logistics cost
↳ transportation cost
↳ storage cost

(FTL)

In-class Example

Choosing a Transportation Mode

Hypothetical company has:

- 1 plant, producing 1 product (→ Type)
- 1 warehouse, storing the product from which customers supplied directly
- Plant keeps 0 safety stock inventory, warehouse maintains 100 units
- Average customer demand rate at warehouse: 10 units/day (→ d)
- Plant produces 10 units/day

plant → warehouse → Customers

demand = production rate = 10 units/day

Suppose we would like to compare two modes of transportation for moving product from the plant to the warehouse. We want to make a mode choice decision that minimizes total cost.

Rail:

Capacity
- Shipments in railcars containing at most 400 units
- Cost: $1200 / carload
- Transit time: 20 days

Truck:

Capacity
- Truckloads of at most 100 units
- Cost: $700 / truckload
- Transit time: 3 days Inventory costs

Inventory cost:

500 — 506 — 512

Value of items at plant: $500 / unit

Value of items at warehouse: $512 / unit

Inventory holding cost rate: 25% of value per year held

Plant space cost: $20 / unit*year

Warehouse space cost: $25 / unit*year

	Rail	Truck
Capacity	400	100
Cost	$1200 / carload	$700 / truckload
Transit time	20 days	3 days

plant

S.S = Zero

10 unit/day
↳ produce

$500/unit

Holding = 25%

$20 / unit*year

warehouse

S.S = 100 units

10 unit/day
↳ = demand

$512/unit

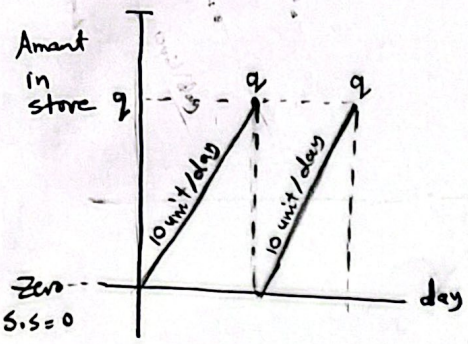
Holding = 0.25

$25/unit*year

Scanned with CamScanner

Scanned with CamScanner

plant



the maximum accumulation of units in the plant = q

So, inventory space is q units

Step 1 :-

$$\text{Space rent cost} = \frac{\$20}{\text{unit} \cdot \text{year}} \times q\ \text{unit} = \$20q/\text{year}$$

Step 2 :-

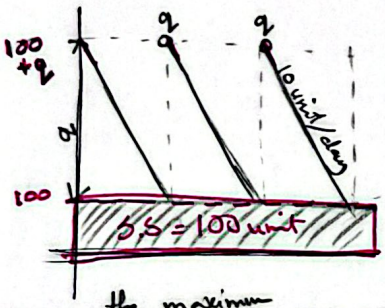
$$\text{Holding cost} = \frac{25}{100\ \text{year}} \times \frac{\$500}{\text{unit}} \times \frac{q}{2}\ \text{unit}$$

$$= \$62.5q/\text{year}$$

annual total Inventory cost "plant" $= \$(20q + 62.5q)/\text{yr}$

$= \$82.5q/\text{year}$

Warehouse



the maximum accumulation = $100 + q$ unit

Step 3

$$\text{Space rent cost} = \frac{\$25}{\text{unit} \cdot \text{year}} \times (100 + q)\ \text{unit} = (2500 + 25q)\$/\text{year}$$

Step 4

$$\text{Holding cost} = \frac{25}{100\ \text{year}} \times \frac{\$512}{\text{unit}} \times \left(100 + \frac{q}{2}\right)\text{unit}$$

$$= \$(12{,}800 + q\,64)/\text{year}$$

Tot. Annual Inventory cost "warehouse" $= (2{,}500 + 12{,}800) + (25 + 64)q = \$(15{,}300 + 89q)/\text{year}$

Step (5) pipline inventory cost = Holding cost of inventory during transportation (Intransit)

× doesn't depend on the shipment size (q), only depends on the mode of transportation

① Value at pipline $= \frac{500 + 512}{2} = \$506/\text{unit}$

$$\text{Pipline Inventory cost ("Holding" by rail)} = \frac{25}{100\ \text{year}} \times \frac{\$506}{\text{unit}} \times \frac{3{,}650\ \text{unit}}{\text{year}} \times 20\ \text{days} \times \frac{1\ \text{year}}{365\ \text{day}}$$

$= \$25{,}300/\text{year}$

↳ constant, since it doesn't depend on q

Annual transported units $= \frac{10\ \text{unit}}{\text{day}} \times 365\ \frac{\text{day}}{\text{year}}$

$= 3{,}650\ \text{unit/year}$

↳ Annual production = # of transported units during the whole year

Goal :

- Our goal is to determine the optimal shipment size q for each mode, and the total cost of the optimally configured system.

Assumptions:

- Assume for now that we will never ship more than a single railcar or a single truckload at a time when minimizing cost

1. What are the annual inventory holding costs at the production plant? (Include the space and holding cost in this calculation.)
2. What are the annual inventory holding costs at the warehouse (Include the space and holding cost in this calculation.)
3. What are the pipeline inventory costs for each mode?
4. What are the transportation costs for each mode?
5. What are the total cost equations for each mode of transportation?

pipline Inventory cost = Holding by truck $= \frac{25}{100\,year} \circledast \frac{\$506}{unit} \circledast \frac{3650\,unit}{year} * 3\,days * \frac{1\,year}{365\,days}$

$= \$3,795/year$

(steps) Transportation cost : regardless of the chosen mode, first we determine number of required ($shipment/year$) as a fun. of shipment size

Tot. Amount Shipped/year is $\frac{10\,unit}{day} * \frac{365\,day}{year} = 3650\,unit/year$

# of shipment per year $= \left(\frac{3,650}{q}\right) \frac{truckload}{year}$ $q \ll$ Capacity

tot. transportation cost by rail $= \frac{\$1200}{carload} * \left(\frac{3650}{q_{rail}}\right) \frac{carload}{year} = \frac{\$4,380,000}{q_{rail} \cdot year}$

tot transportation cost by truck $= \frac{\$700}{truckload} * \left(\frac{3650}{q_{truck}}\right) \frac{truckload}{year} = \frac{\$2,555,000}{q_{truck}\,year}$

if there's a handling cost "load is & unloading" $/truck

Transportation cost $= \$(2700 + 100) \left(\frac{3650}{q_{truck}}\right)/year$

Handling

Scanned with CamScanner

Scanned with CamScanner

Tot. cost = Rail

$$= 82.5\,q_{rail} + 89\,q_{rail} + 15,300 + 25,300 + \frac{4,380,000}{q_{rail}} = 171.5\,q_{rail} + 40,600 + \frac{4,380,000}{q_{rail}}$$

$$\frac{d\,TC}{dq} = 71.5 - \frac{4,380,000}{q^2_{rail}} = 0$$ to find the optimal value of q_{rail}

$$171.5 = \frac{4,380,000}{q^2_{rail}} \rightarrow q_{rail} = \sqrt{\frac{4,380,000}{171.5}} = 159.8 = 160 \text{ units}$$

$160 \leq 400$ ✓

$$TC(160 \text{ units}) = 171.5(160) + 40,600 + \frac{4,380,000}{160} = \$95,415/\text{year}$$

$$TC_{truck} = 82.5\,q_{truck} + 15,300 + 89\,q_{truck} + 3795 + 2,555,000$$

$$TC(q_{truck}) = 171.5\,q_{truck} + 19,095 + \frac{2,555,000}{q_{truck}}$$

$$\frac{dTC}{dq} = 171.5 + -\frac{2,555,000}{q^2_{truck}} = 0 \rightsquigarrow q_{truck} = \sqrt{\frac{2,555,000}{171.5}} = 122.05 \text{ units}$$

$122.05 > 100$ ✗ not feasible

so $q_{optimal} = 100$

$TC_{truck}(100) = 61,795\ \$/\text{year}$

if it must be full_load - truck so, we take q = the capacity

truck $TC(100) = 61,795\ \$/\text{year}$ $TC_{rail}(400)$??

$v(j)$ $pred(j)$

Cost of the path from (s) to $d(j)$

node predecessor node j in the min cost path

Dijkstra's algorithm → "Can be implemented on directed & undirected graphs"

- **Requires that $c_{ij} \geq 0$**

Initialization:

$$v(j) \leftarrow \begin{cases} 0 & j = s \\ +\infty & \text{otherwise} \end{cases}$$

$$pred(j) \leftarrow \begin{cases} 0 & j = s \\ -1 & \text{otherwise} \end{cases}$$

$PERM = \emptyset$; $\overline{PERM} = N$

For source: 0 (0)

others: ∞ (-1)

Visited: ∅

un-visited: U

Iterations:

while $\overline{PERM} \neq \emptyset$:

1. **Let $p \in \overline{PERM}$ be node for which $v(p) = \min\{v(j), j \in \overline{PERM}\}$**
2. **$PERM \leftarrow PERM \cup \{p\}$** p → visited
3. **$\overline{PERM} \leftarrow \overline{PERM} \setminus \{p\}$**
4. **For every arc $(p, j) \in A$, $j \in \overline{PERM}$ leading from node p:**
   (a) **if $v(p) + c_{pj} < v(j)$, then:**
   - **$v(j) \leftarrow v(p) + c_{pj}$**
   - **$pred(j) \leftarrow p$**

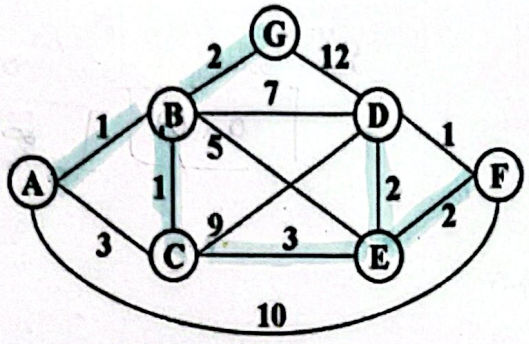
1. Assign to every node a distance value. Set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes as unvisited. Set initial node as current.
3. For current node, consider all its unvisited neighbors and calculate their tentative distance (from the initial node). For example, if current node (A) has distance of 6, and an edge connecting it with another node (B) is 2, the distance to B through A will be 6+2=8. If this distance is less than the previously recorded distance (infinity in the beginning, zero for the initial node), overwrite the distance.
4. When we are done considering all neighbors of the current node, mark it as visited. A visited node will not be checked ever again; its distance recorded now is final and minimal.
5. If all nodes have been visited, finish. Otherwise, set the unvisited node with the smallest distance (from the initial node) as the next "current node" and continue from step 3.

Scanned with CamScanner

Unique cost but not unique path (2 maybe get 2 distinct trees but one

Example # 1: Given the graph in the Figure shown below, find the shortest paths to all nodes using the vertex A as the starting source. Use Dijkstra's algorithm. unique cost)

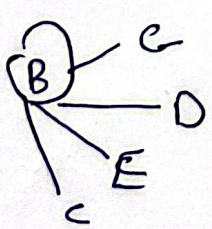
$V_{(j)}$ Pred (j)

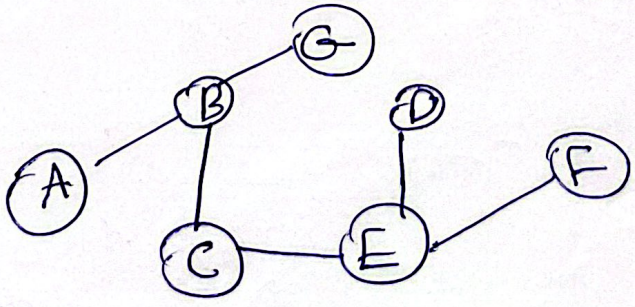


Source

Visited Un-visited

Iteration	A	B	C	D	E	F	G	P	PERM	PERM
0	0 (0)	∞ (-1)	∞ (-1)	∞ (-1)	∞ (-1)	∞ (-1)	∞ (-1)		∅	N
1	0 (0)*	1 (A)	3 (A)	∞ (-1)	∞ (-1)	10 (A)	∞ (-1)	A	A	N / {A}
2	—	1* (A)	2 (B)	8 (B)	6 (B)	10 (A)	3 (B)	B	A, B	N / {A, B}
3	—	—	2* (B)	8 (B)	5 (C)	10 (A)	3 (B)	C	A, B, C	N / {A, B, C}
4	—	—	—	8 (B)	5 (C)	10 (A)	3* (B)	G	A, B, C, G	N / {A, B, C, G}
5	—	—	—	7 (E)	5* (C)	7 (E)	— (Choose any)	E	A, B, C, G, E	F, D
6	—	—	—	7* (E)	—	7* (E)	—	D	N-D	F
7	—	—	—	—	—	7* (E)	—	F	N	∅



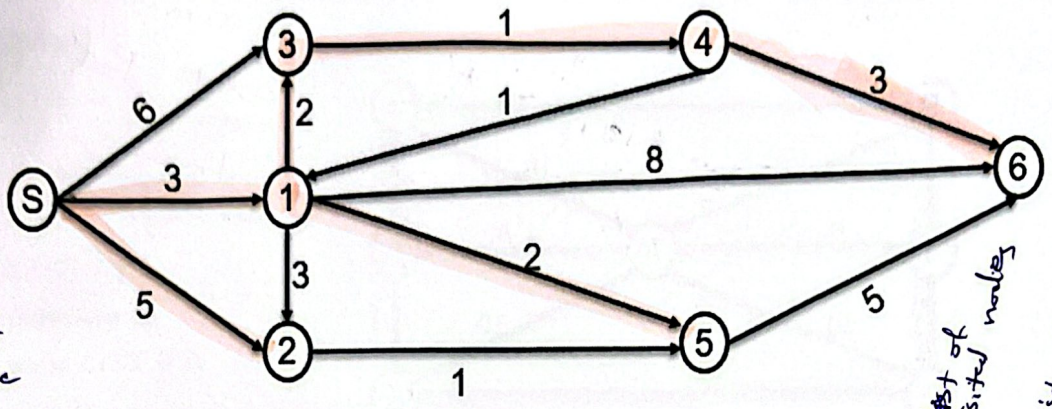


Scanned with CamScanner

V(j) ← cost of the path (S to j)

pred (j) ← node predecessor of node j in the current min cost path

Example # 2: Given the graph in the Figure shown below, find the shortest paths to all nodes using the vertex S as the starting source. Use Dijkstra's algorithm



min cost tree

list of visited nodes

list of unvisited nodes

current node

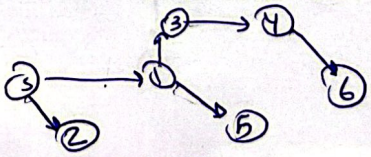
Iteration	S	1	2	3	4	5	6	P	PERM	$\overline{PERM}$
0	0(0)	∞ (-1)	∞ (-1)	∞ (-1)	∞ (-1)	∞ (-1)	∞(-1)		∅	N
1	0(0*)	3 (S)	5 (S)	6 (S)	∞(-1)	∞ (-1)	∞ (-1)	S	S	N / {S}
2	—	3* (S)	5(S)	5(1)	∞ (-1)	5(1)	11(1)	1	S, 1	N / {S, 1}
3	—	—	5 (S)*	5(1)	∞(-1)	5(1)	11(1)	2	S, 1, 2	3, 4, 5, 6
4	—	—	—	5*(1)	6(3)	5(1)	11(1)	3	S, 1, 2, 3	4, 5, 6
5	—	—	—	—	6(3)	5*(1)	10(5)	5	S, 1, 2, 3, 5	4, 6
6	—	—	—	—	6(3)*	—	9(4)	4	S, 1, 2, 3, 4, 5	6
7	—	—	—	—	—	—	9(4)*	6	N	∅

Initialization

(7 + 1) = # of iterations

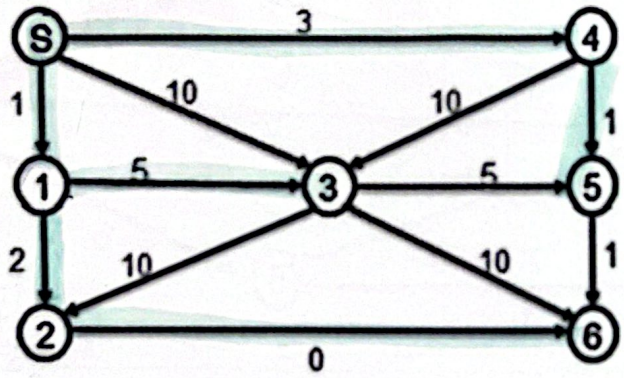
(7 nodes, 1 Initialization)

* Unique path from (S) to any node with the minimal cost

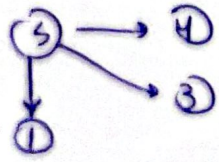


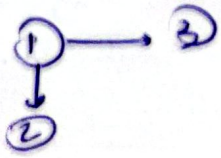
Given the graph in the Figure shown below, find the shortest paths to all nodes using the vertex s as the starting source.

minimum cost Path from



Iteration	S	1	2	3	4	5	6	P	PERM	PERM
0	0(0)	∞(-1)	∞(-1)	∞(-1)	∞(-1)	∞(-1)	∞(-1)		ϕ	N
1	0(0)*	1(S)	∞(-1)	10(S)	3(S)	∞(-1)	∞(-1)	S	S	N/{S}
2	—	1(S)*	3(1)	6(1)	3(S)	∞(-1)	∞(-1)	1	S,1	N/{S,1}
3	—	—	3*(1)	6(1)	3(S)	∞(-1)	3(2)	2	S,1,2	N/{S,1,2}
4	—	—	—	6(1)	3(S)*	4(4)	3(2)	4	S,1,2,4	3,5,6
5	—	—	—	6(1)	—	4(4)	3(2)*	6	S,1,2,4,6	3,5
6	—	—	—	6(1)	—	4*(4)	—	5	N(3)	3
7	—	—	—	6(1)*	—	—	—	3	N	ϕ





Scanned with CamScanner

ertex s

Bellman-Ford algorithm → Could have negative weight for arcs

Sometimes, *label-correcting* algorithm. No requirement of non-negative arc costs. Often more efficient in practice for sparse transportation problems.

***Initialization*:**

$$v(j) \leftarrow \begin{cases} 0 & j = s \\ +\infty & \text{otherwise} \end{cases}$$

$$pred(j) \leftarrow \begin{cases} 0 & j = s \\ -1 & \text{otherwise} \end{cases}$$

$LIST = \{s\}$

***Iterations*:**

while $LIST \neq \emptyset$:

1. Remove node p from top of $LIST$
2. For every arc (p, j) leading from node p:
   (a) if $v(p) + c_{pj} < v(j)$, then:
   - $v(j) \leftarrow v(p) + c_{pj}$
   - $pred(j) \leftarrow p$
   - if $j \notin LIST$, $LIST \leftarrow LIST + \{j\}$

papes modification

if $j \notin$ LIST (j isn't in the current list)

2 cases

↳ if j has never been in the list before

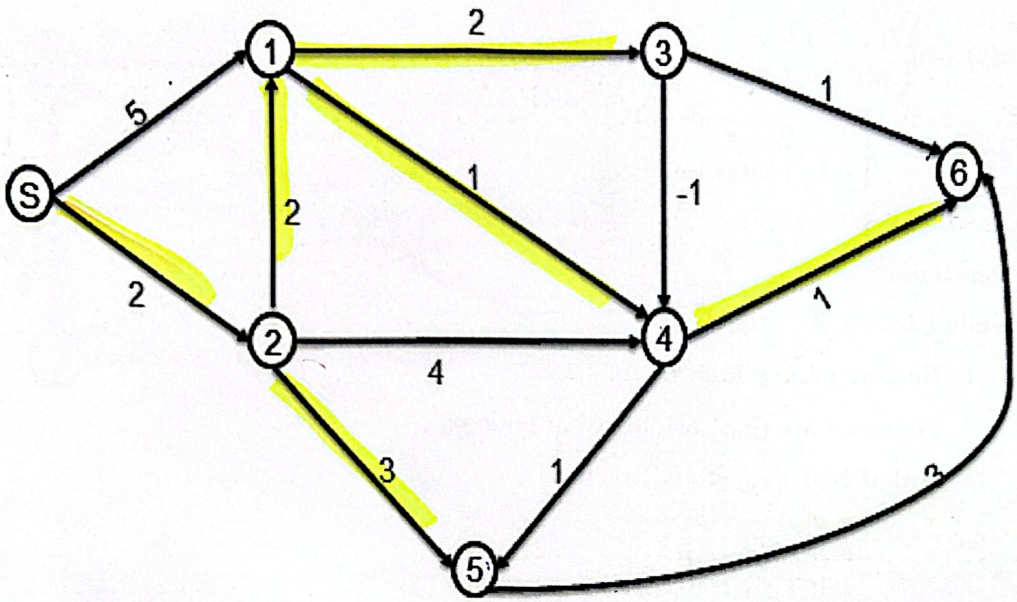
$$LIST \leftarrow LIST + \{j\}$$

↳ if j has been in the list before the

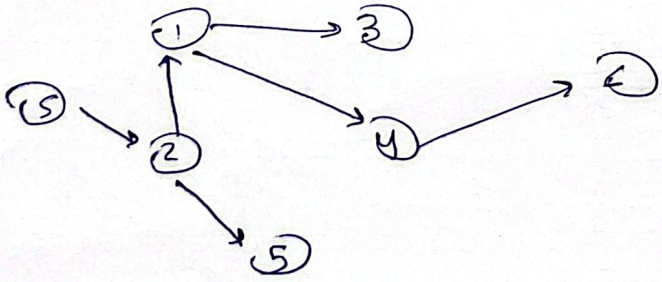
$$LIST \leftarrow \{j\} + LIST$$

directed only

Example:
Given the graph in the Figure shown below, find the shortest paths to all nodes using the vertex s as the starting source. Use Bellman-Ford algorithm.



Iteration	s	1	2	3	4	5	6	P	LIST
0	0(0)	∞(-1)	∞(-1)	∞(-1)	∞(-1)	∞(-1)	∞(-1)	—	(S)
1	0(0)*	5(S)	2(S)	∞(-1)	∞(-1)	∞(-1)	∞(-1)	S	(1), 2
2	0(0)	5(S)*	2(S)	7(1)	6(1)	∞(-1)	∞(-1)	(1)	(2), 3, 4,
3	0(0)	4(2)	2(S)*	7(1)	6(1)	5(2)	∞(-1)	2	1, 3, 4, 5
4	0(0)	4(2)*	2(S)	6(1)	5(1)	5(2)	∞(-1)	1	3, 4, 5
5	0(0)	4(2)	2(S)	6(1)*	5(1)	5(2)	7(3)	3	4, 5, 6
6	0(0)	4(2)	2(S)	6(1)	5(1)*	5(2)	6(4)	4	5, 6
7	0(0)	4(2)	2(S)	6(1)	5(1)	5(2)*	6(4)	5	6
8	0(0)	4(2)	2(S)	6(1)	5(1)	5(2)	6(4)	6	∅



minimum cost
Path tree

Node

(Cost, time)

lable

Multi-label algorithm

Initialization:

Initially, the origin node s is given label $(0,0)_1^s$. (The subscript indicates the reference number $k = 1$ for this label, and the superscript indicates that this label exists at node s) All other nodes receive no labels. The predecessor for this label, $pred(1)$ is 0.

$LIST = \{(0,0)_1^s\}$ $r = 2$

Iterations:

while $LIST \neq \emptyset$:

1. Remove label k from the beginning of the list. Suppose label k has components $(d_k, t_k)_k^p$, so that it exists at node p
2. For every arc (p, j) leading from node p:
   (a) Create a new label r at node j where $(d_r, t_r) = (d_k + d_{pj}, t_k + t_{pj})$
   (b) Define $pred(r) = k$
   (c) Add label r to the end of the $LIST$
   (d) Compare label r to each existing label (d_q, t_q) currently at node j:
      i. If $d_r \geq d_q$ -and- $t_r \geq t_q$, then label r is dominated by (d_q, t_q). Discard the new label, and remove it from $LIST$.
      ii. Else If $d_q \geq d_r$ -and- $t_q \geq t_r$, then label r dominates label q. Discard the label (d_q, t_q), and if necessary remove it from $LIST$.
      iii. Else retain both labels

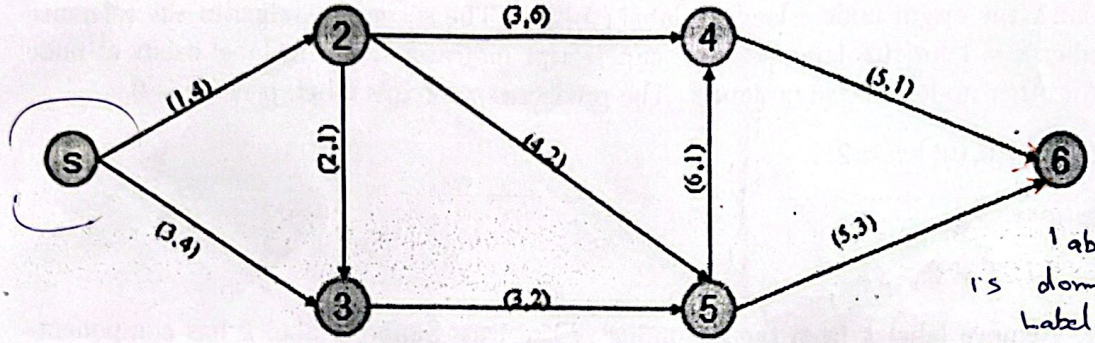
Stopping criterion:

When no labels remain in $LIST$, this indicates that no remaining label corrections can be found, and all of the paths have been found.

Scanned with CamScanner

(distance, time) node label number

Example: Given the graph in the Figure shown below, find the shortest paths to all nodes using the vertex s as the starting source. Use **Multi-label algorithm**



label 4 is dominated by Label3

	S	2	3	4 (2 trees)	5	6 (3 trees)	LIST
Labels	$(0,0)^{s}_{1}$ pred(1)=s	$(1,4)^{2}_{2}$ pred(2)=1	$(3,4)^{3}_{3}$ pred(3)=1	$(4,10)^{4}_{5}$ pred(5)=2	$(5,6)^{5}_{6}$ pred(6)=2	$(9,11)^{6}_{8}$ pred(8)=5	~~$(0,0)^{s}_{1}$~~
			~~$(3,5)^{3}_{4}$~~ pred(4)=3	$(11,7)^{4}_{10}$ pred(10)=6	~~$(6,6)^{5}_{7}$~~ pred(7)=3	$(10,9)^{6}_{9}$ pred(9)=6	$(1,4)^{2}_{2}$
						~~$(16,8)^{6}_{11}$~~ pred(11)=10	$(3,4)^{3}_{3}$
							~~$(3,5)^{3}_{4}$~~
							~~$(4,10)^{4}_{5}$~~
							$(5,6)^{5}_{6}$
							~~$(6,6)^{5}_{7}$~~
							~~$(9,11)^{6}_{8}$~~
							$(10,9)^{6}_{9}$
							~~$(11,7)^{4}_{10}$~~
							~~$(16,8)^{6}_{11}$~~

Determine non-dominated min cost path from S to 6

If undirected
Le Dijestra

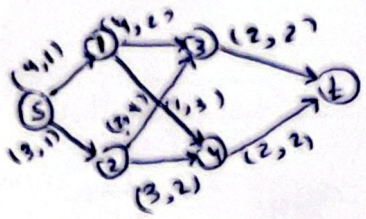
directed → Multi-label
& 2 objectives

Minimum Cost Paths

Question .

Suppose you have gone to the grocery store and stocked up with a week of groceries. You would like to make it home in a reasonable amount of time without traveling too far so that your ice cream does not melt. The grocery store will be your starting location, referred to as *s*, and your house will be the final destination, referred to as *t*. The table below shows all connections in the transportation network you will use to travel home (if a connection is not in the table, that arc does not exist in the network). The table also gives the distance and time between any pair of nodes for which a connection exist.

directed



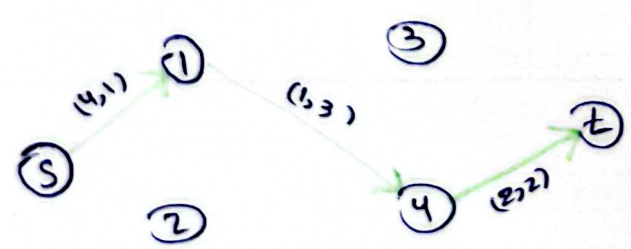
Connection	Distance	Time
(s,1)	4	1
(s,2)	3	1
(1,3)	4	2
(1,4)	1	3
(2,3)	2	4
(2,4)	3	2
(3,t)	2	2
(4,t)	2	2

(a) Draw the network, labeling each node with the appropriate node number $\{s,t,1,2,3,4\}$ and labeling each arc with the cost (d_{ij},t_{ij}) where d_{ij} is the distance of the connection and t_{ij} is the time of the connection.

(b) Use the **multi-label algorithm** to find the minimum cost paths from the grocery store *s* to your home *t*, where the two types of costs being considered are described below. Show your work in the form of a table as we did in class.

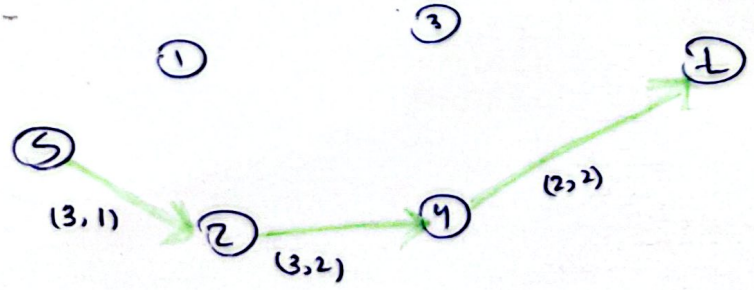
i. Find the minimum distance path from *s* to *t*. (=,)

ii. Find the path from *s* to *t* that minimizes the total time. (, =)



$(7,6)^t_q$
=

min distance
← Path

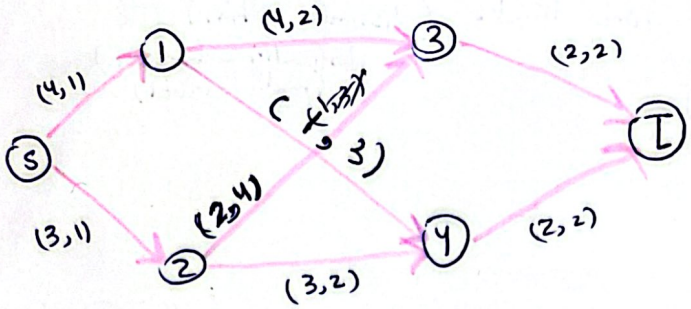


$(8,5)^t$
= 11

min Time
← Path

Scanned with CamScanner

Solution:
The network



label "10" is dominated by label "9"

Node s	Node 1	Node 2	Node 3	Node 4	Node T	LIST
$(0,0)^S_1$ pred(1) = s	$(4,1)^1_2$ pred(2) = 1	$(3,1)^2_3$ pred(3) = 1	$(8,3)^3_4$ pred(4) = 2 $(5,5)^3_6$ pred(6) = 3	$(5,4)^4_5$ pred(5) = 2 $(6,3)^4_7$ pred(7) = 3	$(10,5)^T_8$ pred(8) = 4 $(7,6)^T_9$ pred(9) = 5 $(7,7)^T_{10}$ pred(10) = 6 $(8,5)^T_{11}$ pred(11) = 7	$(0,0)^S_1$ $(4,1)^1_2$ $(3,1)^2_3$ $(8,3)^3_4$ $(5,4)^4_5$ $(5,5)^3_6$ $(6,3)^4_7$ $(10,5)^T_8$ $(7,6)^T_9$ $(7,7)^T_{10}$ $(8,5)^T_{11}$

label "8" is dominated by label "11"

Scanned with CamScanner

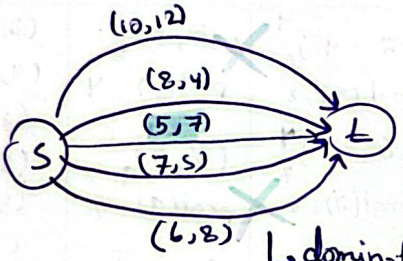
→ In some problems, we want to find path that balance multiple objectives.

Ex:- path with minimum distance, that doesn't exceed a time limit. (distance, time)
↳ (multi-objective)
(multi-label)

(d_k, t_k) (k) ↳ label num ; ↳ node number

↳ d_k: tot. distance required to reach node (K) from (S)

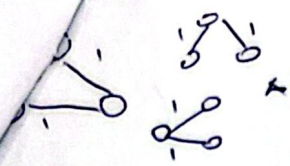
↳ t_k: tot time required to reach node (K) from (S)



(5,7) dominat both of (6,8) & (10,12)

↳ dominated by (5,7)

for (5,7) & (7,5) no domination --



Same cost but different trees.

We have unique cost and not unique tree..

Minimum spanning tree models

Trees

A tree T in G is a collection of arcs such that all n nodes are connected, and no *cycles* exist. (All nodes connected implies the existence of a path between all pairs of nodes on T.) An equivalent definition of a tree is a set of $n-1$ arcs such that all nodes are connected. An important property of a tree is that it defines, for each pair of nodes, a unique (not necessarily directed) path between them.

Tree examples: Directed network, undirected network.

Minimum spanning tree

A minimum spanning tree T^* in a network where each arc (i,j) has a cost c_{ij} is a tree such that the total cost (sum) of all the arcs in the tree is minimized:

where the sum of arc's cost is minimum

$$T^* = \text{argmin}_{T \in G}\{\sum_{(i,j) \in T} c_{ij}\}$$

easy to solve optimization problem

Question: Is the minimum spanning tree the same as the shortest path tree?
Answer: No. In the shortest path tree, the length of the path from some origin node to each other node is minimized. In the minimum spanning tree, the *total* cost of all the arcs in the tree is minimized. This is not the same thing.

2 - efficient algorithms

ass:- Undirected Complete graph

Kruskal's algorithm

Initialization: $TREE = \emptyset$

Iterations:

1. Sort the arcs in order of non-decreasing c_{ij} into the $LIST$
2. while $|TREE| < n-1$:
   (a) Remove (i,j) from the top of the $LIST$
   (b) Add (i,j) to $TREE$ if so doing does not create a cycle

At the end, $T^* \leftarrow TREE$.

Scanned with CamScanner

Prim's algorithm

Cycles يعني مستقل

Cuts

To explain the next algorithm, it necessary to understand the concept of a *cut* in a network.

Suppose the nodes in N are partitioned into two independent subsets, S and T. That is, $S \cup T = N$ and $S \cap T = \emptyset$. The cut set (S, T) is given by all arcs (i, j) such that $i \in S$ and $j \in T$ or $i \in T$ and $j \in S$.

Prim's algorithm

Initialization:

$TREE = \emptyset$

$S = \{1\}, T = N \setminus S$

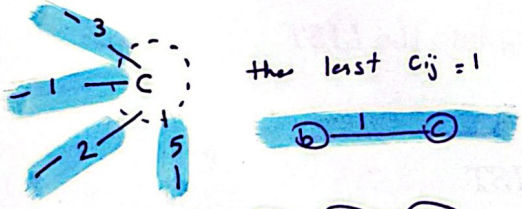
(In English, the set S contains node 1, while the set T contains all nodes except node 1)

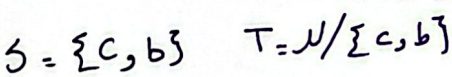
Iterations:

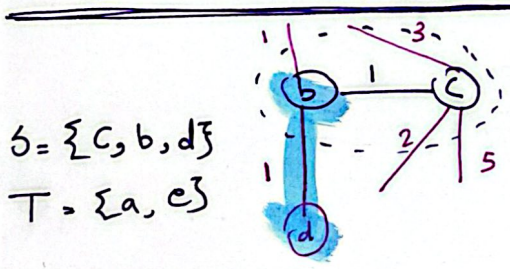
while $|TREE| < n - 1$:

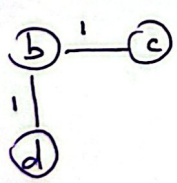
1. Find the arc (i, j) in (S, T) with minimum cost c_{ij}
2. Add (i, j) to $TREE$
3. Remove j from T and add j to S

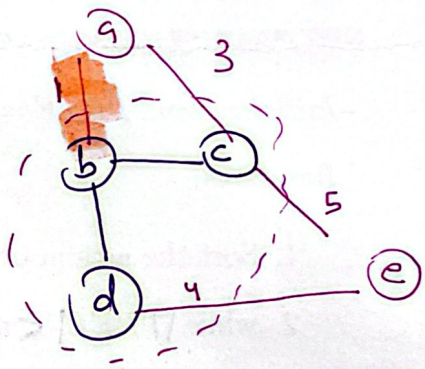
At the end, $T^* \leftarrow TREE$.

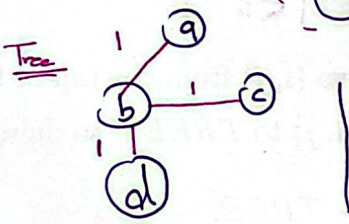


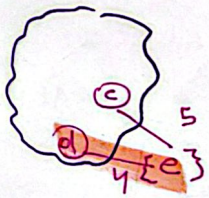


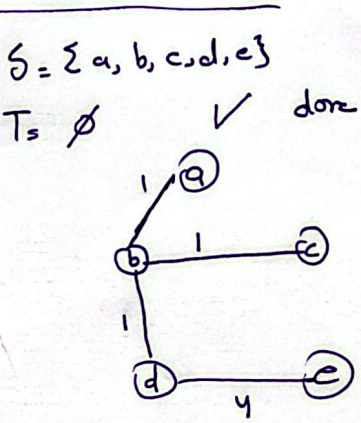












C(T) = 7 $

A simple lower bound for the TSP

to decompose the $C(T^*)$ in 2 terms

Often, we generate lower bounds using a decomposition strategy. For example, suppose a feasible solution to a problem can be decomposed into two components. If these two components are optimized separately, the resulting solution may not be feasible for the original problem, but the resulting cost is a lower bound on the optimal cost of the original problem.

Consider any feasible solution to the TSP, that is a tour visiting all nodes in G. Suppose $TOUR$ is a set containing all of the arcs used in the tour; clearly $|TOUR| = n$. If one arc (i, j) from the tour is removed, the remaining structure is a $TREE$ on G; all nodes are connected and the $TREE$ contains $n - 1$ arcs. In set terminology, we could write this decomposition as:

arc set = n

$$TOUR = TREE + \{(i, j)\}$$

The cost could be written as follows:

$$C(TOUR) = C(TREE) + c_{ij}$$

Now let's minimize the decomposition. Clearly, the minimum cost tree in a network is given by the MST. And, in general, (i, j) could be any arc in the network, but more specifically is some arc that is not in the TREE. Therefore, a lower bound on the optimal TSP cost is the following:

$$C(TSP^*) \geq C(MST^*) + c^*_{ij}$$

where c^*_{ij} is minimum cost arc not in MST^*.

↳ any arc (i,j) with the cost $C^*(i,j)$ → least cost that doesn't belong to the MST..

How to find LB :-

↳ Simple LB : by finding MST and then take the arc with the least cost that doesn't belong to the MST.. $LB = C(MST) + C_{ij}^*$

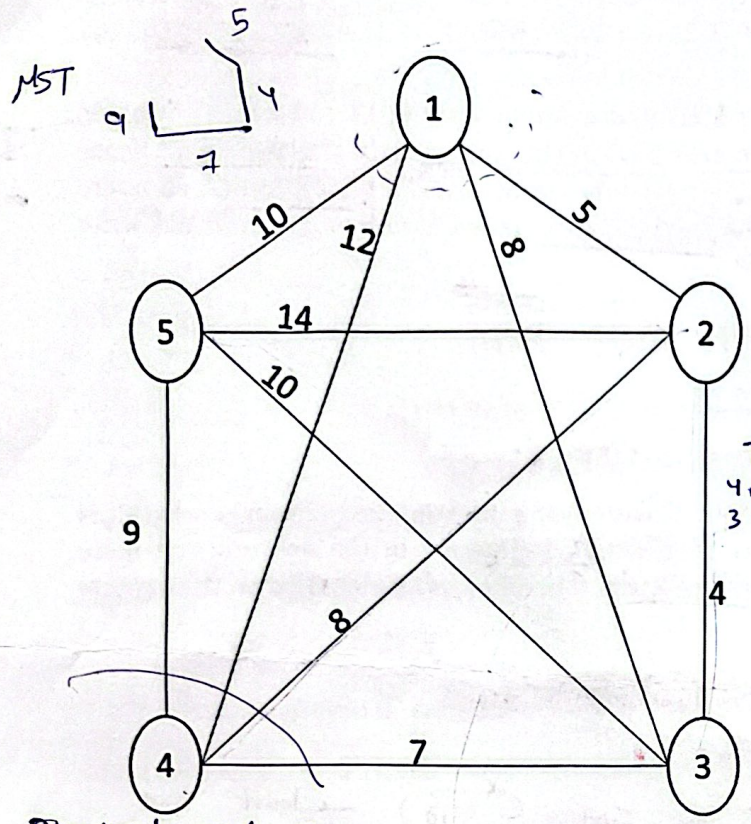
↳ by omitting one node, MST for the nodes without the omitted one (MST → (n-1) nodes → (n-2) arcs) then, we add the omitted node and its 2 least cost adjacent arcs..

i.e. $TREE_{(i)}$ where i is the omitted node..

① step 1:- LB

Extra page:

Example 1: Consider the following symmetric, undirected network G where the numbers next to each arc are the arc costs. Suppose that your goal is to find a good traveling salesman tour that visits all of the nodes of G.

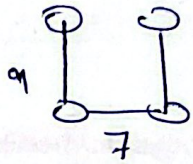


MST

LB_1 = MST + C_{ij}
= 25 + 8 = 33

LB_2 :- omitted node ①

Tree
4 nodes
3 arcs

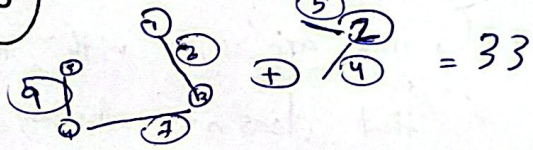


= 20



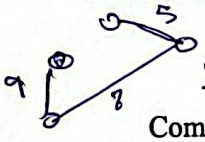
= 13 = 33

Omitted node (2)



= 33

Omitted node 3



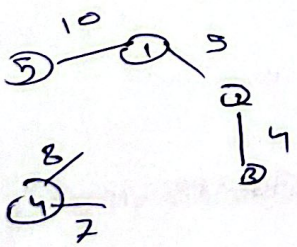
22 + 11 = 33

Complete the following table that shows the lower bounds using several techniques:

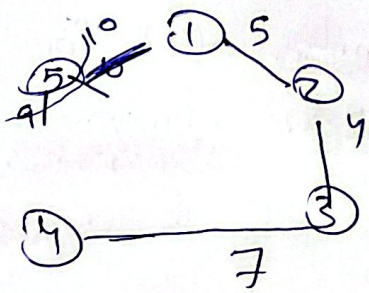
technique	Lower bound
MST+arc	25 + 8 = 33
$1-TREE_1$	20 + 13 = 33
$1-TREE_2$	24 + 9 = 33
$1-TREE_3$	22 + 11 = 33
$1-TREE_4$	19 + 15 = 34
$1-TREE_5$	19 + 16 = 35

the closest value to the optimal

Omitted node 4



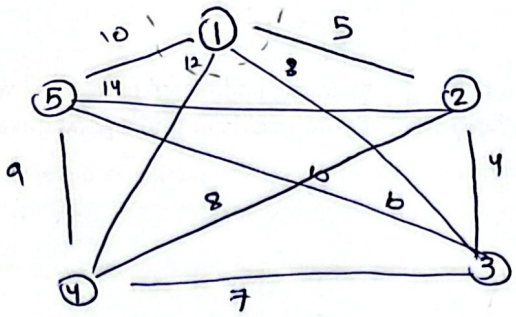
Omitted node (5)



Tour

rs next to
that

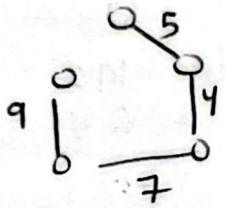
Extra page:



Find the LB

1) Simple LB :- $LB = MST + (i,j)^*$

MST



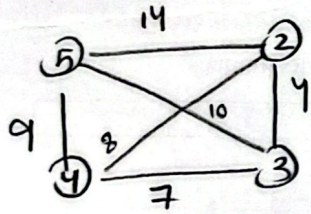
$C(MST) = 25$

$\downarrow$

$C(i,j)^* = 8$ ⟶ $LB = 25 + 7 = \underline{33}$

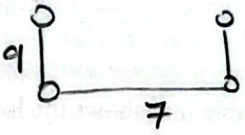
---

2) 1 - TREE,

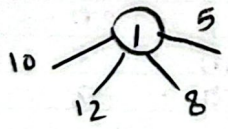


4 nodes & 3 arcs

→ MST :-

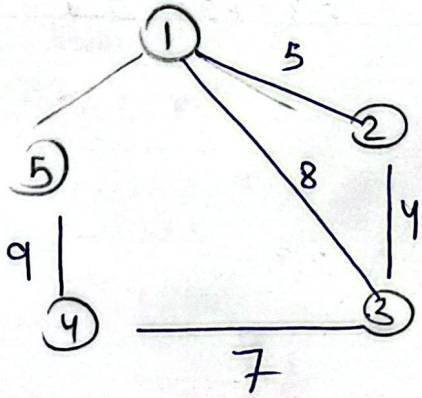


$= 20$



take 2 arcs with the least costs

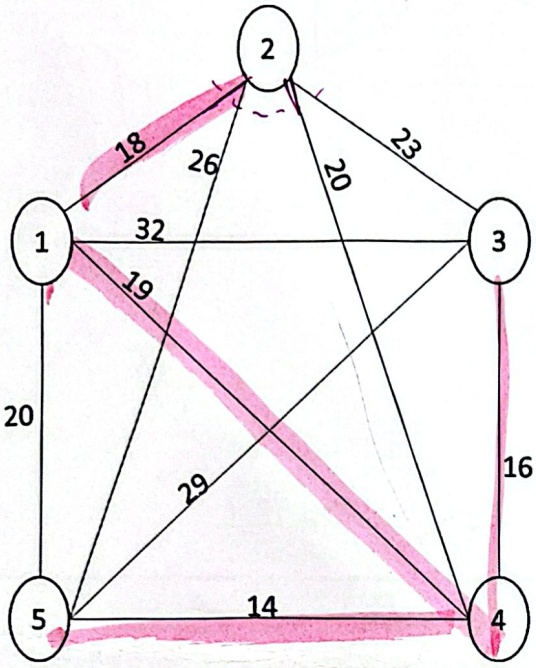
5 & 8

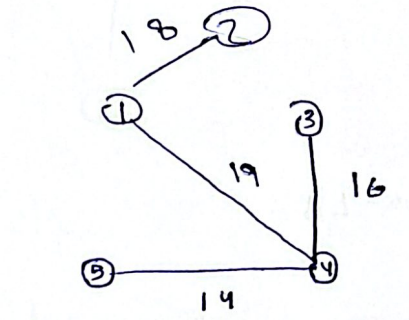


$20 + 5 + 8 = \underline{33}$

Scanned with CamScanner

Example 2: Consider the following symmetric, undirected network G where the numbers next to each arc are the arc costs. Suppose that your goal is to find a good traveling salesman tour that visits all of the nodes of G.



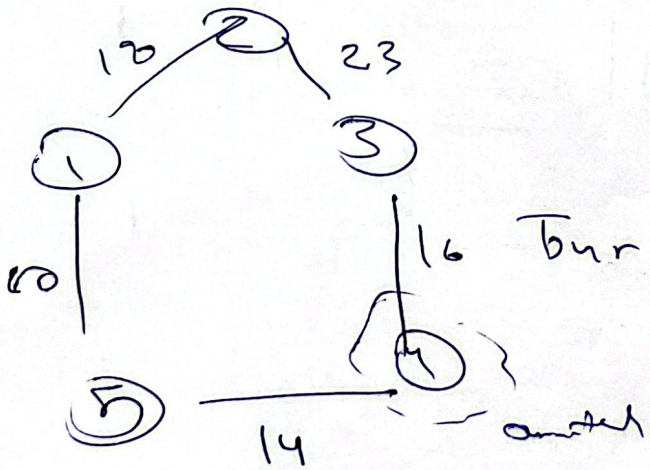


14 + 16 + 19 + 17

MST = 67

Complete the following table that shows the lower bounds using several techniques:

technique	Lower bound
MST+arc	67 + 20 = 87
$1-TREE_1$	50 + 37 = 87
$1-TREE_2$	49 + 38 = 87
$1-TREE_3$	→ 30 + 37 + 23 = 90
$1-TREE_4$	30 + 23 + 38 = 91
$1-TREE_5$	53 + 34 = 87



Twice around MST Heuristic

The MST 2-approximation heuristic will generate a TSP tour with a cost no greater than double the optimal cost for networks satisfying the triangle inequality. This is not good, but at least it's a guarantee!

An α-approximation heuristic: MST 2-approximation

Steps:

1. Generate a minimum spanning tree MST^* of the nodes N.
2. Let $LIST$ contain two copies of each arc in the minimum spanning tree, MST^*.
3. Generate a $WALK$ of nodes of G using each arc in $LIST$ exactly once; $\{v_1, v_2, ..., v_{2(n-1)}, v_1\}$. Note that v_i are not unique.
4. Create a heuristic tour T^H by following the nodes in the order specified by $WALK$, but skipping repeat visits to any node (the "taking shortcuts" process)

* In this method, recognize that $WALK$ does not fit with our definition of a tour, since it may contain nodes that are visited more than once. So, we simply eliminate repeat visits in our WALK to identify T^H, sometimes referred to as an embedded tour.

TSP with performance guarantee
↳ an α-optimal Heuristics (AKA also known as Approximation Heuristic)
it'll never return a sol whose cost more than α-times turn optimal solution.
We'll focus 2 Heuristic with performance guarantee
↳ Twice around MST → $\alpha = 2$
↳ Christofied Heurstic → $\alpha = 1.5$

* when you skip repeated nodes, because of the triangle inequality → skipping will not increase the total cost, May actualy decrease/reduce it.

Christofides' لا يعني انه

$T_H \leq 1.5 T^*$ كل الاخطل هي بتحقيق

مستحيل تخطيها اي اكثر من $1.5 T^*$ يعني انه ممكن

اطلع اني اسن

Christofides' Heuristic $\alpha = 1.5$ الى الآن ما في اقل من

using other heuristic

Christofides' heuristic has the best known worst-case performance bound for the traveling salesman problem on complete networks satisfying the triangle-inequality. Similar to the Twice-Around MST heuristic, Christofides' heuristic utilizes the concept of minimum spanning trees. (1) In addition, the heuristic relies (2) on the idea of minimum cost perfect matchings (MCPM). Consider the following definitions and facts:

(Christofides Worst-case Performance Bound) Christofides heuristic is 1.5-*optimal for every complete network G satisfying the Δ-inequality, that is:*

$$C(T^C) \leq \frac{3}{2} C(TSP^*)$$

where TSP^* is an optimal traveling salesman tour on G.

Christofides' Heuristic

Steps:

1. Given an undirected, complete network G satisfying the Δ-inequality, find a minimum spanning tree MST^* on G.
2. Let S be the nodes of G with odd degree with respect to MST^*.
3. Find a perfect node matching W^* of the nodes in S with minimum total cost. (This problem can be solved in $O(n^3)$ time by the method of Lawler).
4. Generate a tour of the nodes T^C using arcs in the set $MST^* \cup W^*$ exactly once. (T^C can be improved by introducing shortcuts to avoid revisiting nodes).

↳ skipping

for any tree the numbers of nodes with odd degree is even --

$\sum$ degree = 2A → even number

$\sum$ degree = $\sum$ degree (nodes with even degree) ⊕ $\sum$ degree (nodes with odd degree)

↳ even

↳ sum for even numbers → even

↳ sum of odd number can't be even unless their number is even

k terms

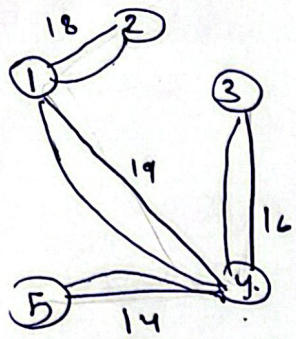
odd + odd + -- odd → k: even → sum even

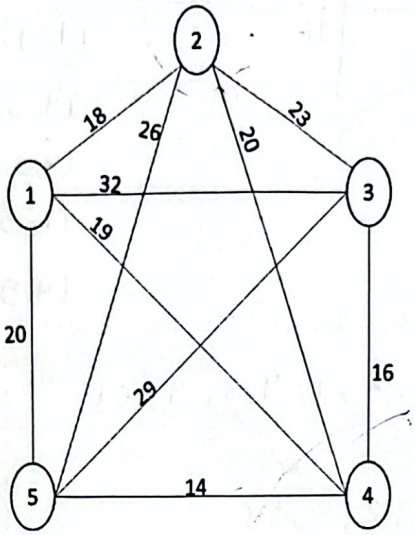
→ k: odd → odd

Scanned with CamScanner

① MST = 14 + 16 + 19 + 18 = 67

Consider the following undirected network G where the numbers next to each arc are the arc costs. Suppose that your goal is to find a good traveling salesman tour that visits all of the nodes of G.





Copy each arc (LIST)

(1,2) (1,4) (3,4) (4,5)
(1,2) (1,4) (3,4) (4,5)

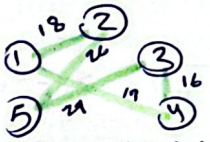
Walk from (2 to 2) (any node)

W = {2, 1, 4, 3, 4, 5, 4, 1, 2}

we walk along each arcs.

T^H = {2, 1, 4, 3, 5, 2}

C_{T_H} = 108

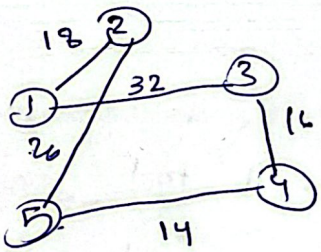


a) Determine the TSP heuristic tour yielded by applying the twice around MST heuristic.
b) Determine the TSP heuristic tour yielded by applying Christofides heuristic.

Walk 4–4 = {4, 3, 4, 1, 2, 1, 4, 5, 4}

T^H = {4, 3, 1, 2, 5, 4}

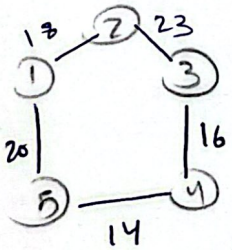
$C(T^H)$ = 106



Walk from 3 to 3

W = {3, 4, 5, 4, 1, 2, 1, 4, 3}

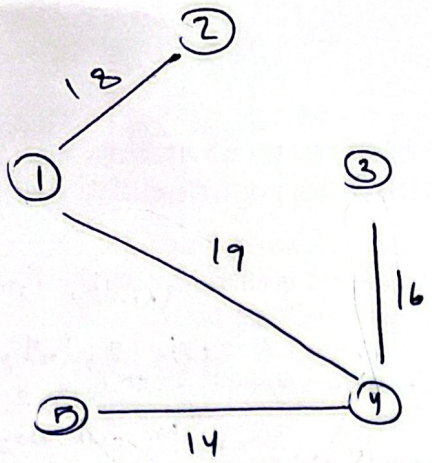
T^H = {3, 4, 5, 1, 2, 3}



C^H = 91

LB (1-Tree 4) = 91

LB = C^H → = optimal



LIST

(1,2) ✓

(1,2) ✓

(1,4) ✓

(1,4) ✓

(4,3) ✓

(4,3) ✓

(4,5) ✓

(4,5) ✓

Walk (5 to 5)

$W = \{5, 4, 1, 2, 1, 4, 3, 4, 5\}$

$T_H = \{5, 4, 1, 2, 3, 5\}$

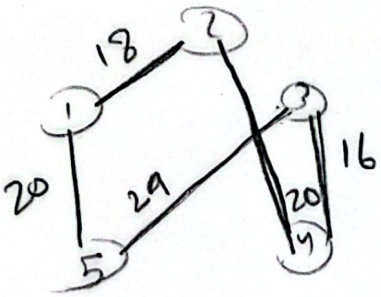
$C^H = 14 + 19 + 18 + 23 + 29 = 103$

---

from 1 to 1

$W = \{1, 2, 4, 4, 3, 4, 5, 4, 1\}$

$T_H = \{1, 2, 4, 3, 5, 1\}$



$C_H = 103$

Nearest insertion heuristic

Initialization:

Given an undirected network $G = (N, A)$, with costs c_{ij} associated with each $(i, j) \in A$, where c_{ij} satisfies the Δ-inequality.

Find the arc (i, j) with minimum cost in A. Let $T = \{i, j, i\}$

Steps:

while $|T| < n + 1$:

1. Find the node $j \notin T$ that is *closest* to a node currently in T. That is, find $j \notin T, i \in T$ such that c_{ij} is minimized.
2. Find the *insertion location* for node j into the existing tour. To do so, find the arc (ℓ, k) in the current tour with the minimum insertion cost: $c_{\ell j} + c_{jk} - c_{\ell k}$. Insert j between ℓ and k in T.

end;

Farthest insertion heuristic

Initialization:

Given an undirected network $G = (N, A)$, with costs c_{ij} associated with each $(i, j) \in A$, where c_{ij} satisfies the Δ-inequality.

Find the arc (i, j) with maximum cost in A. Let $T = \{i, j, i\}$

Steps:

while $|T| < n + 1$:

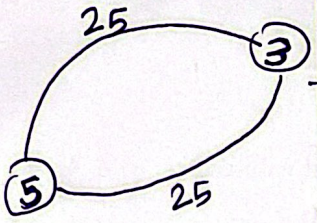
1. Let $C_j(T)$ be $\min_{i \in T} c_{ij}$ for all nodes $j \notin T$. The node j to be inserted into the tour is one with maximum $C_j(T)$.
2. Find the *insertion location* for node j into the existing tour. To do so, find the arc (ℓ, k) in the current tour with the minimum insertion cost: $c_{\ell j} + c_{jk} - c_{\ell k}$. Insert j between ℓ and k in T.

end;

In English, Step 1 finds the node j not on the tour for which the minimum distance to a vertex on the tour is maximum, and adds that node to the current tour.

Example: Consi
the arc costs
nodes

Initialization



$T = \{3, 5, 3\}$

$C(T) = 25 + 25 = 50$

$T = \{3, 2, 5, 4, 3\}$

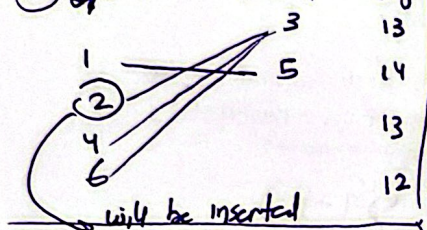
$C = 59 + 6 = 65$

i∉T	j∈T	C_{ij}	
1	2	12	maximum
6	2	5	③

→ to be inserted

3 _ 2 _ 5 _ 4 _ 3 → 4 expected insertion locations

① i∉T j∈T C_{ij}



1 3	13
1 5	14
2 3	13
2 5	12
4 3	
4 5	
6 3	
6 5	

will be inserted to the current tour

$(3,2) = C_{31} + C_{12} - C_{32}$

$(2,5) = C_{21} + C_{15} - C_{25}$ ✱ min = 5

$(5,4) = C_{51} + C_{14} - C_{54}$

$(4,3) = C_{41} + C_{13} - C_{43}$

$C(T) = 65 + 5 = 70$

$T = \{3, 1, 5, 3\}$ (2)

cost $= C_{32} + C_{25} - C_{35} = 9$

cost $= 50 + 9 = 59$

④ $T = \{3, 2, 1, 5, 4, 3\}$

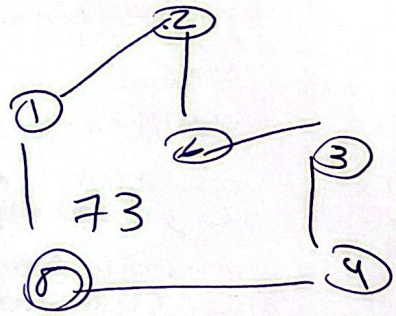
$C_{32} = 3$ ✱ pair wise

$C_{41} = 3$ 2 ✓1

$C_{15} = 11$

$C_{54} = 11$ cost = 73

$C_{43} = 14$

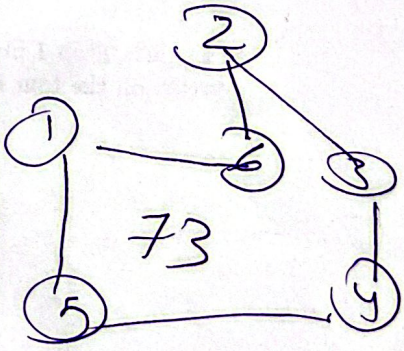


j∉T	i∈T	C_{ij}
1	2	12
4	3	13
6	2	5

② node ④ will be inserted

$[3, 6, 2, 1, 5, 4, 3]$ —①

$[3, 2, 6, 1, 5, 4, 3]$ —①



Iter → $\{3, 2, 5, 3\}$ (4)

btw $(3,2) = C_{34} + C_{42} - C_{32} = 17$

btw $(2,5) = C_{24} + C_{45} - C_{25} = 16$

btw $(5,3) = C_{54} + C_{43} - C_{53} = 6$ ✱

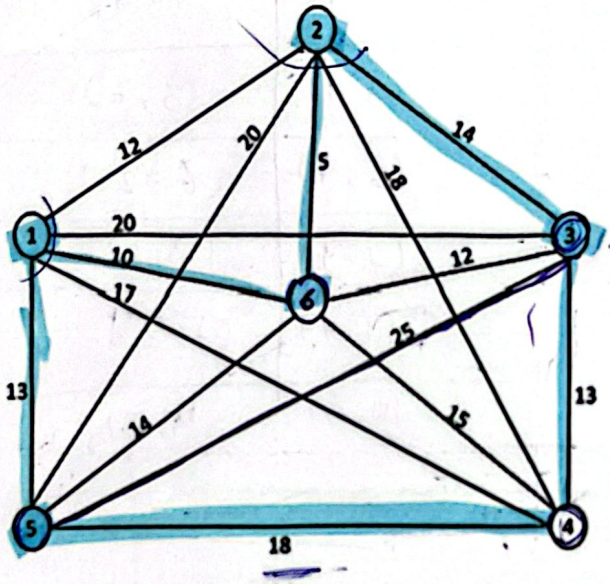
Min insertion cost

1 → 1
2 → 3
3 → 4
17 → (18) ✓

Expected # of insertion location

37 nodes
↳ (37-2) = 35
+ one node each iteration

Example: Consider the following undirected network G where the numbers next to each arc are the arc costs. Suppose that your goal is to find a good traveling salesman tour that visits all of the nodes of G.



6 nodes

{ , 5 , }

5 nodes

a) Determine the TSP heuristic tour yielded by applying the Nearest insertion heuristic.
b) Determine the TSP heuristic tour yielded by applying the Farthest insertion heuristic.

= = = = = using the nearest neighbor

$\sum \text{degree} = (2n-1)$

$= 2n - 2$

Nearest neighbor Heuristic

↳ greedy heuristic

↳ usually it yields to a bad-quality solution

However, it's used as an initial step before improving heuristic

→ first, we start from arbitrary node or least cost arc ..

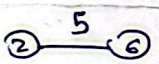
→ then, determine the nearest neighbor nodes to each end ..

↳ then visit the node that has the minimum cost ..

↳ update end, search for nearest neighbor

continue until all nodes are visited

↳ end : node of degree one.

1) 

end: 2, 6

End	node to be added
2, 6	add arc (6, 1)
2, 1	add arc (1, 5)
2, 5	add arc (2, 3)
3, 5	add arc (3, 4)

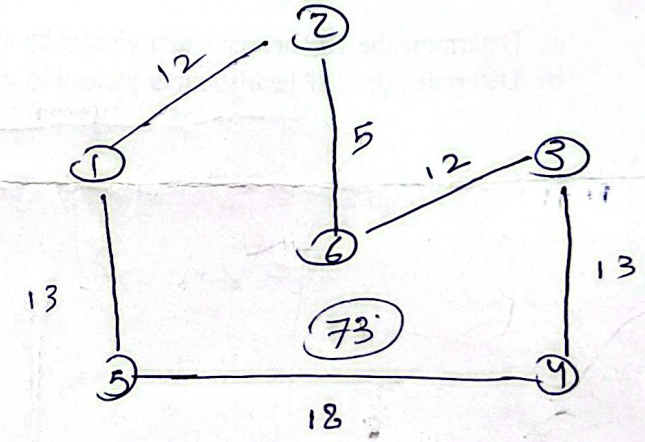
add arc (5, 4) for ends to have degree of 2 and form a tour ...

$C(T) = 5+10+13+14+13+18 = 73$

إذا الـ least cost arc بالبلش بالـ Node اللي موجود

Start with/from arbitrary node

End	arc to be added
4	(3, 4)
(3, 4)	(3, 6)
(6, 4)	(6, 2)
(2, 4)	(2, 1)
(1, 4)	(1, 5)
(4, 5)	(4, 5)



the farthest in practice is better than the nearest ..

# of iterations = n - 2 = 4 iterations

Insertion cost = Cost of arcs added - Cost of arc removed

Initialization: arc with minimum cost (2, 6)

$T = \{2, 6, 2\}$

$C^*(T) = 5 + 5 = 10$



Iterations:- ① which node to be inserted ② where to be inserted

Iteration 1 :- Nearst closest in T

Iteration to be inserted

$j \notin T$	$i \in T$	C_{ij}
* 1	6	10
3	6	12
4	6	15
5	6	14

at Iteration "1" → one possible Insertion location

$C.Insertion = C_{21} + C_{16} - C_{26} = 17$

$C(T)^* = 10 + 17 = 27$

$T = \{2, 1, 6, 2\}$ $C(T) = 27$

Iteration 2:-

$j \notin T$	$i \in T$	C_{ij}
3	6	12
4	6	15
5	1	13

Iteration 2 → 3 possible Insertion locations

Insertion cost « Minimum »:-

$btw(1,2) = C_{13} + C_{32} - C_{12} = 22$

$btw(1,6) = C_{13} + C_{36} - C_{16} = 22$

$btw(6,2) = C_{63} + C_{32} - C_{62} = 21$ *

$T = \{2, 1, 6, 3, 2\}$

$C(T)^* = 27 + 21 = 48$

Iteration 3:- 4 Insertion locations are possible

$j \notin T$	$i \in T$	C_{ij}
4	3	13
5	1	13

→ Choose arbitrary

In cost: الكلفة هي نفسها & In tours ..

$btw(1,2) = 23$

$btw(1,6) = 22$

$btw(6,3) = 16$ → minimum insertion cost ...

$btw(3,2) = 17$

$T = \{2, 1, 6, 4, 3, 2\}$

$C(T)^* = 48 + 16 = 64$

Iteration 4 → last Iteration → 5 Possible Insertion location

node 5 to be inserted

$btw(2,1) = 21$

$btw(1,6) = 17$ → take them both as a solution only if it's the last iteration

$btw(6,4) = 17$

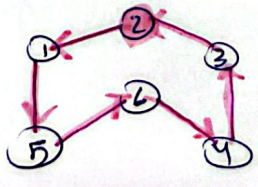
$btw(4,3) = 30$

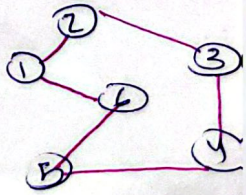
$btw(3,2) = 31$

$T_1 = \{2, 1, 5, 6, 4, 3, 2\}$

$T_2 = \{2, 1, 6, 5, 4, 3, 2\}$

$C(T) = 64 + 17 = 81$





Scanned with CamScanner

Farthest Insertion

* Initialization : arc with the maximum cost

(5,3) , T = [5, 3, 5] , C(T)* = 25+25 = 50

Iteration (1) :- ① which node to be insarted

② Where to be insarted

to be Insated j ∉ T	Nearest i ∈ T	Maximum Cij
1	5	13
2	3	14 * Maximum
4	3	13
6	3	12

btw (5,3) = $C_{52} + C_{23} - C_{53}$ = 9

T = [5, 2, 3, 5]

C(T)* = 59

Iteration 2 → 3 Insention possibilities

j ∉ T	Nearest i ∈ T	Maximum Cij
1	2	12
4	3	13 *
6	2	5

btw (5,2) = 16

btw (2,3) = 17

btw (3,5) = 6 ← Min Insention cost

T = [5, 2, 3, 4, 5]

C(T)* = 59 + 6 = 65

Iteration 3 4 (IL)

j ∉ T	i ∈ T	Cij
1	2	12 *
6	2	5

btw (5,2) = 5

btw (2,3) = 18

btw (3,4) = 24

btw (4,5) = 12

T = [5, 1, 2, 3, 4, 5]

C(T*) = 65 + 5 = 70

Iteration 4 (+1 = 5 PIL)

node 6 to be added

Insention cost :-

btw (5,1) = 11

btw (1,2) = 3 — we must take them both. In solution

btw (2,3) = 3

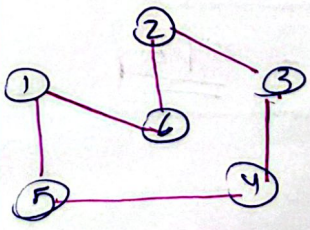
btw (3,4) = 14

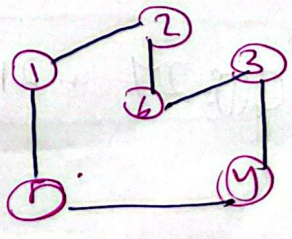
btw (4,5) = 11

T_1 = [5, 1, 6, 2, 3, 4, 5]

T_2 = [5, 1, 2, 6, 3, 4, 5]

C_4 = 70 + 3 = 73





Bin Pa

Scanned with CamScanner

Bin Packing

The bin-packing problem is another important problem in operations research with applications in logistics and transportation system operations.

Problem definition: Suppose we have a supply of bins, and each bin has a capacity to hold Q units. Furthermore, suppose we have a number of items that need to be packed into bins. Each item i has a size $q_i \in (0, Q]$ which represents the portion of a bin's capacity that this item requires. Items cannot be "split" between multiple bins. Suppose for explanatory purposes the item sizes have been arranged into a list, $L = \{q_1, q_2, q_3, \ldots, q_n\}$.

Problem BPP: Assign each item to a bin such that:

1. The total weight of all items in a bin is no greater than Q (no bin violates capacity)
2. The smallest number of bins is used

Integer Programming Formulation

Consider now the following mathematical programming formulation for the bin-packing problem. Here, we will assume that n bins are available for packing, since in the worst-case, each item would require its own bin.

Decision variables

$$y_j = \begin{cases} 1, & \text{if bin } j \text{ is used} \\ 0, & \text{otherwise} \end{cases} \quad \forall\, j = 1 \ldots m$$

$$x_{ij} = \begin{cases} 1, & \text{if item } i \text{ is assigned to bin } j \\ 0, & \text{otherwise} \end{cases} \quad \forall\, i = 1 \ldots n,\ j = 1 \ldots n$$

(BPP)

$$\min \sum_{j=1}^{m} y_j \qquad \text{(OF)}$$

subject to:

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall\, i = 1 \ldots n \qquad \text{(C1)}$$

$$\sum_{i=1}^{n} q_i x_{ij} \le Q \quad \forall\, j = 1 \ldots m \qquad \text{(C2)}$$

$$x_{ij} \le y_j \quad \forall\, i = 1 \ldots n,\ j = 1 \ldots m \qquad \text{(C3)}$$

$$0 \le x_{ij} \le 1 \quad \forall\, (i, j) \qquad \text{(C4)}$$

$$x_{ij} \text{ integer} \quad \forall\, (i, j) \qquad \text{(C5)}$$

y_j	x_{ij}	
1	>	0
0	=	0
1	=	1

$y_j \geq x_{ij}$

j is opened & not assigned to j

j not opened → i will not be assigned to j

y_j opened & i is assigned to j

The minimization nature of this formulation attempts to set $y_j = 1$ for as few bins as possible, and the objective function counts the number of bins opened. Constraints (C1) ensure that each item is assigned to some bin, and (C2) require that no bin exceeds its capacity Q. Constraints (C3) require that closed bins $y_j = 0$ can have no items assigned to them.

$x_{ij} \forall i, \forall j = 0, y_j = 0$

Like the traveling salesman problem, no efficient algorithm has been developed for the bin-packing problem; it is in the problem class NP-hard. If one wishes to solve the problem optimally, it is necessary to enumerate all possible ways of assigning items to bins, and this process is inefficient. → easy to understand → hard to solve → we need heuristic

Applications in Logistics

The primary application of bin-packing in logistics is in the loading of freight. Suppose that you have n pieces of freight to load with various sizes, and you wish to use the fewest containers as possible to transport the freight. This becomes a bin-packing problem. This situation also arises when designing vehicle routing tours. One potential problem in vehicle routing is to determine the minimum number of transportation vehicles required to serve a set of customer demands, under the assumption that customer demands cannot be split between vehicles.

For example, suppose that you operate a fleet of vehicles each with capacity of 20 tons. Further, suppose that you need to serve 13 customers on a given day, and the following is their demands in tons: $L = \{16, 10, 14, 12, 4, 8, 3, 12, 7, 14, 9, 6, 3\}$. How many trucks are required? Which trucks should serve which demands?

Online heuristics for BPP

اقرب الى الواقع

First, a general definition. An online heuristic is one in which input data are received over time, and irrevocable decisions are made based on known data within some time limit. Online heuristics have no foresight; they cannot anticipate future data that has not yet "arrived." Often, real-world problems require online heuristics. For example, if you are operating a truck fleet and planning which trucks will serve which demands, the demands from customers may arrive dynamically and you may not have anticipatory knowledge of what the next customer demand will be.

الكل

Short local visibility

In the following online heuristics for BPP, let's assume that customer lot sizes become known one at a time. Upon the arrival of a lot, an irrevocable assignment of the lot to a bin is made. Such heuristics have practical significance in the loading of transport containers, since changing a decision of where to stow a particular piece of cargo would require unloading it (and potentially other lots) and reloading it into a different container.

Suppose that customer lots arrive one at a time, in the order specified by L. Let's develop some heuristics.

Next-Fit Heuristic

Description: This heuristic places the next item in the currently open bin. If it does not fit, the bin is closed and a new bin is opened.

off-line heuristic

→ data about all items are available before any decision need to be finalized

Initialization:
Given a list of item weights $L = \{w_1, w_2, ..., w_n\}$.
Place item 1 in bin 1 and remove from L. Let $i = 1$, $j = 2$.

Iterations:

1. If item j fits in bin i, place j in i. If not, open a new bin $i+1$ and place j in bin $i+1$. Let $i = i+1$.
2. Remove item j from L. Let $j = j+1$.
3. While items remain in L, repeat from Step 1.

The next-fit heuristic has a special advantage over other online heuristics in this case, since it allows immediate dispatch. In the heuristic, whenever a container is closed it will not receive any future items and therefore can be immediately dispatched. This might mean a truck can leave a depot, or a container can be loaded onto a ship or train, or something similar.

First-Fit Heuristic

Description: This heuristic keeps all unfull bins open. It places the next item in the lowest-numbered bin in which the item fits. If it does not fit in any bin, a new bin is opened.

Initialization:
Given a list of item weights $L = \{w_1, w_2, ..., w_n\}$.
Place item 1 in bin 1 and remove from L. Let $j = 2$, $m = 1$.

Iterations:

1. Find the lowest numbered bin i in which item j fits, and place j in i. If j does not fit in any bin, open a new bin and number it $m+1$ and let $m = m+1$.
2. Remove item j from L. Let $j = j+1$.
3. While items remain in L, repeat from Step 1.

Best-Fit Heuristic

Description: This heuristic attempts to create the fullest bin possible each time an item is assigned. Again, all unfull bins are kept open. It places the next item j in the bin whose current contents are largest, but do not exceed $1 - w_j$ (so the item fits). If it does not fit in any bin, a new bin is opened.

Initialization:

First fit	Best fit
* All bins "unfull" are opened	* All unfull Bins are opened
* place i in the first fitted bin	place i in the Best "least RC" Bin

Given a list of item weights $L = \{w_1, w_2, ..., w_n\}$.
Place item 1 in bin 1 and remove from L. Let $j = 2$, $m = 1$.

Iterations:

1. Find the bin i whose contents are maximum but not greater than $1 - w_j$ (if S_i are the items in bin i, $\sum_{k \in S_i} w_k$ is the contents of bin i), and place j in i. If j does not fit in any bin, open a new bin and number it $m + 1$ and let $m = m + 1$.
2. Remove item j from L. Let $j = j + 1$.
3. While items remain in L, repeat from Step 1.

Performance guarantees
The next-fit heuristic is 2-optimal. It has been shown that the first-fit and best-fit heuristics are 1.7-optimal; that is, for any input item sizes L the number of bins that they will generate is no worse than 1.7 times the optimal number of bins.

Practical issues with first-fit and best-fit
Both of these online heuristics lack the dispatch advantage of the next-fit heuristic. Of course, if a container is completely filled at some stage of the processing it can be immediately dispatched. However, in the pure form of these heuristics, bins that are open but not yet full cannot be dispatched. Thus, to use these methods in practical settings it will be necessary to augment the methods with dispatch rules. For example, a container that has reached some minimum fill percentage ρ and/or has been open for some minimum time γ might be dispatched.

Offline heuristics for BPP

Offline heuristics assume that you have access to all the item sizes when you start the heuristics. There are three important offline heuristics for bin-packing that all work by first sorting the items by non-increasing size. They are called respectively *next-fit decreasing*, *first-fit decreasing*, and *best-fit decreasing*.

A good example string of numbers is 6,6,5,5,5,4,4,4,4,2,2,2,2,3,3,7,7,5,5,8,8,4,4,5.
Sorted becomes 8,8,7,7,6,6,5,5,5,5,5,5,4,4,4,4,4,4,3,3,2,2,2,2.

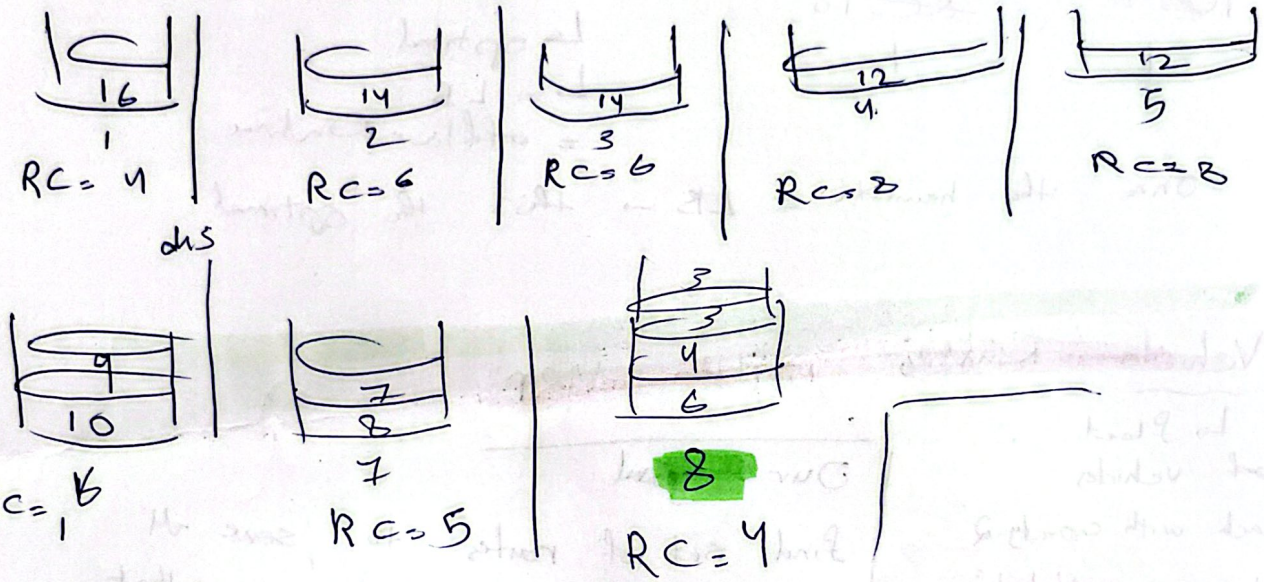
Performance guarantees
It has been shown that the first-fit and best-fit decreasing heuristics perform better (usually) than the online versions. They are both 1.5-optimal.

This means, that there is an advantage of having data already

هشترط انه using offline اطلع جواب احسن , بس صعيل الـ online
يكون احسن من الـ offline .. offline يعني ابدوه الاحوال الـ
بعطي نفس الـ online

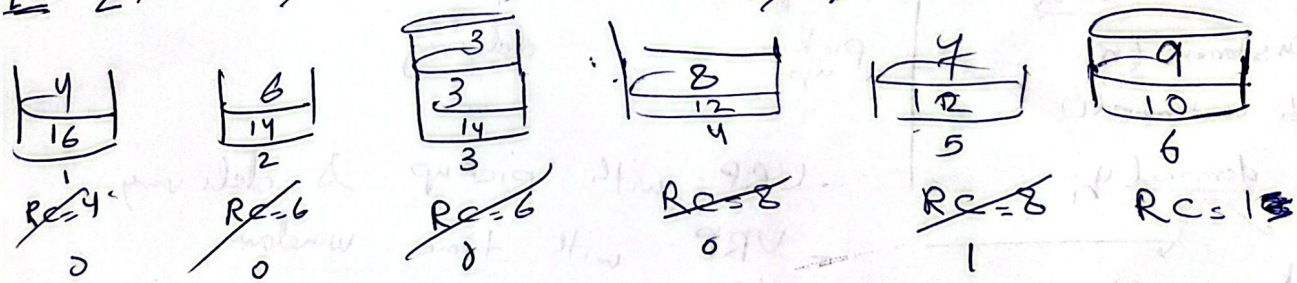
Next-fit . Decreasing heuristic

$L = \{16, 14, 14, 12, 12, 10, 9, 8, 7, 6, 4, 3, 3\}$



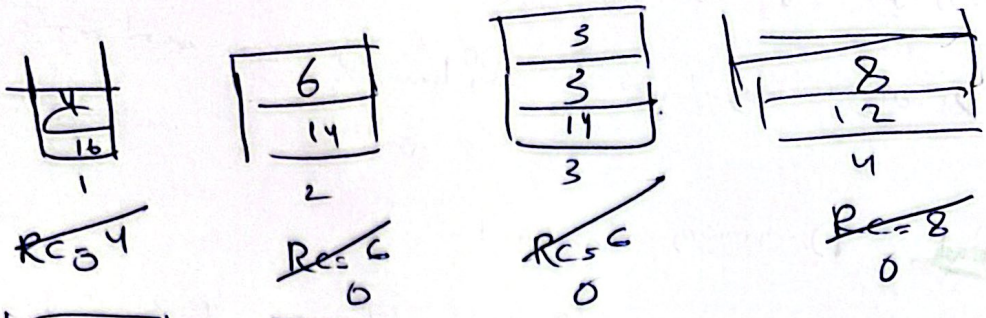
First-fit-heuristic

$L = \{16, 14, 14, 12, 12, 10, 9, 8, 7, 6, 4, 3, 3\}$ non increasing order

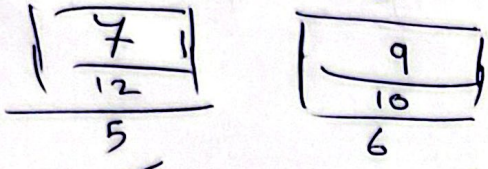


6 offline احسن بـ 4 online

Best-fit - heuristic "offline"



~~RC=4~~ ~~Re=6~~ 0 ~~RC=6~~ 0 ~~Re=8~~ 0



~~Re=8~~ 1 ~~RC=10~~ 1

6 ~~number~~ ~~bins~~ bins

↳ optimal

↳ = LB

= offline Online

One the heuristic = LB → this the optimal

Vehicle Routing Problem "VRP"

↳ fleet of vehicles each with capacity Q "homogenous Vehicles"

↳ All are station at the depot denoted by 0

↳ Customers (n) each customer (i) has demand q_i

Pick up or delivery

Our goal

find set of routes to serve all customers with minimum distance without violating Vehicle capacity.

↳ Demand ↳ pick up ↳ delivery

- VRP: with pickup & delivery
- VRP: with time window
- VRP: with simultanous pickup & "2 orders" delivery

↳ Truck & Trailer Routing problem

VRP → hard optimization Problem

Route-first Cluster-second Heuristics

Heuristics in this class begin first by creating an ordering of the nodes through some procedure, and then create feasible customer clusters to be served by a vehicle using the fixed ordering. After clusters are created, most of these heuristics simply serve customers following the initial order. However, others (like the Sweep heuristic) perform yet another step in which optimal or near-optimal TSP tours are created for each customer cluster and the depot. Technically, the Sweep Heuristic might be called an Order-first Cluster-second Route-third heuristic.

Generic TSP Partitioning Heuristic

Steps:

1. Let T be an optimal or near-optimal TSP tour that visits all of the customers and the depot. Suppose wlog $T = \{0, i_1, i_2, ..., i_n, 0\}$.
2. Define the node ordering $T_O = \{i_1, i_2, ..., i_n\}$.
3. Create customer clusters by running the next-fit bin packing heuristic with lot sizes q_i and vehicle capacity Q on the customer order defined by T_O.
4. Suppose the generated clusters are $\{i_1, i_2, ..., i_k\}$, $\{i_{k+1}, i_{k+2}, ..., i_\ell\}$, $\{i_{\ell+1}, i_{\ell+2}, ..., \}$. The final VRP tours are then given as $\{0, i_1, i_2, ..., i_k, 0\}$ etc. where the order of the customers in the VRP tours is identical to the order in the original TSP tour T.

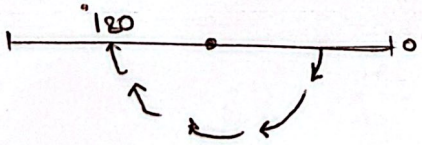
Cluster-first Route-second Heuristics

Often, in practical vehicle routing problems, the subproblem of determining which customers to group together in a cluster served by a single vehicle is relatively more important than routing considerations. Since effective clustering can lead to reduced fleet sizes through better packing, problems in which dispatch or fixed vehicle costs are high may include packing as a primary concern. However, it can be shown that by focusing on clustering before routing even in the traditional vehicle routing problem focused on distance-based costs can result in good heuristic solution methods.

Sweep Heuristic

In the sweep heuristic, nodes must have geographic positions. Without loss of generality, suppose that the depot is given cartesian coordinates $(0, 0)$, and suppose each customer i has polar coordinates (r_i, θ_i) relative to the depot. Here, suppose $0 \leq \theta_i < 2\pi$ is measured clockwise from the positive x-axis.

1. Choose a start ray r, with a polar angle θ.
2. For each customer, calculate Δ_i as the relative angle between r and the customer location, measured clockwise.
3. Sort the customers in order of non-decreasing Δ_i. Suppose the order created is $T_O = \{i_1, i_2, ..., i_n\}$.
4. Create customer clusters by running the next-fit bin packing heuristic with lot sizes q_i and vehicle capacity Q on the customer order defined by T_O.
5. Create final VRP tours by generating a separate TSP tour for each customer cluster and the depot.



Scanned with CamScanner

Example 1: Find

Suppose that we want to solve the vehicle routing problem with the following distance matrix and demands (units) at each customer location from 1 to 6. If each vehicle has capacity of 15 units.

	C_1	C_2	C_3	C_4	C_5	C_6
Demand	4	6	3	5	3	6

distance

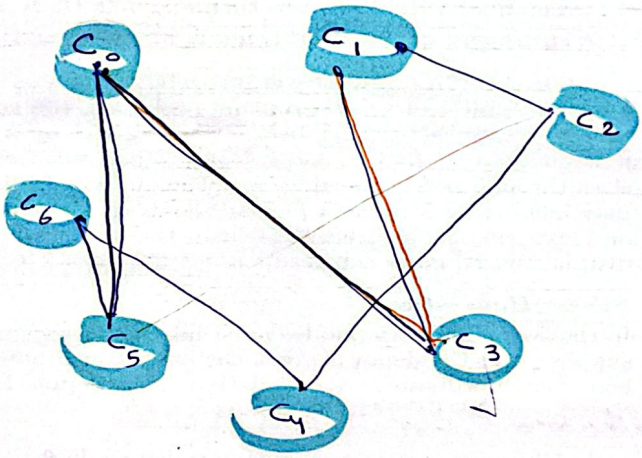
	C_0	C_1	C_2	C_3	C_4	C_5	C_6
C_0	-----	20	18	14	16	12	19
C_1	20	-----	22	18	30	26	28
C_2	18	22	-----	32	20	22	21
C_3	14	18	32	-----	20	22	21
C_4	16	30	20	20	-----	30	32
C_5	12	26	22	22	30	-----	26
C_6	19	28	21	21	32	26	-----

min cost arc

a) Use the TSP tour partitioning heuristic (a standard route first, cluster-second approach) to **generate a set of good vehicle routes** for this problem. Begin by using the Nearest Neighbor heuristic to generate a TSP tour. Determine all vehicles routes.

1) Nearest neighbor

Ends	arc to be added
	(C_0, C_5)
(C_0, C_5)	(C_0, C_3)
(C_3, C_5)	(C_1, C_3)
(C_1, C_5)	(C_1, C_2) → cycles (C_0, C_4)
(C_5, C_2)	(C_2, C_4)
(C_4, C_5)	(C_5, C_6)
(C_6, C_4)	(C_6, C_4)



$T = \{C_0, C_3, C_1, C_2, C_4, C_6, C_5, C_0\}$

depot first $T = \{C_3, C_1, C_2, C_4, C_6, C_5\}$

15 > 13

$C_1 = \{C_3, C_1, C_2\} \rightarrow R_1 = \{0, C_3, C_1, C_2, 0\}$

$C_2 = \{C_4, C_6, C_5\} \rightarrow R_2 = \{0, C_4, C_6, C_5, 0\}$

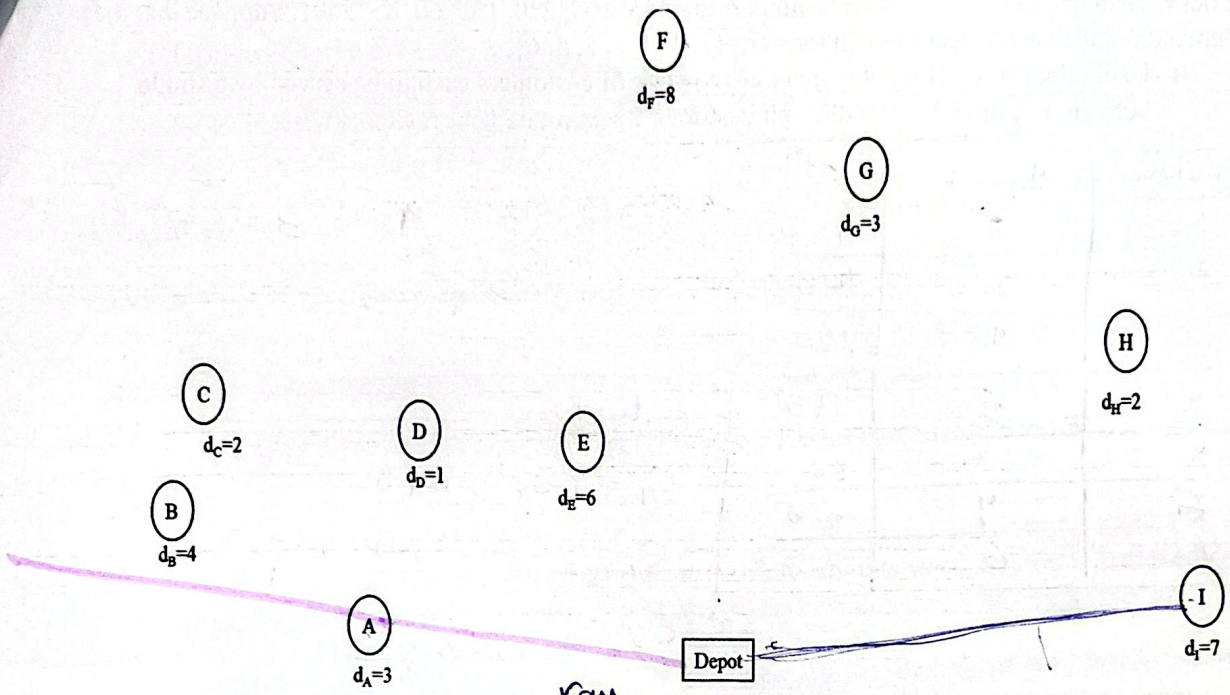
15 > 14

for cost sum the route's cost not the C_1, C_2 ..

matrix and

multiple customers in one truck

Example: Suppose you are the dispatcher for an LTL trucking firm, and you need to dispatch small delivery trucks for the local delivery of freight. Suppose you have many trucks available, each with the capacity to haul 10 tons of freight.



ray

{ A(3), B(4), C(2), D(1), E(6), F(8), G(3), H(2), I(7) }

a) Using the Sweep Heuristic, generate clusters of customers each to be served by a single vehicle. Begin with a ray through customer A and rotate your ray clockwise.

$C_1 = \{A, B, C, D\}$ $R_1 = \{0, A, B, C, D, 0\}$

$C_2 = \{E\}$ $R_2 = \{0, E, 0\}$

$C_3 = \{F\}$ $R_3 = \{0, F, 0\}$

$C_4 = \{G, H\}$ $R_4 = \{0, G, H, 0\}$

$C_5 = \{I\}$

$R_5 = \{0, I, 0\}$

b) Using the Sweep Heuristic, generate clusters of customers each to be served by a single vehicle. Begin with a ray through customer B and rotate your ray clockwise.

T = { B(4), C(2), D(1), E(6), F(8), G(3), H(2), I(7), A(3) }

$C_1 = \{B, C, D\}$ → $R_1 = \{0, B, C, D, 0\}$

$C_2 = \{E\}$ → $R_2 = \{0, E, 0\}$

$C_3 = \{F\}$ → $R_3 = \{0, G, H, 0\}$

$C_4 = \{G, H\}$ $R_3 = \{0, F, 0\}$

$C_5 = \{I, A\}$ $R_5 = \{0, I, A, 0\}$

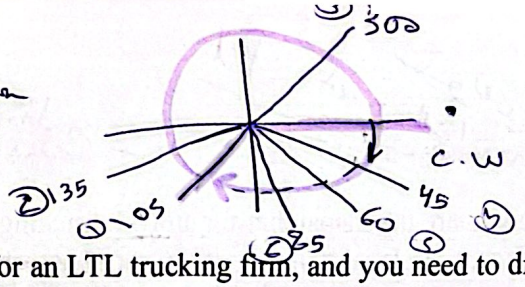
فيها ممكن انه يكون
distance/radius الـ

B — depot

A

by Dividing customers into group, based on

Start from C_2



Example 3:

Suppose you are the dispatcher for an LTL trucking firm, and you need to dispatch small delivery trucks for the local delivery of freight. Suppose you have many trucks available, each with the capacity to haul 10 tons of freight. Suppose that the polar angle (relative to the x-axis, and measured clockwise in degrees) of the seven customers are {45, 105, 220, 135, 60, 85, 300}. Suppose that the demands of the seven customers in tons are {7, 3, 4, 5, 2, 4, 6}.

a) Using the Sweep Heuristic, generate clusters of customers each to be served by a single vehicle. Begin with a ray through customer 2 and rotate your ray clockwise.

Customer	demand	θ_i
1	7	45
2	3	105
3	4	220
4	5	135
5	2	60
6	4	85
7	6	300

$T_0 = \{ C_2, C_4, C_3, C_7, C_1, C_5, C_6 \}$

Clusters:

$C_1 = \{C_2, C_4\}$

$C_2 = \{C_3, C_7\}$

$C_3 = \{C_1, C_5\}$

$C_4 = \{C_6\}$

Customer	θ	q_i
2	105	3
4	135	5
3	220	4
7	300	6
1	45	7
5	60	2
6	85	4

$T_0 = \{2, 4, 3, 7, 1, 5, 6\}$ (3, 10, 9, 4)

$\{0, 2, 4, 0\}$

$\{0, 3, 7, 0\}$

$\{0, 1, 5, 0\}$

$\{0, 6, 0\}$

Scanned with CamScanner

9 x 10 → no all feasible due to capacity
2
90% ↑

fast construction heuristic

Clarke-Wright Savings Heuristic

of vehicle = # of customer

The Clarke-Wright method begins with a poor initial solution in which each vehicle visits a single customer. Tours are then combined as long as total cost savings are produced. Suppose the depot is node 0.

Generic Clarke-Wright

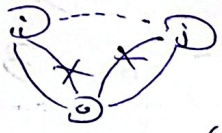
$T_i = \{0, i, 0\}$

1. For each node $i \in N$, construct a tour $T_i = \{i\}$ beginning at the depot, traveling to node i, and returning to the depot. Let S be the set of tours.
2. For each pair of tours T_i and T_j in S (list of tours) such that the total customer demands of the two tours combined does not exceed Q (*i.e.*, $\sum_{k \in T_i \cup T_j} q_k \leq Q$), compute the savings: $S_{ij} = c_{end(i)0} + c_{0begin(j)} - c_{end(i)begin(j)}$.
3. Choose the pair of tours i and j that maximize S_{ij}. If $S_{ij} \geq 0$, then add $T_i + T_j$ to S and remove T_i and T_j from S. Else, end the heuristic.
4. If no tours were joined, end the heuristic. Else, repeat from Step 2.

Implemented Clarke-Wright Heuristic

You may note in the preceding presentation that savings need not be calculated more than once for each pair of nodes. Consider the following version of the heuristic:

1. For each node $i \in N$, construct a tour $T_i = \{i\}$ beginning at the depot, traveling to node i, and returning to the depot. Let S be the set of tours.
2. Calculate the savings $S_{ij} = c_{0i} + c_{j0} - c_{ij}$ for all pairs of nodes i, j.
3. Sort the savings in nonincreasing order. ↓
4. Walk down the savings combination list, and find the next feasible combination pair (i, j) such that:
   - i and j are on different tours, and i is the last node on its tour T_k and j is the first node on its tour T_ℓ.
   - $\sum_{k \in T_i \cup T_j} q_k \leq Q$.
   - $S_{ij} \geq 0$



$T_1 = \{0, i, 0\}$
$T_2 = \{0, j, 0\}$
$(c_{i0} + c_{0j} - c_{ij}) \geq 0$

Add the combined tour $T_i + T_j$ to S, and remove T_i and T_j. Repeat this step as long as feasible combinations remain in the savings list.

* Any improvement approach that we apply to TSP sol, also applicable to VRP solution
2-exchange is also applicable for VRP sol

# of arcs added = # of arcs deleted

LIST:

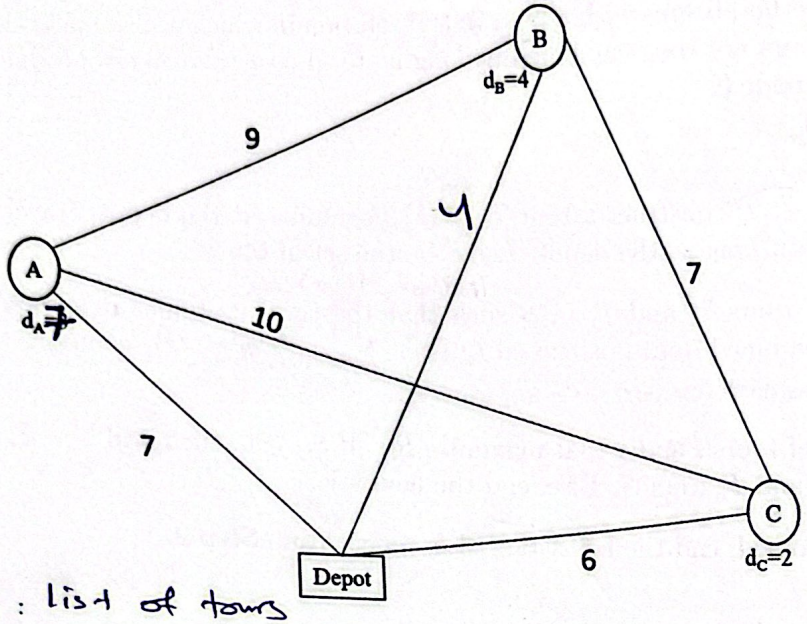
~~{0, A, 0}~~

{0, B, 0}

~~{0, C, 0}~~

{0, A, C, 0}

Simple Example



4 + 7 = ~~16~~

S : list of tours

① T_A : {0, A, 0} = 7+7 = 14

T_B = {0, B, 0} = 8

T_C = {0, C, 0} = 12

ΣC_T = 34

Q = 10

List of Possible Savings

② $S_{AB} = S_{0A} + C_{0B} - C_{BA} = 2$

$S_{AC} = C_{0A} + C_{0C} - C_{AC} = 3$

$S_{BC} = C_{0B} + C_{0C} - C_{BC} = 3$

③ Sorted list of savings:

S_{AC} = 3 ✓

~~S_{BC} = 3~~

~~S_{AB} = 2~~ Feasible

Combine T_A & T_C → $T_A + T_C$: {0, A, C, 0} Σq = 7 + 2 = 9

T_1 = {0, A, C, 0} → C_T = 7+10+6 = 23 → 31

T_2 = {0, B, 0} → 8

= 34 − 3 = 31

($\bar{C}$ saving, T_0 (cost))

Final
7 customers

Clarke Wright heuristics

A warehouse, located at node **C0**, is trying to route its trucks to serve all customers. The travel distances are shown in table below, and the daily demand for each customer is given in the table. Each truck has a capacity of 15. You decide to solve the problem using Clarke wright algorithm to generate a set of routes to serve customers.

Customer	C1	C2	C3	C4	C5	C6
Demand	4	6	3	5	3	6

Travel distances between customers

	C0	C1	C2	C3	C4	C5	C6
C0	-----	20	18	14	16	12	19
C1		-----	22	18	30	26	28
C2			-----	32	20	22	21
C3				-----	20	22	21
C4					-----	30	32
C5						-----	26
C6							-----

According to Clark Wright's heuristic, what are the construct routes? What is the total distance traveled by utilized vehicles?

Step 1:-

$T_1 = \{C_0, C_1, C_0\} = 40$

$T_2 = \{C_0, C_2, C_0\} = 36$

$T_3 = \{C_0, C_3, C_0\} = 28$

$T_4 = \{C_0, C_4, C_0\} = 32$

$T_5 = \{C_0, C_5, C_0\} = 24$

$T_6 = \{C_0, C_6, C_0\} = 38$

$\sum_{1}^{6} C_T = 198$

Possible combs:- $(6 \times 5)/2 = 15$

$C_1, C_2 = C_{10} + C_{02} - C_{12} = 16$ ✓

$C_1, C_3 = C_{10} + C_{03} - C_{13} = 16$ ✓

$C_1, C_4 = C_{10} + C_{04} - C_{14} = 6$

$C_1, C_5 = 6$

$C_1, C_6 = 11$

$C_2, C_3 = C_{20} + C_{30} - C_{23} = 0$

$C_2, C_4 = 14$

$C_2, C_5 = 8$

$C_2, C_6 = 16$

$C_3, C_4 = 10$

$C_3, C_5 = 4$

$C_3, C_6 = 12$

$C_4, C_5 = -2$

$C_4, C_6 = 3$

$C_5, C_6 = 5$

$T_{12} = \{C_0, C_1, C_2, C_0\}$

$T_{1,2,3} = \{C_0, C_3, C_1, C_2, C_0\}$

$T_{5,6} = \{C_0, C_5, C_6, C_0\}$

$T_{4,5,6} = \{C_0, C_5, C_6, C_4, C_0\}$

$198 - (16+16+5+3) = 158$

→ Cost

Tot Savings $= 16+16+5+3 = 40$

$C_2, C_1 = 16$

$C_1, C_3 = 16$

$C_2, C_6 = 16$

$C_2, C_4 = 14$

$C_3, C_6 = 12$ ✗

$C_1, C_6 = 11$ ✗

$C_3, C_4 = 10$ ✗

$C_2, C_5 = 8$ ✗

$C_1, C_4 = 6$ ✗

$C_1, C_5 = 6$ ✗

$C_5, C_6 = 5$ ✓

$C_3, C_5 = 4$ ✗

$C_1, C_6 = 3$

$C_2, C_3 = 0$ ✗

$C_4, C_5 = -2$ ✗

Scanned with CamScanner

SWAP

↳ we move customer i from its position to customer j's position, & j to customer i's position.

↳ must satisfy capacity constraints (for different tours)

$$Savings = C_{pred(i),i} + C_{i,succ(i)} + C_{pred(j),j} + C_{j,succ(j)} - \left(C_{pred(j),i} + C_{i,succ(j)} + C_{pred(i),j} + C_{j,succ(i)} \right)$$

$T_{4,5,6} = \{C_0, C_5, C_6, C_4, C_0\}$

$T_{1,2,3} = \{C_0, C_3, C_1, C_2, C_0\}$

SWAP (6, 1)

↳ $Saving = C_{5,6} + C_{6,4} + C_{3,1} + C_{1,2} - (C_{5,1} + C_{1,4} + C_{3,6} + C_{6,2})$

$= 98 - 98 = 0$ no Improvement

Insertion Improvement .. INSRT (3, 1)

$T_{4,1,5} = \{C_0, C_5, C_1, C_4, C_0\}$

$T_{2,6,3} = \{C_0, C_3, C_6, C_2, C_0\}$

+ node 3 to the first tour

$Saving = C_{0,3} + C_{3,6} + C_{1,4} - (C_{1,3} + C_{5,4} + C_{0,6})$

$= 14 + 21 + 30 - (18 + 20 + 19) = 10$

$T_1 = \{C_0, C_5, C_1, C_3, C_4, C_0\}$

$T_2 = \{C_0, C_6, C_2, C_0\}$

Scanned with CamScanner