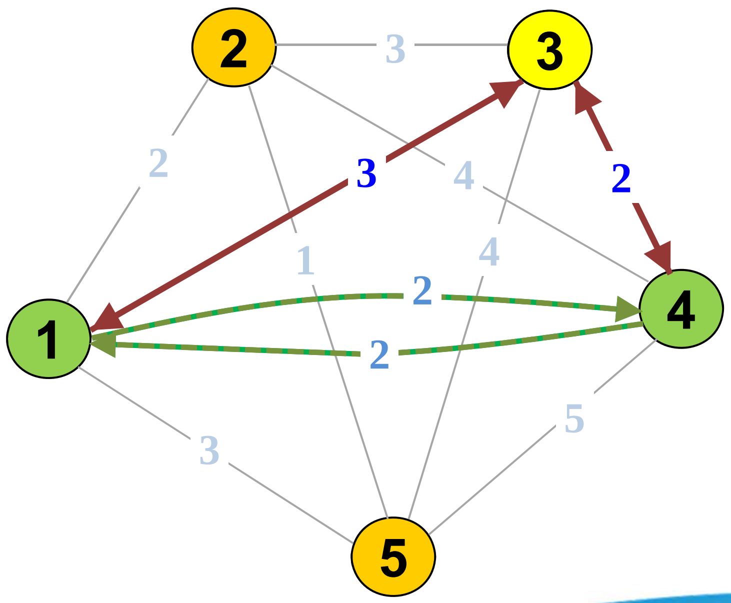
Heuristic - Nearest insertion (NI)

Step 3: Node 3 and 2 are closest to node 1 and 4 respectively. Node 3 is selected arbitrarily.

(Selection)

1-2: 2
1-3: 3
1-5: 3

4-2: 4
4-3: 2
4-5: 5



(Insertion)

1-3-4: 3+2-2=3
4-3-1: 3+2-2=3

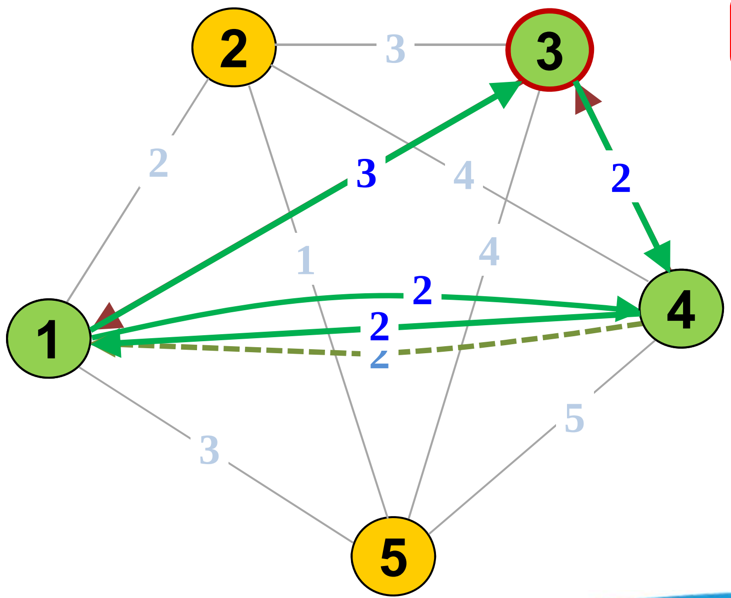
Heuristic - Nearest insertion (NI)

Step 3: Node 3 and 2 are closest to node 1 and 4 respectively. Node 3 is selected arbitrarily.

(Selection)

1-2: 2
1-3: 3
1-5: 3

4-2: 4
4-3: 2
4-5: 5



(Insertion)

arbitrary choose one

$\boxed{\text{1-3-4: 3+2-2=3}}$

4-3-1: 3+2-2=3

Heuristic - Nearest insertion (NI)

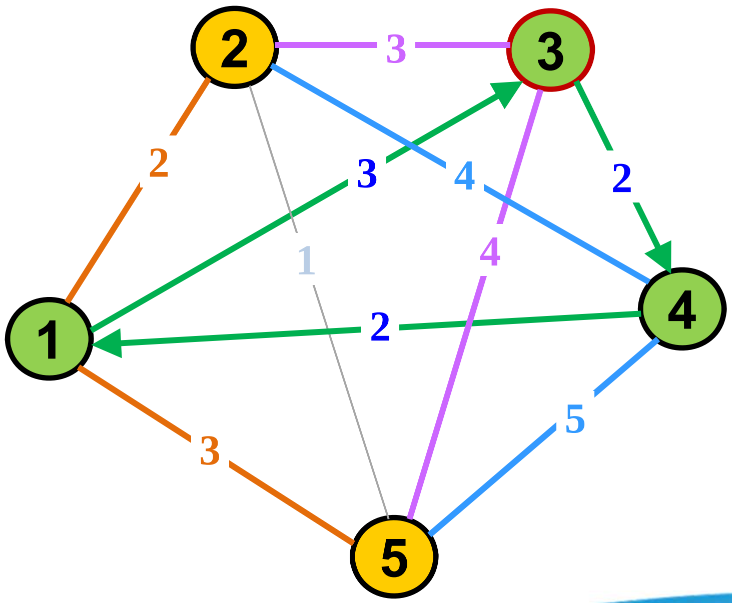
Step 3: Node 2 is closest to node 1 in the subtour.

(Selection)

1-2: 2
1-5: 3

3-2: 3
3-5: 4

4-2: 4
4-5: 5



Heuristic - Nearest insertion (NI)

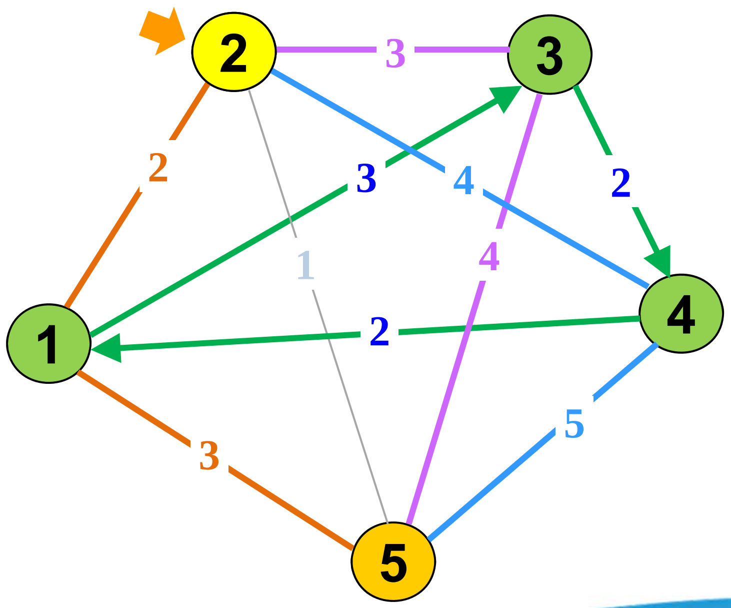
Step 3: Node 2 is closest to node 1 in the subtour. Node 2 is selected.

(Selection)

1-2: 2
1-5: 3

3-2: 3
3-5: 4

4-2: 4
4-5: 5



Heuristic - Nearest insertion (NI)

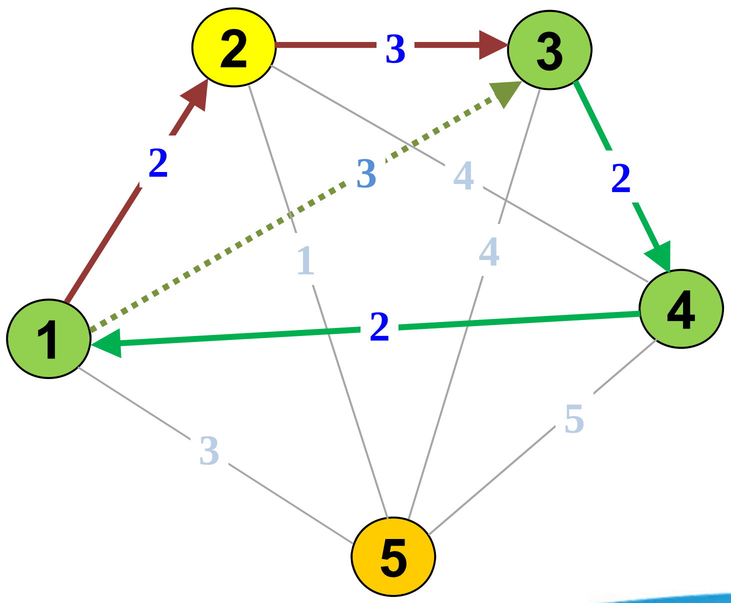
Step 4: The selected node 2 is inserted between node 1 and 3 in the subtour with the minimal increasing cost = 2.

(Selection)

1-2: 2
1-5: 3

3-2: 3
3-5: 4

4-2: 4
4-5: 5



(Insertion)

1-2-3: 3+2-3=2

Heuristic - Nearest insertion (NI)

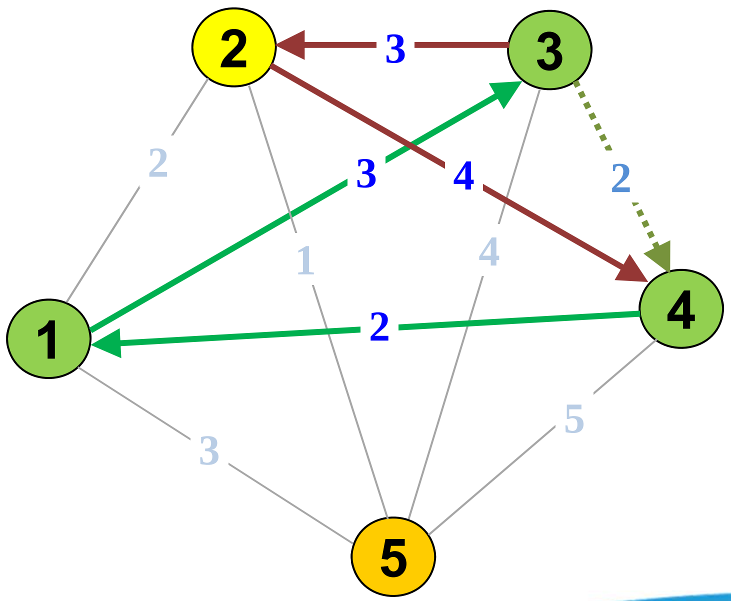
Step 4: The selected node 2 is inserted between node 1 and 3 in the subtour with the minimal increasing cost = 2.

(Selection)

1-2: 2
1-5: 3

3-2: 3
3-5: 4

4-2: 4
4-5: 5



(Insertion)

1-2-3: 3+2-3=2
3-2-4: 3+4-2=5

Heuristic - Nearest insertion (NI)

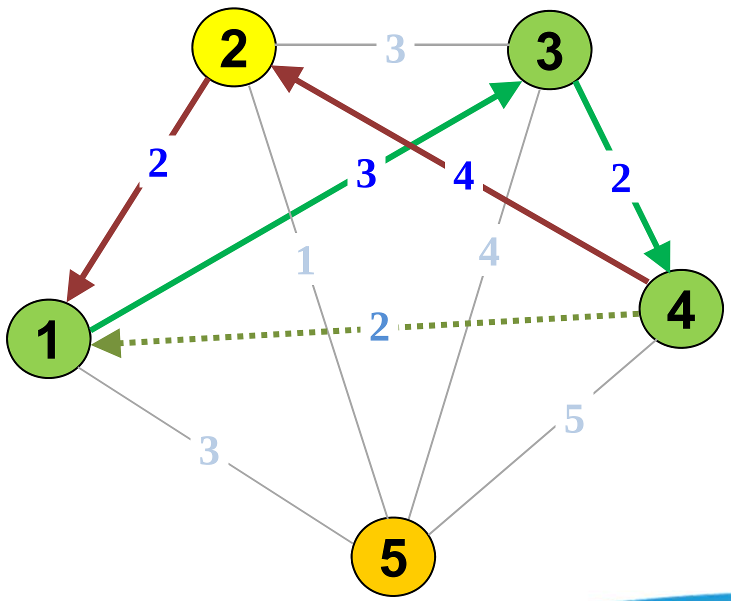
Step 4: The selected node 2 is inserted between node 1 and 3 in the subtour with the minimal increasing cost = 2.

(Selection)

1-2: 2
1-5: 3

3-2: 3
3-5: 4

4-2: 4
4-5: 5



(Insertion)

1-2-3: 3+2-3=2
3-2-4: 3+4-2=5
4-2-1: 2+4-2=4

Heuristic - Nearest insertion (NI)

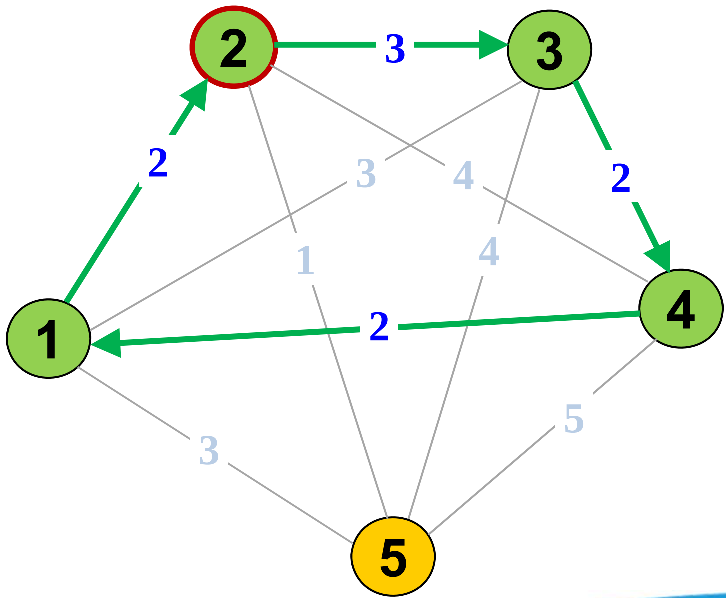
Step 4: The selected node 2 is inserted between node 1 and 3 in the subtour with the minimal increasing cost = 2.

(Selection)

1-2: 2
1-5: 3

3-2: 3
3-5: 4

4-2: 4
4-5: 5



(Insertion)

1-2-3: 3+2-3=2

3-2-4: 3+4-2=5

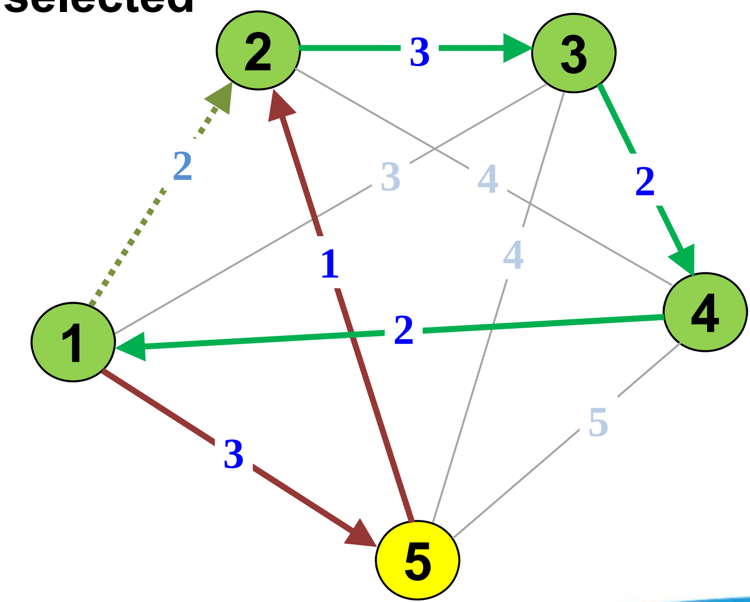
4-2-1: 2+4-2=4

Heuristic - Nearest insertion (NI)

Step 3: Node 5 is the only choice, so node 5 is selected.
Step 4: The selected node 5 is inserted between node 1 and 2 in the subtour with the minimal increasing cost = 2.

(Selection) **(Insertion)**

Only Node 5 can be selected



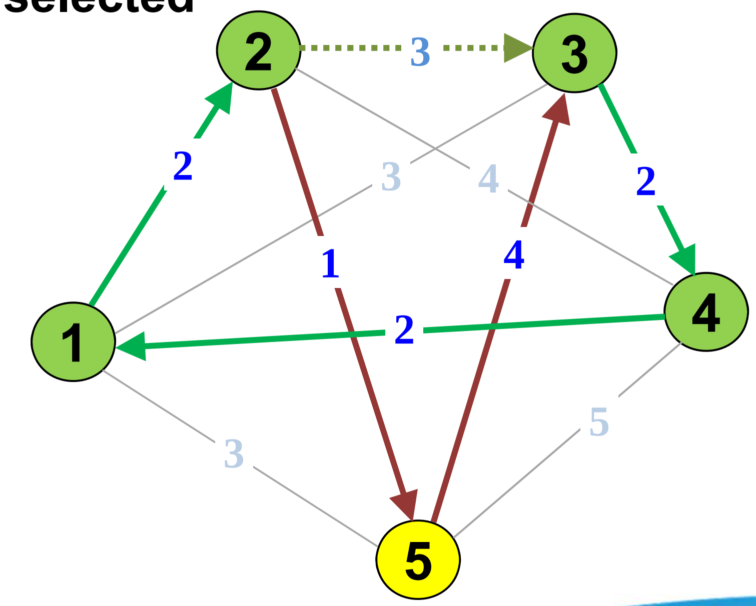
1-5-2: 3+1-2=2

Heuristic - Nearest insertion (NI)

Step 4: The selected node 5 is inserted between node 1 and 2 in the subtour with the minimal increasing cost = 2

(Selection)

Only Node 5 can be selected



(Insertion)

1-5-2: 3+1-2=2

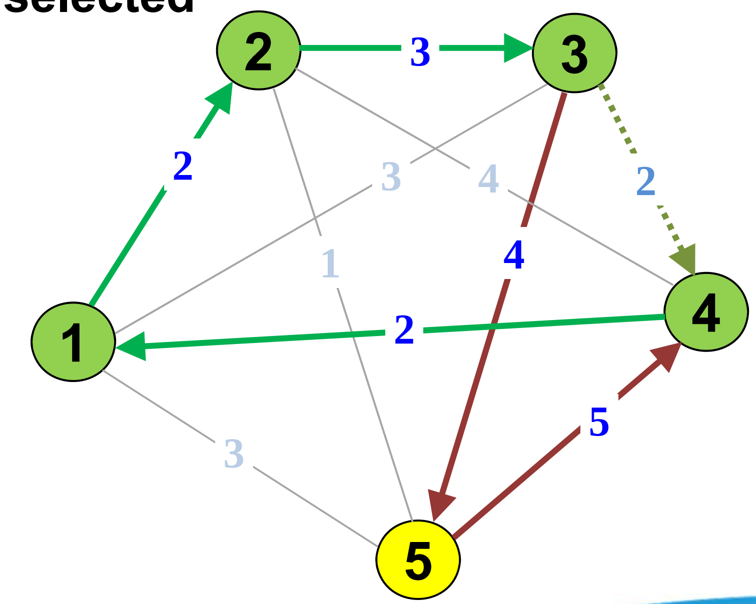
2-5-3: 1+4-2=3

Heuristic - Nearest insertion (NI)

Step 4: The selected node 5 is inserted between node 1 and 2 in the subtour with the minimal increasing cost = 2

(Selection)

Only Node 5 can be selected



(Insertion)

1-5-2: 3+1-2=2

2-5-3: 1+4-2=5

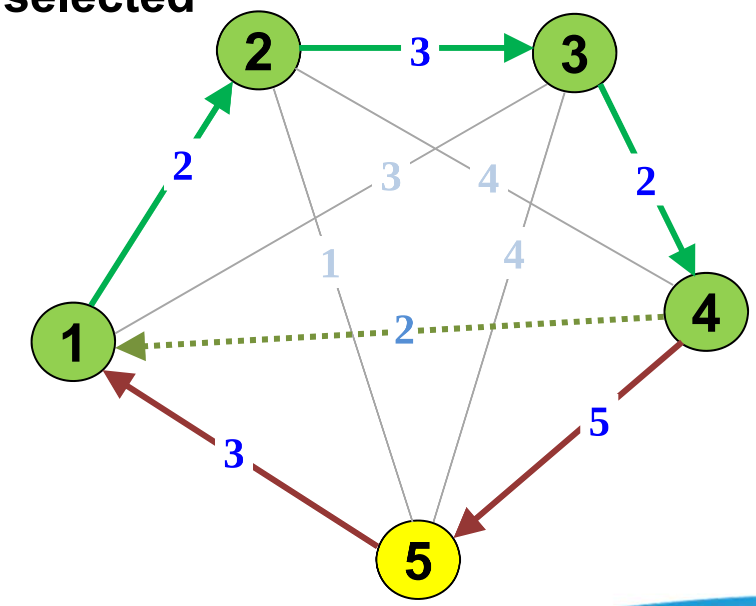
3-5-4: 4+5-2=7

Heuristic - Nearest insertion (NI)

Step 4: The selected node 5 is inserted between node 1 and 2 in the subtour with the minimal increasing cost = 2

(Selection)

Only Node 5 can be selected



(Insertion)

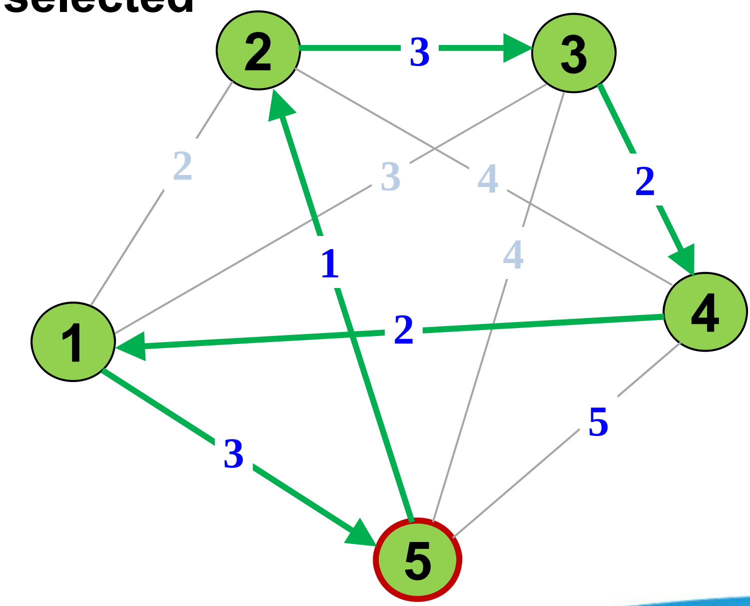
1-5-2: 3+1-2=2
2-5-3: 1+4-2=5
3-5-4: 4+5-2=7
4-5-1: 3+5-2=6

Heuristic - Nearest insertion (NI)

Step 4: The selected node 5 is inserted between node 1 and 2 in the subtour with the minimal increasing cost = 2 and total cost is 3+1+3+2+2 = 11

(Selection)

Only Node 5 can be selected



(Insertion)

1-5-2: 3+1-2=2

2-5-3: 1+4-2=5

3-5-4: 4+5-2=7

4-5-1: 3+5-2=6

Heuristic - Cheapest insertion (CI)

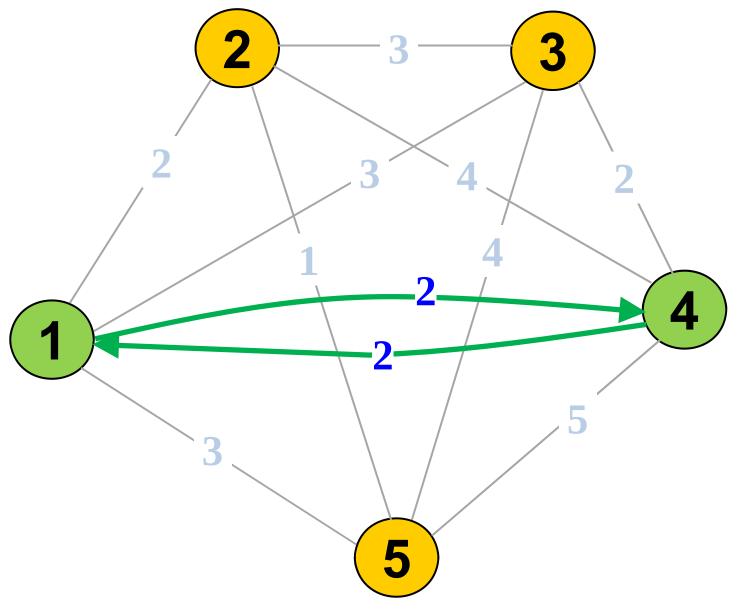
- Cheapest insertion for TSP
  1. Start with a subroute consisting of node i only.
  2. Find the $arc(i, j)$ in the subtour and node k, such that $c_{ik}+c_{kj}-c_{ij}$ is minimal. Then, insert k between i and j. (Insertion)
  3. Go to step3 unless we have a Hamiltonian cycle.

Heuristic - Cheapest insertion (CI)

Step 1: Suppose node 1 is chose as beginning.

Step 2: The node 4 is selected such that subtour with minimal cost $2c_{14} = 4$

Initial route: 1-4-1

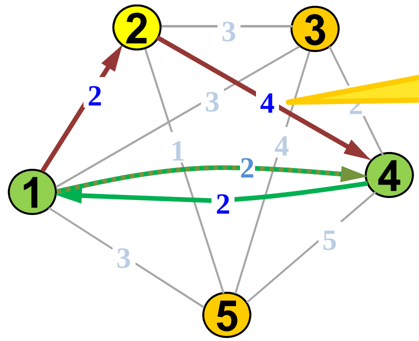


Heuristic - Cheapest insertion (CI)

Step 3: Find node k and insert it between node i and j in the subtour, such that the insertion cost is minimal, where $k \in \{2, 3, 5\}$.

(Insertion Cost)

1-2-4: 3+5-2=6



Insert node 2 into $arc(1, 4)$ with the increasing cost = 6.

Initial route: 1-4-1

Heuristic - Cheapest insertion (CI)

Step 3: Testing every enumerations, node 3 is inserted into $arc(1, 4)$ with the minimal insertion cost = 3.

(Insertion Cost)

1-2-4: 3+5-2=6

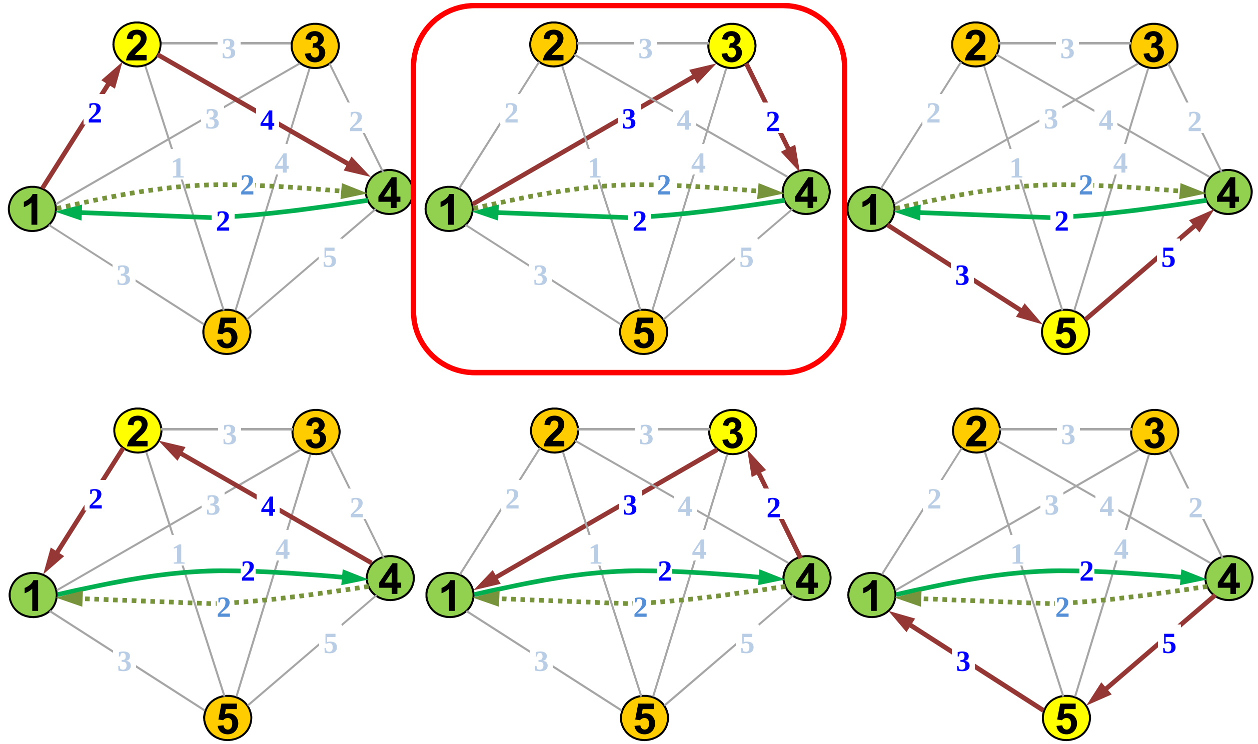
1-3-4: 3+2-2=3

1-5-4: 4+2-2=4

4-2-1: 2+4-2=4

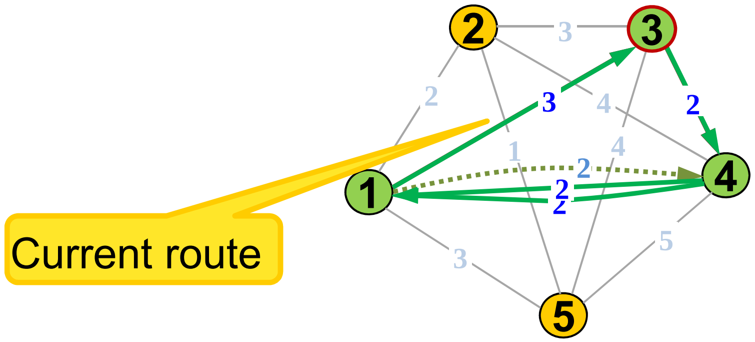
4-3-1: 2+3-2=3

4-5-1: 5+3-2=6



Heuristic - Cheapest insertion (CI)

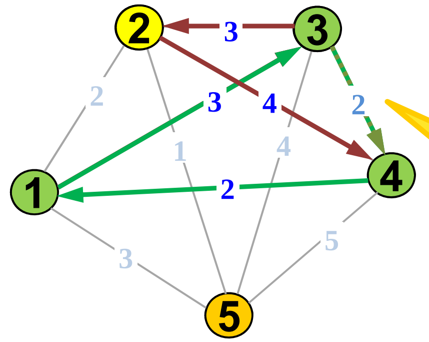
Step 3: Testing every enumerations, node 3 is inserted into $arc(1, 4)$ with the minimal insertion cost = 3.



Heuristic - Cheapest insertion (CI)

Step 3: Find node k and insert it between node i and j in the subtour, such that the insertion cost is minimal, where $k \in \{2, 5\}$.

(Insertion Cost)



Insert node 2 into $arc(3, 4)$ with the increasing cost = 5.

3-2-4: 3+4-2=5

Heuristic - Cheapest insertion (CI)

Step 3: Testing every enumerations, node 2 is inserted into $arc(1, 3)$ with the minimal insertion cost = 2.

(Insertion Cost)

1-2-3: 2+3-3=2

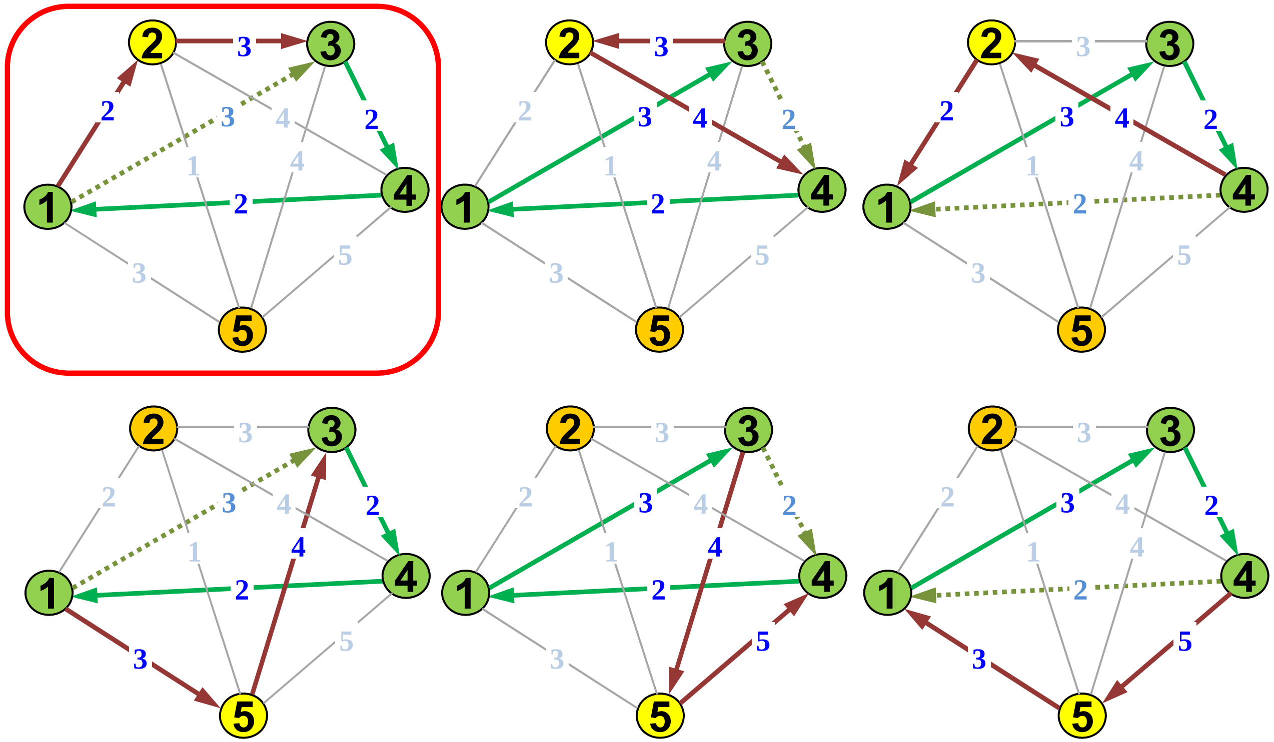
1-5-3: 3+4-3=4

3-2-4: 3+4-2=5

3-5-4: 4+5-2=7

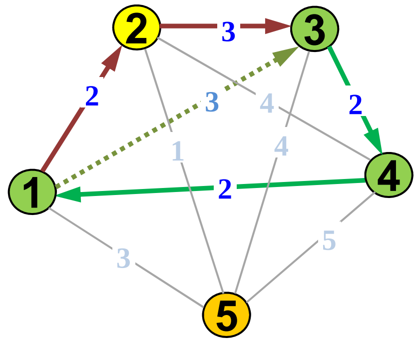
4-2-1: 4+2-2=4

4-5-1: 5+3-2=6



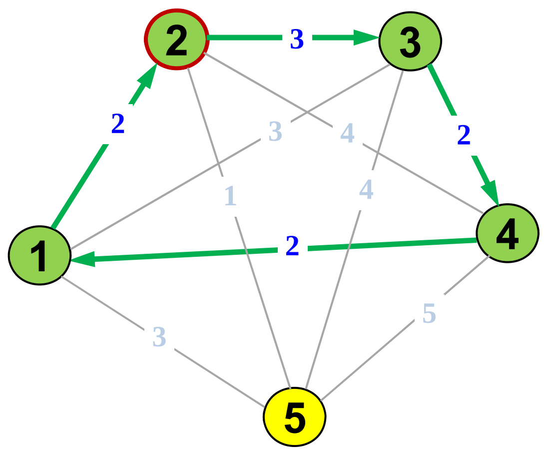
Heuristic - Cheapest insertion (CI)

Step 3: Testing every enumerations, node 2 is inserted into $arc(1, 3)$ with the minimal insertion cost = 2.



Heuristic - Cheapest insertion (CI)

Step 3: Testing every enumerations, node 2 is inserted into $arc(1, 3)$
with the minimal insertion cost = 2.

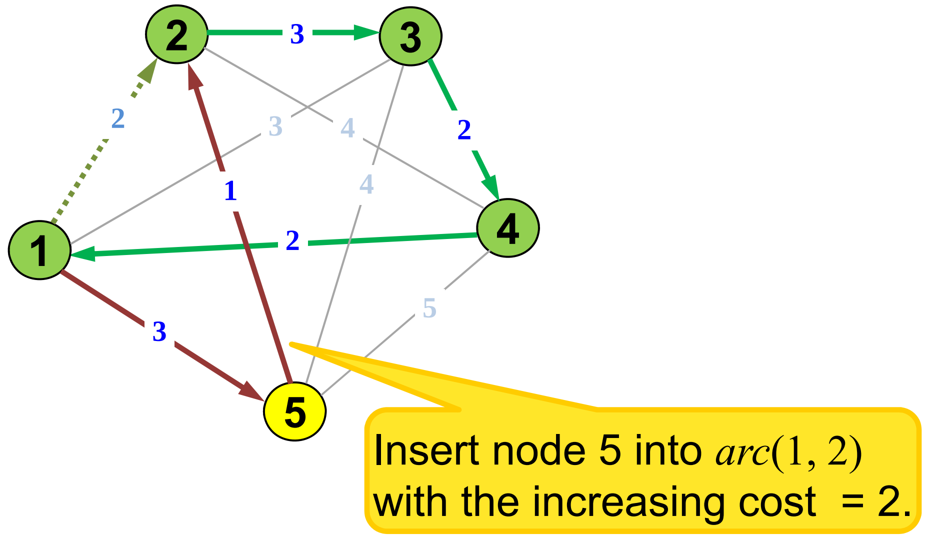


Heuristic - Cheapest insertion (CI)

Step 3: Find an $arc(i, j)$ in the subtour, which has the minimal insertion cost after inserting node 5.

(Insertion Cost)

1-5-2: 3+1-2=2



Heuristic - Cheapest insertion (CI)

Step 3: Testing every arcs, node 5 is inserted into $arc(1, 2)$ with the minimal insertion cost = 2.

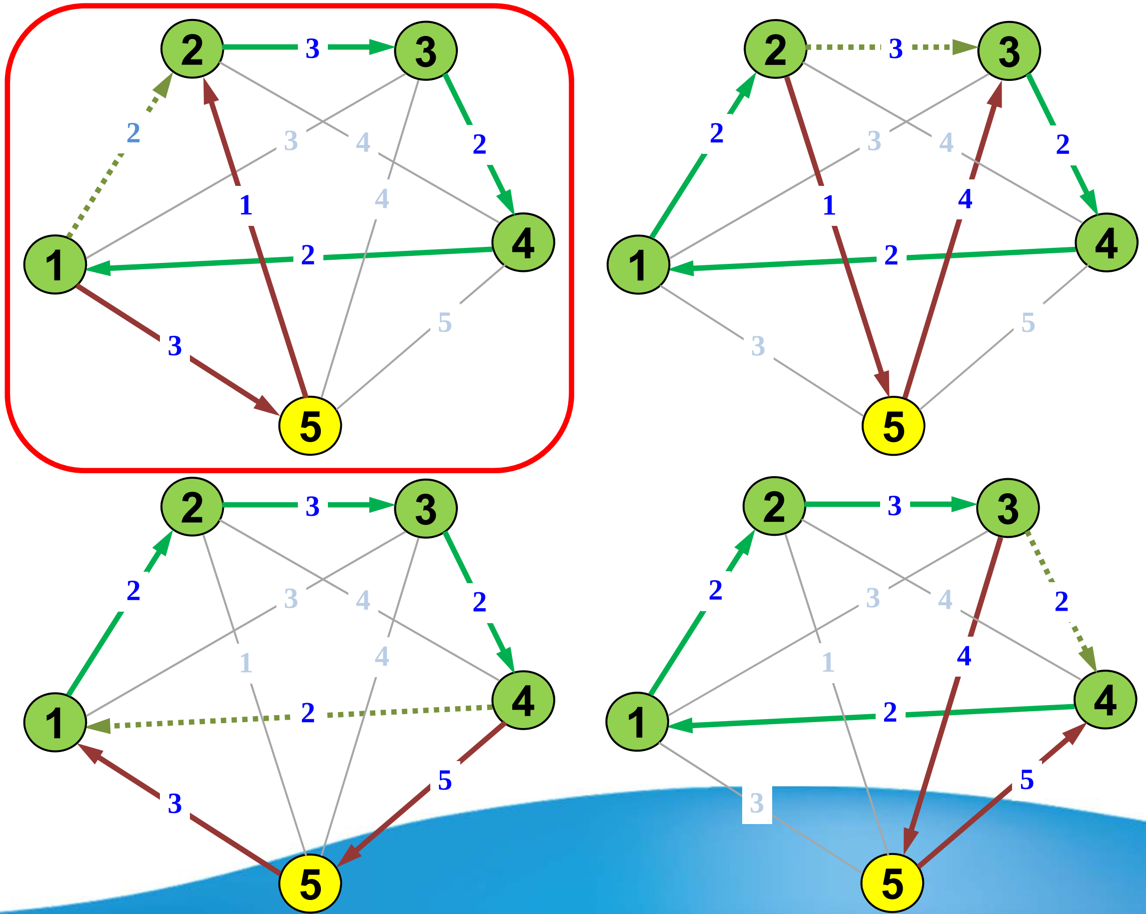
(Insertion Cost)

1-5-2: 3+1-2=2

2-5-3: 1+4-3=2

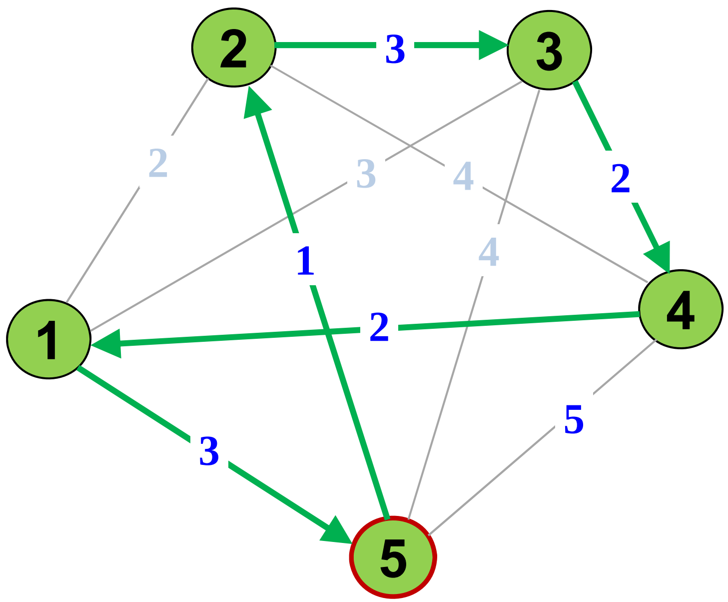
3-5-4: 4+5-2=7

4-5-1: 5+3-2=6



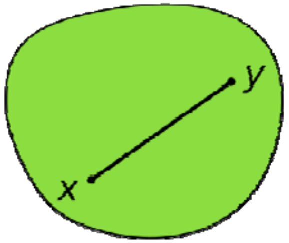
Heuristic - Cheapest insertion (CI)

Step 4: Node 5 is inserted between node 1 and 2 in the subtour and the total cost is 3+1+3+2+2 = 11

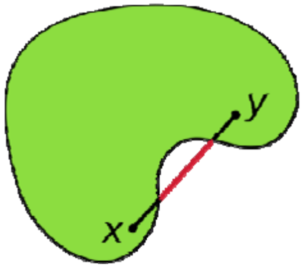


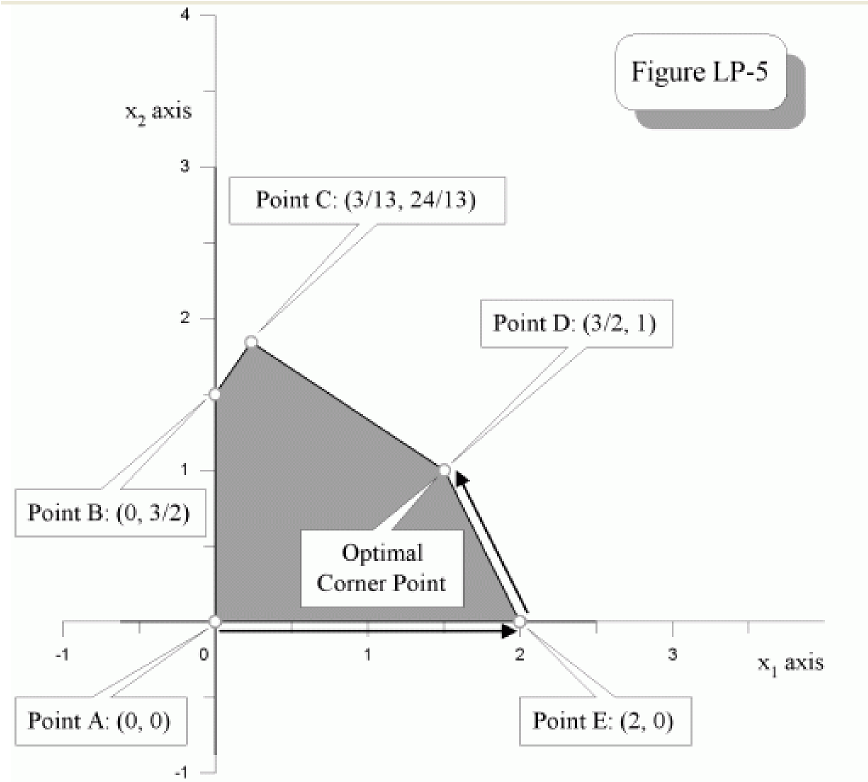
Local Search Algorithms

- Simplex method
- Convex



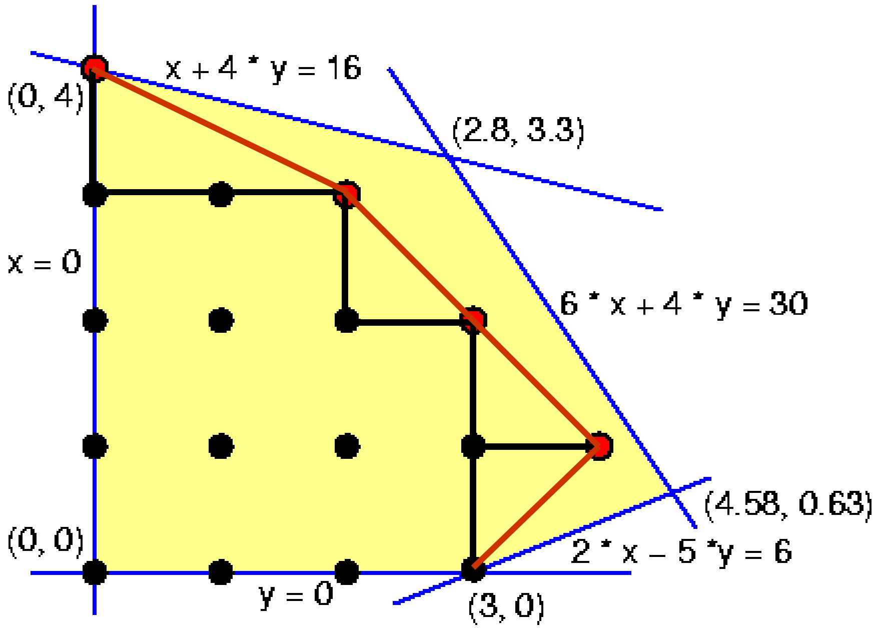
- Concave





Local Search Algorithms

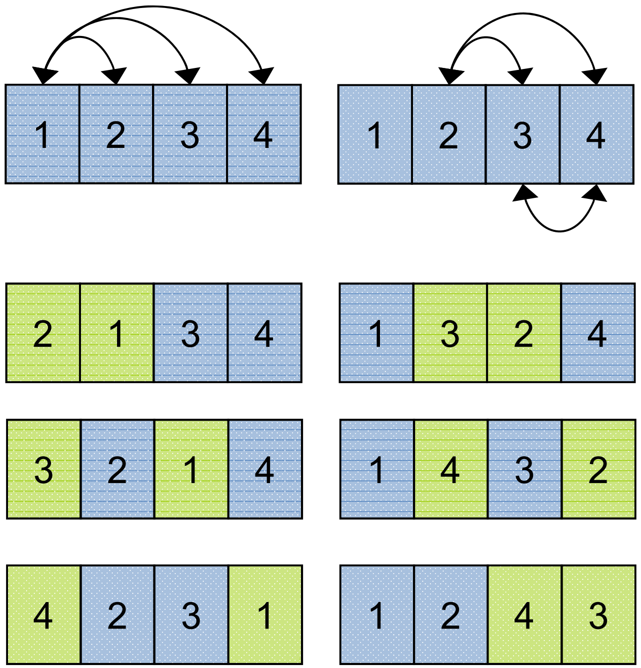
- Integer linear programming
- Combinatorial optimization:
  - Knapsack Problem
  - TSP
  - Vehicle routing problem (VRP)

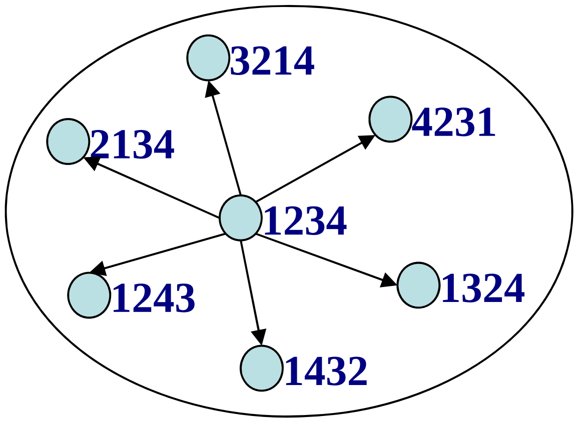


Local Search Algorithms

- Neighborhood
- Swap

  The neighborhood size of swap-based local search is $n(n\text{-}1)/2$

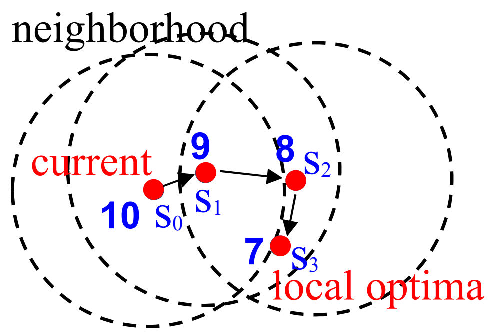




Local Search Algorithms

Local Search

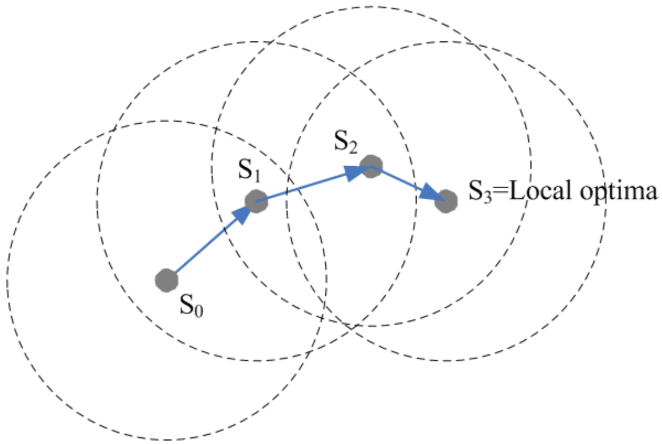
- Local search starts from a initial solution and then move to neighbor solution iteratively.
  - First improvement.
  - Best improvement.



Local Search Algorithms

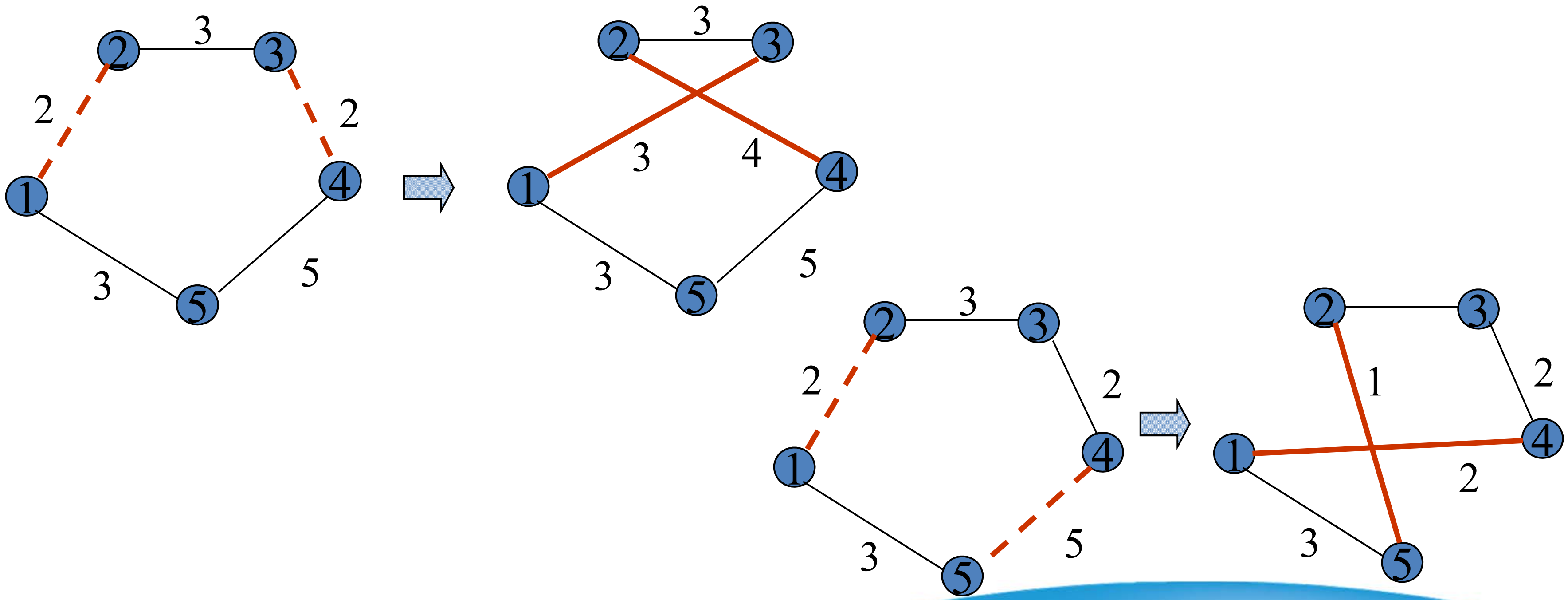
Local Search for TSP

- 2-opt
- k-opt
- OR-opt



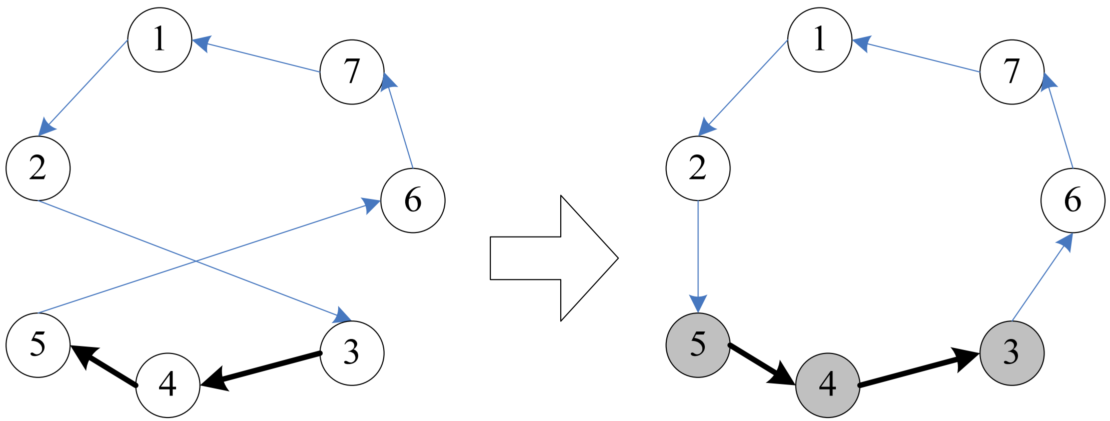
Local Search Algorithms – 2-opt

- The neighborhood size of 2-opt is $n\,(n\text{-}1)/2 - n$



Local Search Algorithms – 2-opt

Implementation of 2-opt with array

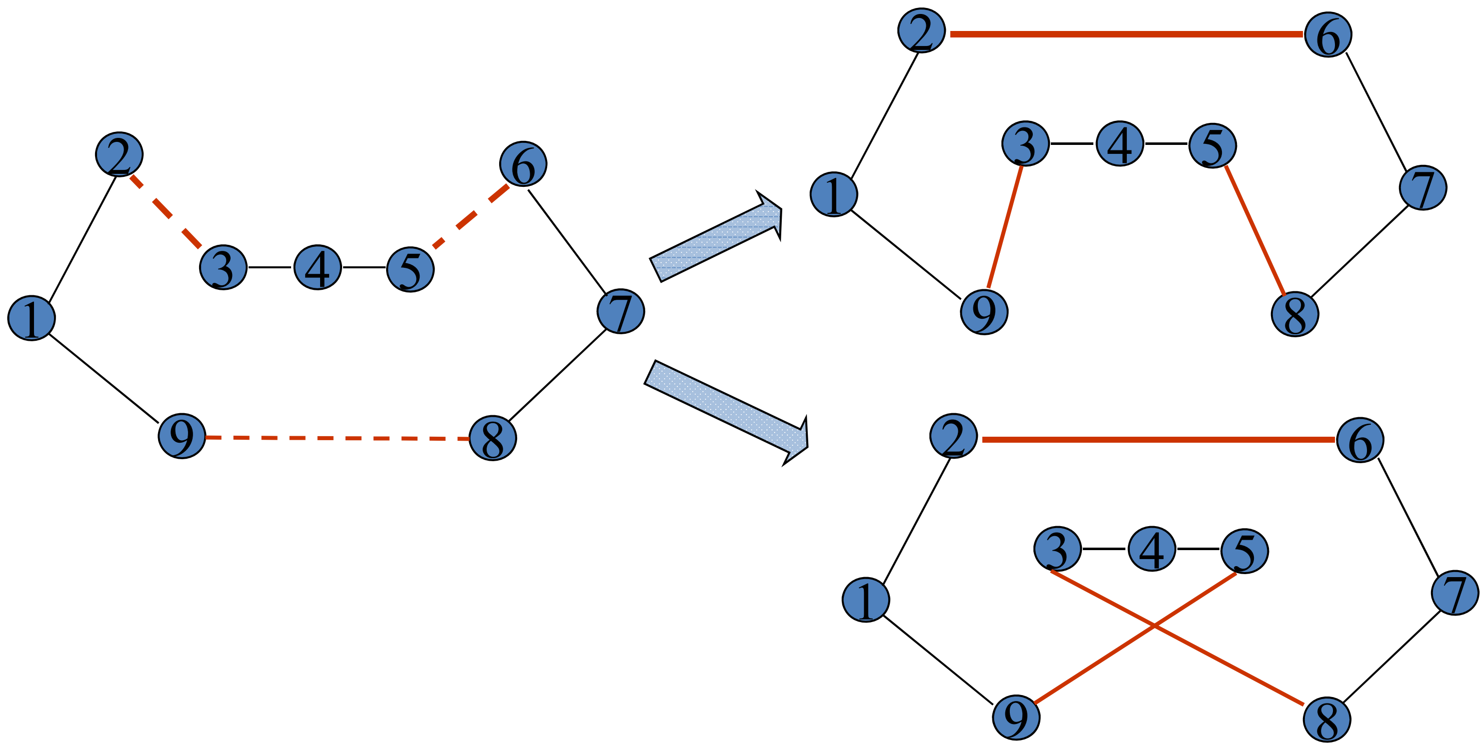


1	2	3	4	5	6	7

1	2	5	4	3	6	7

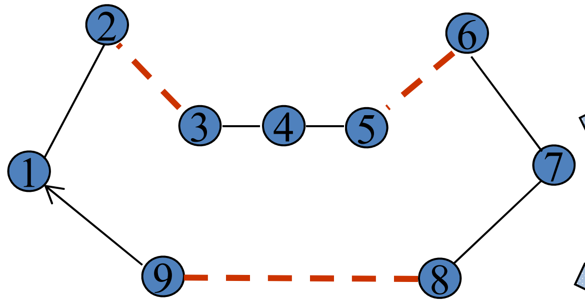
Local Search Algorithms – Or-opt

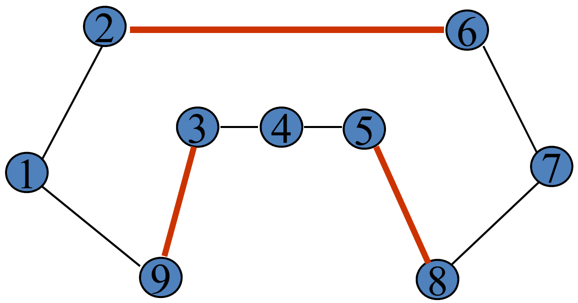
- The neighborhood size of 2-opt is $n(n-1)/2 - n$



Local Search Algorithms – Or-opt

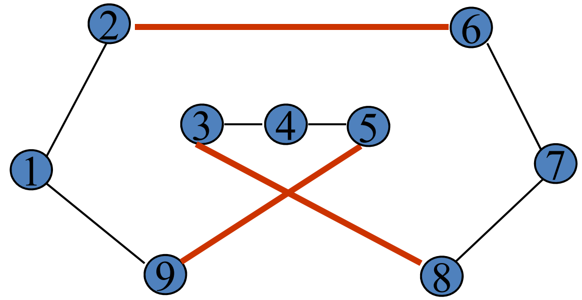
Implementation of Or-opt with array





1	2	6	7	8	**5**	**4**	**3**	9

1	2	3	4	5	6	7	8	9



1	2	6	7	8	**3**	**4**	**5**	9

TSP問題: 2-opt -練習 1

1. 假設有一個包含10個元素的陣列a[10]，嘗試在不需其他陣列的幫助下，將這個陣列元素反轉。
2. 嘗試在不需其他陣列的幫助下，將陣列中a[2]至a[7]的元素反轉。
3. 隨機產生兩個介於0-9的變數r1與r2，且此兩個變不可相同，嘗試在不需其他陣列的幫助下，將陣列中a[r1]至a[r2]的元素反轉。

1	2	3	4	5	6	7	8	9	10
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

→

1	2	3	7	6	5	4	8	9	10
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

TSP問題: 2-opt -練習 2

延續練習3，隨機產生兩個介於0-9的變數r1與r2，且此兩個變不可相同，嘗試在不需其他陣列的幫助下，將陣列中a[r1]至a[r2]的元素反轉。但在作業中需選若選擇的r1與r2使得| r1-r2 |>|10- (r1-r2) |，則反轉r1與r2外的陣列範圍，如圖所示。

case1: | r1-r2 |<=|10- (r1-r2) |

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	2	3	4	5	6	7	8	9	10

→

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	2	3	7	6	5	4	8	9	10

case2: | r1-r2 |>|10- (r1-r2) |

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	2	3	4	5	6	7	8	9	10

→

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
2	1	10	4	5	6	7	8	9	3

Metaheuristic (巨集啟發式演算法)

- Local Optima vs. Global Optima