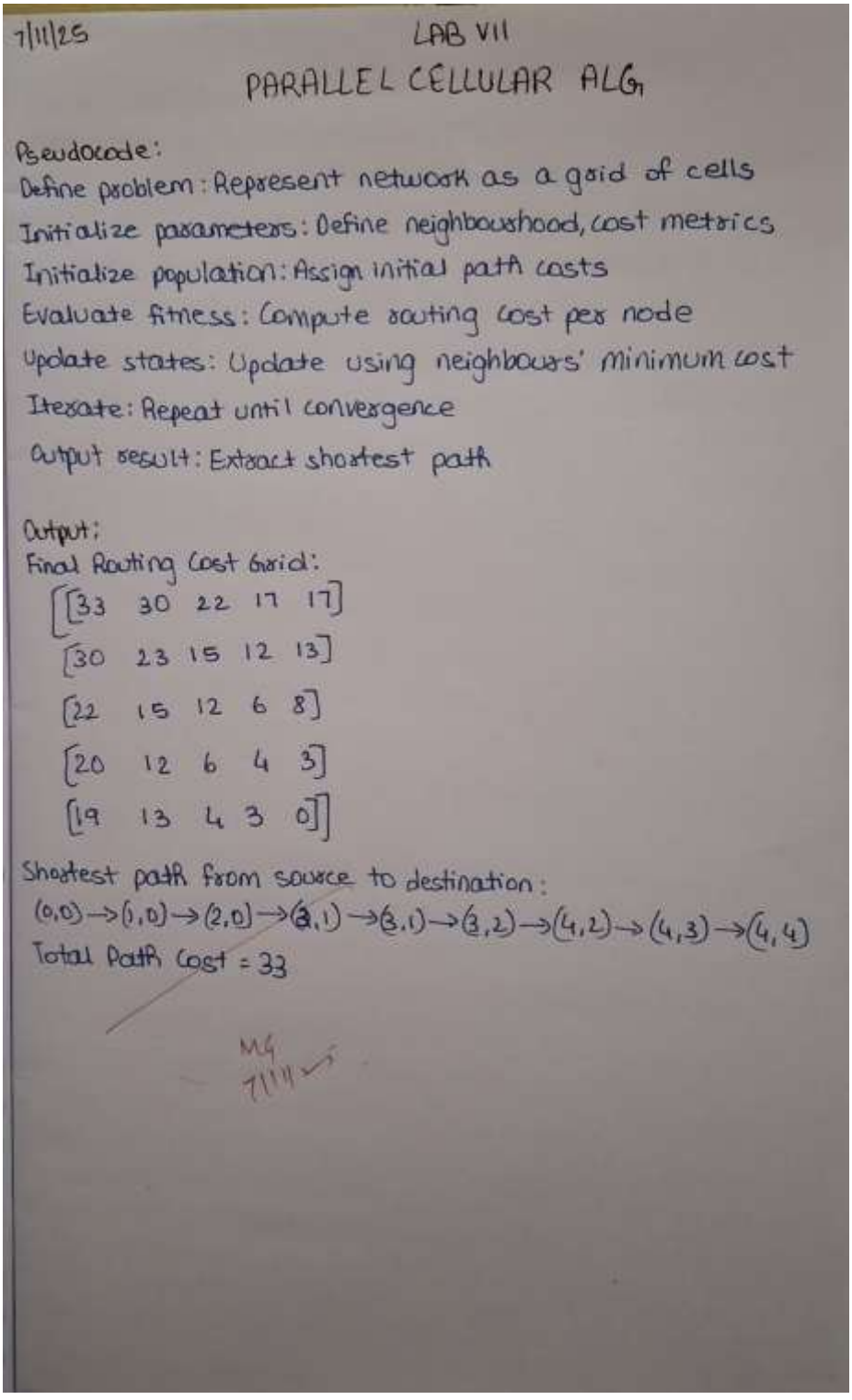
Lab 7

Parallel Cellular Algorithm



7/11/25

LAB VII

PARALLEL CELLULAR ALG

Pseudocode:

Define problem: Represent network as a grid of cells

Initialize parameters: Define neighbourhood, cost metrics

Initialize population: Assign initial path costs

Evaluate fitness: Compute routing cost per node

Update states: Update using neighbours' minimum cost

Iterate: Repeat until convergence

Output result: Extract shortest path

Output:

Final Routing Cost Grid:

```
[[33 30 22 17 17]
 [30 23 15 12 13]
 [22 15 12  6  8]
 [20 12  6  4  3]
 [19 13  4  3  0]]
```

Shortest path from source to destination:

(0,0) → (1,0) → (2,0) → (3,1) → (3,2) → (4,2) → (4,3) → (4,4)

Total Path Cost = 33

MG
7/11

Code:

```
import numpy as np
print('Shreya Raj 1BM23CS317')
# ----------------------------------
# Step 1: Define Network as Grid
# ----------------------------------
GRID_SIZE = 5 # 5x5 network grid
MAX_ITER = 100
INF = 1e9

# Define source and destination
source = (0, 0)
destination = (4, 4)

# Create cost matrix for each link (random or distance-based)
np.random.seed(42)
cost_matrix = np.random.randint(1, 10, size=(GRID_SIZE, GRID_SIZE))

# Initialize state matrix (each cell's current best cost to destination)
state = np.full((GRID_SIZE, GRID_SIZE), INF)
state[destination] = 0 # destination cost = 0

# Define 4-neighborhood (up, down, left, right)
neighbors = [(-1, 0), (1, 0), (0, -1), (0, 1)]

# ----------------------------------
# Step 2: Define Helper Function
```

```
# ----------------------------------
def get_neighbors(i, j):
    """Return valid neighboring cells"""
    valid_neighbors = []
    for dx, dy in neighbors:
        ni, nj = i + dx, j + dy
        if 0 <= ni < GRID_SIZE and 0 <= nj < GRID_SIZE:
            valid_neighbors.append((ni, nj))
    return valid_neighbors


# ----------------------------------
# Step 3: PCA Iteration for Routing
# ----------------------------------
for iteration in range(MAX_ITER):
    new_state = state.copy()
    for i in range(GRID_SIZE):
        for j in range(GRID_SIZE):
            if (i, j) == destination:
                continue
            neighbor_costs = []
            for ni, nj in get_neighbors(i, j):
                # Update rule: min(cost to neighbor + neighbor's state)
                total_cost = cost_matrix[ni, nj] + state[ni, nj]
                neighbor_costs.append(total_cost)
            if neighbor_costs:
                new_state[i, j] = min(neighbor_costs)
    # Check for convergence
```

```
    if np.allclose(new_state, state):
        print(f"Converged after {iteration} iterations.")
        break
    state = new_state

# ----------------------------------
# Step 4: Extract Path from Source
# ----------------------------------
path = [source]
current = source
while current != destination:
    i, j = current
    nbs = get_neighbors(i, j)
    next_cell = min(nbs, key=lambda n: state[n])
    path.append(next_cell)
    current = next_cell

# ----------------------------------
# Step 5: Display Results
# ----------------------------------
print("Final Routing Cost Grid:")
print(np.round(state, 2))
print("\nShortest Path from Source to Destination:")
print(" → ".join([str(p) for p in path]))
print(f"\nTotal Path Cost: {state[source]}")
```

Output:

```
Shreya Raj 1BM23CS317
Converged after 8 iterations.
Final Routing Cost Grid:
[[33. 30. 22. 17. 17.]
 [30. 23. 15. 12. 13.]
 [22. 15. 12.  6.  8.]
 [20. 12.  6.  4.  3.]
 [19. 13.  4.  3.  0.]]

Shortest Path from Source to Destination:
(0, 0) → (1, 0) → (2, 0) → (2, 1) → (3, 1) → (3, 2) → (4, 2) → (4, 3) → (4, 4)

Total Path Cost: 33.0
```