University of Ruhuna
Faculty of Engineering

Take Home Assignment

Assignment 1

EE7220/EC7208

Optimization Techniques for Engineers

Group Members

01	Ashfaq M.R.M	EG/2021/4417
02	Munsif M.F.A	EG/2021/4684

Contents

1. HPC Cloud Server Resource Allocation (Knapsack Problem)

1.1. Scenario

LankaCloud Pvt Ltd is a cloud computing provider based in Colombo, Sri Lanka. The company operates a High-Performance Computing (HPC) server that can host managed application instances for its clients. The server has a limited compute capacity shared among all hosted applications.
LankaCloud offers three types of managed application hosting packages:

k	Application Type	vCPU per instance	Monthly Profit per instance
1	Web Application Server	3 vCPU	Rs. 7,000
2	Mobile API Server	5 vCPU	Rs. 12,000
3	ML Inference Server	9 vCPU	Rs. 25,000

The HPC server has a total capacity of **20 vCPU units**. Multiple instances of the same application type may be hosted simultaneously. LankaCloud wants to determine how many instances of each application type to host in order to **maximise total monthly profit**, subject to the server's capacity limit.

1.2. Resource Abstraction: vCPU Unit

Real HPC servers have multiple resource dimensions: CPU cores, RAM, and storage. To apply the standard single-constraint Knapsack formulation, all resources are unified into a single **virtual CPU (vCPU)** metric. Each vCPU unit represents a fixed bundle of:

- 2 physical CPU cores
- 4 GB RAM
- 20 GB SSD storage

Application resource requirements expressed in vCPU units:

Application Type	CPU cores	RAM	Storage	vCPU units
Web Application Server	6 cores	8 GB	40 GB	**3 vCPU**
Mobile API Server	10 cores	20 GB	100 GB	**5 vCPU**
ML Inference Server	18 cores	36 GB	180 GB	**9 vCPU**

Total server capacity: 40 CPU cores / 80 GB RAM / 400 GB storage = **20 vCPU units**.

1.3. Mathematical Formulation

Parameters:

- $N = 3$ (number of application types)
- $C = 20$ (total server capacity in vCPU units)
- $w_1 = 3,\ w_2 = 5,\ w_3 = 9$ (vCPU consumed per instance)
- $r_1 = 7{,}000,\ r_2 = 12{,}000,\ r_3 = 25{,}000$ (monthly profit in Rs. per instance)

Decision Variables:

- x_k = number of instances of application type k to host (non-negative integer), $k = 1, 2, 3$

Objective Function:

$$\text{Maximise } Z = 7{,}000x_1 + 12{,}000x_2 + 25{,}000x_3$$

Capacity Constraint:

$$3x_1 + 5x_2 + 9x_3 \leq 20$$

Integrality Constraint:

$$x_1, x_2, x_3 \in \{0, 1, 2, \ldots\}$$

Each application type can be hosted multiple times, so x_k is a non-negative integer with no upper bound. This makes it an **unbounded integer programme** — Simplex cannot handle integrality constraints, so we use **Dynamic Programming** instead.

1.4. Dynamic Programming Formulation

Stages: Application types $k = 1, 2, 3$ (processed from $k = 3$ down to $k = 1$)
State: Remaining server capacity i, where $0 \leq i \leq 20$
Optimal Value Function:

$$V_k(i) = \text{maximum monthly profit achievable from application types } k, k+1, \ldots, 3$$

$$\text{with remaining capacity } i$$

Boundary Condition:

$$V_4(i) = 0 \quad \text{for all } i$$

Recurrence Relation:

$$V_k(i) = \max\{r_k x_k + V_{k+1}(i - w_k x_k)\}$$

$$x_k \geq 0, \quad w_k x_k \leq i$$

1.5. Stage 3: ML Inference Server (w3 = 9, r3 = Rs. 25,000)

$$V_3(i) = \max\{25{,}000 \cdot x_3 : x_3 \geq 0,\ 9x_3 \leq i\}$$

Remaining Capacity (i)	Optimal x_3^*	$V_3(i)$ (Rs.)
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	1	25,000
10	1	25,000
11	1	25,000
12	1	25,000
13	1	25,000
14	1	25,000
15	1	25,000
16	1	25,000
17	1	25,000
18	2	50,000
19	2	50,000
20	2	50,000

1.6. Stage 2: Mobile API Server + ML Inference (w2 = 5, r2 = Rs. 12,000)

$$V_2(i) = \max\{12{,}000 \cdot x_2 + V_3(i - 5x_2) : x_2 \geq 0,\ 5x_2 \leq i\}$$

i	x_2=0 : $V_3(i)$	x_2=1 : 12k+$V_3(i-5)$	x_2=2 : 24k+$V_3(i-10)$	x_2=3 : 36k+$V_3(i-15)$	x_2=4 : 48k+$V_3(i-20)$	Optimal x_2^*	$V_2(i)$ (Rs.)
0	0	—	—	—	—	0	0
1	0	—	—	—	—	0	0
2	0	—	—	—	—	0	0
3	0	—	—	—	—	0	0
4	0	—	—	—	—	0	0
5	0	12,000	—	—	—	1	12,000
6	0	12,000	—	—	—	1	12,000
7	0	12,000	—	—	—	1	12,000
8	0	12,000	—	—	—	1	12,000
9	25,000	12,000	—	—	—	0	25,000
10	25,000	12,000	24,000	—	—	0	25,000
11	25,000	12,000	24,000	—	—	0	25,000
12	25,000	12,000	24,000	—	—	0	25,000
13	25,000	12,000	24,000	—	—	0	25,000
14	25,000	**37,000**	24,000	—	—	1	37,000
15	25,000	**37,000**	24,000	36,000	—	1	37,000
16	25,000	**37,000**	24,000	36,000	—	1	37,000
17	25,000	**37,000**	24,000	36,000	—	1	37,000
18	**50,000**	37,000	24,000	36,000	—	0	50,000
19	**50,000**	37,000	49,000	36,000	—	0	50,000
20	**50,000**	37,000	49,000	36,000	48,000	0	50,000

Note: At $i = 14$, $x_2 = 1$ gives $12{,}000 + V_3(9) = 12{,}000 + 25{,}000 =$ **Rs. 37,000**, which beats the ML-only option of $V_3(14) = 25{,}000$. At $i = 18$, two ML instances ($V_3(18) = 50{,}000$) beat any Mobile API combination.

1.7. Stage 1: All Application Types (Capacity = 20 vCPU)

$$V_1(20) = \max\{7{,}000 \cdot x_1 + V_2(20 - 3x_1) : x_1 \geq 0,\ 3x_1 \leq 20\}$$

x_1	$7{,}000 \times x_1$ (Rs.)	Remaining capacity $(20 - 3x_1)$	$V_2(20 - 3x_1)$ (Rs.)	Total (Rs.)
0	0	20	50,000	50,000
1	7,000	17	37,000	44,000
2	**14,000**	**14**	**37,000**	**51,000 ← max**
3	21,000	11	25,000	46,000
4	28,000	8	12,000	40,000
5	35,000	5	12,000	47,000
6	42,000	2	0	42,000

Optimal decision at Stage 1: $x_1^* = 2$

1.8. Traceback: Recovering the Optimal Solution

Step	Stage	Remaining capacity	Optimal x_k^*	vCPU used	Profit earned
1	$k=1$	20	$x_1^* = \mathbf{2}$	$2\times 3 = 6$	$2\times 7{,}000 =$ Rs. 14,000
2	$k=2$	$20-6=14$	$x_2^* = \mathbf{1}$	$1\times 5 = 5$	$1\times 12{,}000 =$ Rs. 12,000
3	$k=3$	$14-5=9$	$x_3^* = \mathbf{1}$	$1\times 9 = 9$	$1\times 25{,}000 =$ Rs. 25,000

Total vCPU used: $6+5+9 =$ **20/20** (server fully utilised)

1.9. Optimal Solution

Host 2 Web Application Servers, 1 Mobile API Server, and 1 ML Inference Server.

Maximum monthly profit = Rs. 14,000 + Rs. 12,000 + Rs. 25,000 = Rs. 51,000

1.10. Why Greedy Fails

A greedy approach ranks applications by profit-per-vCPU (value-to-weight ratio):

Application Type	Profit/vCPU	Greedy rank
ML Inference Server	$25{,}000/9 \approx$ **2,778**	1st
Mobile API Server	$12{,}000/5 =$ **2,400**	2nd
Web Application	$7{,}000/3 \approx$ **2,333**	3rd

Greedy execution (capacity = 20 vCPU):

1. Pick ML Inference (9 vCPU) $\rightarrow$ capacity remaining: 11 vCPU, profit: Rs. 25,000
2. Pick ML Inference again (9 vCPU) $\rightarrow$ capacity remaining: 2 vCPU, profit: Rs. 50,000
3. No application fits in 2 vCPU $\rightarrow$ **stop**

Greedy result: $x_1 = 0$, $x_2 = 0$, $x_3 = 2 \rightarrow$ **Rs. 50,000** (2 vCPU wasted)

Method	x_1	x_2	x_3	vCPU used	Monthly Profit
Greedy	0	0	2	18/20	Rs. 50,000
DP	**2**	**1**	**1**	**20/20**	**Rs. 51,000**

So greedy wastes 2 vCPU and earns Rs. 1,000 less per month. The DP approach avoids this by checking every feasible combination at each stage, so no capacity goes unused unnecessarily.

2. Network Optimization Using the Reverse-Delete MST Algorithm

2.1. Emergency Fiber Network for Disaster Response

A city needs to connect 10 critical facilities with a fiber communication network that can hold up during floods and cyclones. Engineers have surveyed several possible cable routes along existing roads. The goal is to make sure all facilities can reach each other (directly or via other nodes) while keeping total installation cost as low as possible.

2.2. Graph Model

2.2.1 Facilities (Vertices)

- H: Main Hospital
- P: Police HQ
- F: Fire Station
- G: City Hall / Disaster Control Center
- W: Water Treatment Plant
- U: Power Substation
- T: Telecom Hub
- S: Central School (relief center)
- M: Main Market (supplies hub)
- B: Bus Depot (evacuation transport)

2.2.2 Candidate Fiber Links (Edges with Costs)

Costs are expressed in generic installation cost units. The network is undirected.

Edge	Cost	Edge	Cost
H-G	22	M-B	21
T-B	20	U-B	19
W-B	18	W-U	17
S-M	16	G-T	15
G-S	14	F-B	13
F-G	12	P-M	11
P-G	10	H-T	9
H-P	8	H-F	7
H-S	6	P-F	5
G-U	4	S-W	3
U-T	2	T-M	1

This is essentially a Minimum Spanning Tree (MST) problem — we want the cheapest set of links that still keeps all 10 facilities connected. With 10 nodes, the spanning tree will always have exactly 9 edges.

2.3. Selected MST Technique: Reverse-Delete Algorithm

For a connected, undirected, weighted graph $G = (V, E)$, an MST is a subset of edges $T \subseteq E$ that connects all vertices, contains no cycles, and minimizes total weight $\sum_{e \in T} w(e)$.
The Reverse-Delete algorithm works on the cycle property: within any cycle in the graph, the heaviest edge is not needed for the MST. Removing it still leaves every vertex reachable through the remaining edges in the cycle, just at a lower cost.

2.3.1 Steps

1. Take the full graph with all candidate links.
2. Sort all edges from highest cost to lowest.
3. Go through each edge: try removing it. If the network is still connected, drop it for good. If removing it disconnects anything, put it back and move on.
4. Whatever edges remain at the end form the MST.

This approach suits the scenario well — the city already has a full list of proposed cable routes, and the algorithm trims the expensive ones first without ever breaking connectivity.

2.4. Step-by-Step Application of Reverse-Delete

2.4.1 Edge Order (Descending)

Edges are processed from highest cost to lowest cost: 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1.

2.4.2 Reverse-Delete Decision Table

Each edge is checked in order — if removing it keeps the graph connected it gets deleted, otherwise it stays.

Step	Edge	Cost	Decision	Connectivity check (reason)
1	H-G	22	REMOVE	H still reaches G via H-T-U-G.
2	M-B	21	REMOVE	M still reaches B via M-T-B (later replaced by M-T-U-B paths; at this step B-links still exist).
3	T-B	20	REMOVE	T still reaches B via T-U-B.
4	U-B	19	REMOVE	U still reaches B via U-W-B.
5	W-B	18	REMOVE	W still reaches B via W-U-G-F-B.
6	W-U	17	REMOVE	W still reaches U via W-S-H-T-U.
7	S-M	16	REMOVE	S still reaches M via S-H-T-M.
8	G-T	15	REMOVE	G still reaches T via G-U-T.
9	G-S	14	REMOVE	G still reaches S via G-U-T-H-S.
10	F-B	13	KEEP	If removed, B becomes isolated (all other B links have already been deleted).
11	F-G	12	REMOVE	F still reaches G via F-H-T-U-G.
12	P-M	11	REMOVE	P still reaches M via P-F-H-T-M.
13	P-G	10	REMOVE	P still reaches G via P-F-H-T-U-G.
14	H-T	9	KEEP	If removed, the graph splits {G,U,T,M} from {H,S,W,F,P,B}.
15	H-P	8	REMOVE	H still reaches P via H-F-P.
16	H-F	7	KEEP	If removed, {F,P,B} disconnects from the rest.
17	H-S	6	KEEP	If removed, {S,W} disconnects (S has no other remaining links).
18	P-F	5	KEEP	If removed, P becomes isolated (H-P already deleted).
19	G-U	4	KEEP	If removed, G becomes isolated (all other G links were deleted).
20	S-W	3	KEEP	If removed, W becomes isolated (W-U and W-B were deleted).
21	U-T	2	KEEP	If removed, {U,G} disconnects from the rest.
22	T-M	1	KEEP	If removed, M becomes isolated (S-M and P-M were deleted).

2.5. Final MST and Minimum Total Cost

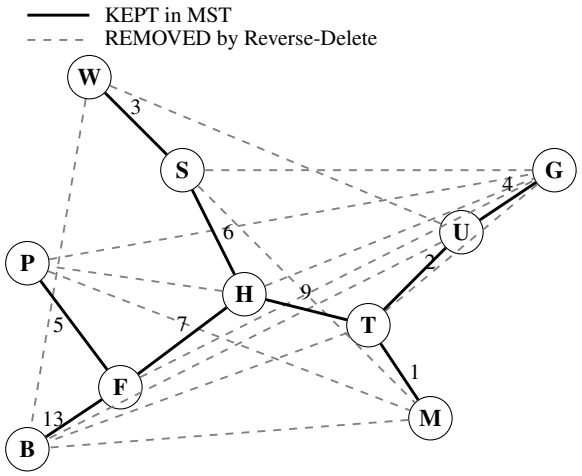
After processing all edges, the remaining edges (marked KEEP) form the MST.

MST edge	Cost
T-M	1
U-T	2
S-W	3
G-U	4
P-F	5
H-S	6
H-F	7
H-T	9
F-B	13

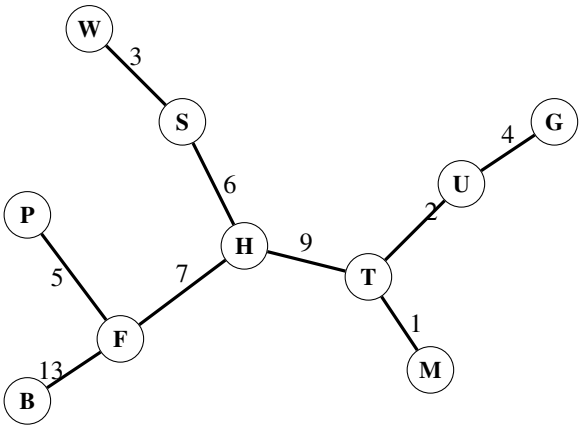
Total minimum cost $= 1+2+3+4+5+6+7+9+13 = 50$.
One way to visualize the final tree (connectivity path) is:
W - S - H - T - U - G, with branches H - F - P and F - B, and T - M.

2.6. Graph Overview

The graphs below provide a visual summary of the Reverse-Delete decisions and the final MST structure.
Remove vs Keep Decisions:



Dashed links were removed because the network stayed connected without them. Solid links are the ones that had to be kept — removing any of them would have split the network. These 9 solid links form the final MST with total cost 50.
Final MST (selected links only):



MST cost from the graph: $1+2+3+4+5+6+7+9+13=50$.