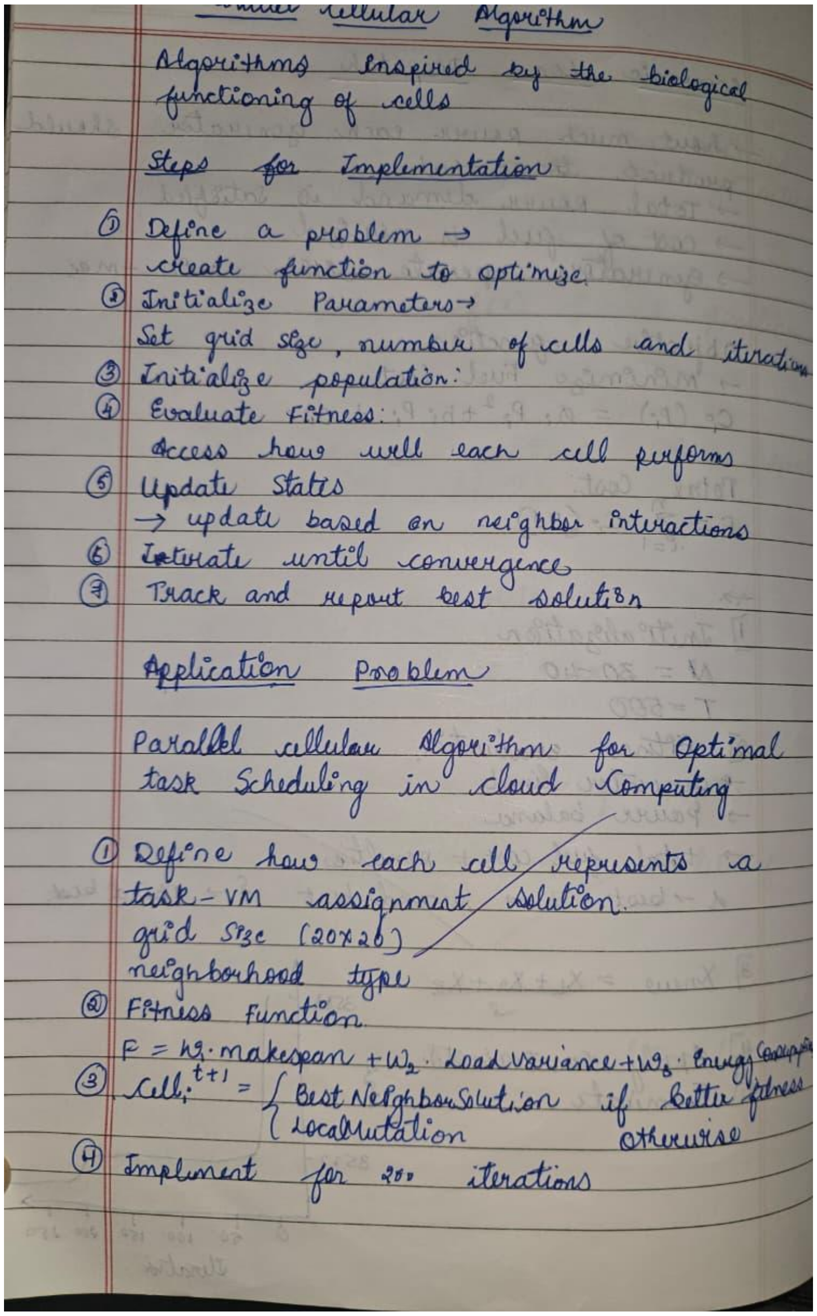
Program 6

Parallel Cellular Algorithms and Programs The task is to perform edge detection or noise reduction in an image using Parallel Cellular Automata (PCA), where each pixel (cell) interacts with its neighbors to enhance edges or reduce noise iteratively.

Algorithm:



Cellular Algorithm

Algorithms inspired by the biological functioning of cells

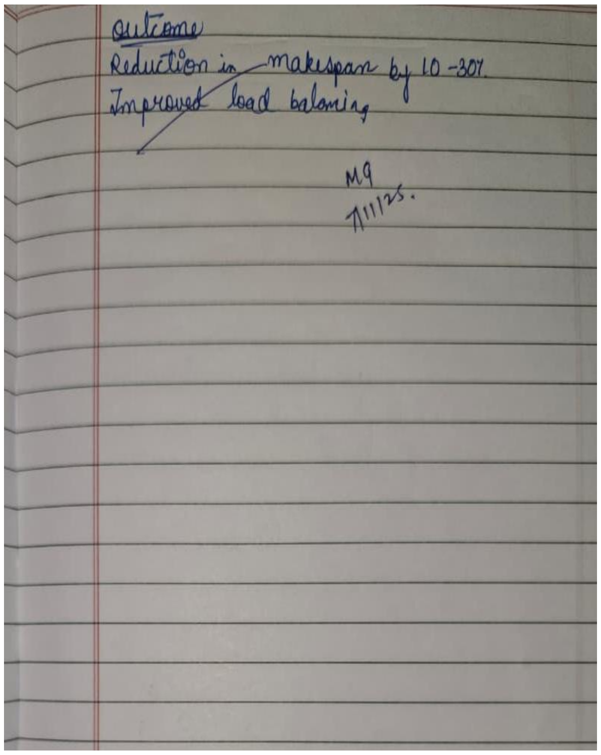
Steps for Implementation

1. Define a problem →
   create function to optimize
2. Initialize Parameters →
   Set grid size, number of cells and iterations
3. Initialize population:
4. Evaluate Fitness:
   Access how well each cell performs
5. Update States
   → update based on neighbor interactions
6. Iterate until convergence
7. Track and report best solution

Application Problem

Parallel cellular Algorithm for Optimal task Scheduling in cloud Computing

1. Define how each cell represents a task-VM assignment solution.
   grid size (20×20)
   neighborhood type
2. Fitness Function.
   $F = w_1 \cdot \text{makespan} + w_2 \cdot \text{Load Variance} + w_3 \cdot \text{Energy Consumption}$
3. $\text{cell}_i^{t+1} = \begin{cases} \text{Best Neighbour Solution} & \text{if better fitness} \\ \text{Local Mutation} & \text{otherwise} \end{cases}$
4. Implement for 200 iterations



outcome

Reduction in makespan by 10-30%

Improved load balancing

M9

7/11/25.

Code:

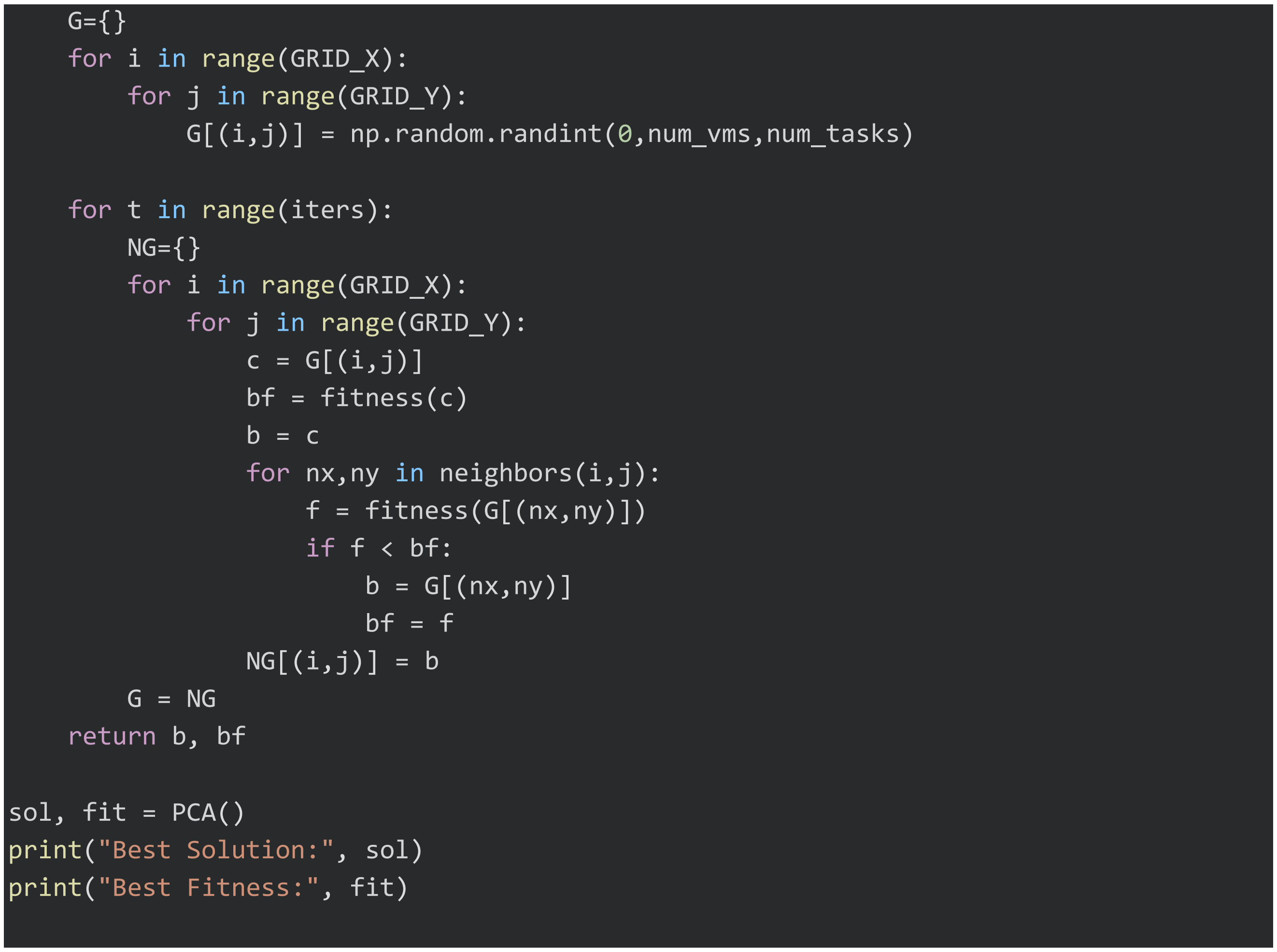
```python
import numpy as np

GRID_X, GRID_Y = 20, 20
num_tasks = 10
num_vms = 4
w1, w2, w3 = 0.5, 0.3, 0.2

def fitness(x):
    loads = np.zeros(num_vms)
    for i in range(num_tasks):
        loads[x[i]] += np.random.randint(1, 10)
    return w1*np.max(loads) + w2*np.var(loads) + w3*(np.sum(loads)*0.1)

def neighbors(x, y):
    r=[]
    for dx,dy in [(1,0),(-1,0),(0,1),(0,-1)]:
        nx,ny=x+dx,y+dy
        if 0<=nx<GRID_X and 0<=ny<GRID_Y:
            r.append((nx,ny))
    return r

def PCA(iters=200):
```

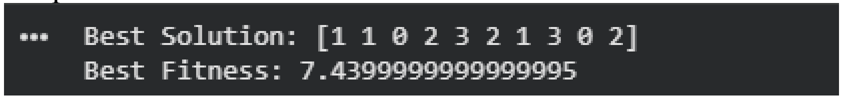


```
    G={}
    for i in range(GRID_X):
        for j in range(GRID_Y):
            G[(i,j)] = np.random.randint(0,num_vms,num_tasks)

    for t in range(iters):
        NG={}
        for i in range(GRID_X):
            for j in range(GRID_Y):
                c = G[(i,j)]
                bf = fitness(c)
                b = c
                for nx,ny in neighbors(i,j):
                    f = fitness(G[(nx,ny)])
                    if f < bf:
                        b = G[(nx,ny)]
                        bf = f
                NG[(i,j)] = b
        G = NG
    return b, bf

sol, fit = PCA()
print("Best Solution:", sol)
print("Best Fitness:", fit)
```

Output:



```
Best Solution: [1 1 0 2 3 2 1 3 0 2]
Best Fitness: 7.4399999999999995
```