Parallel Cellular Algorithms and Programs

```
import numpy as np
import math
import random
from multiprocessing import Pool, cpu_count

def rastrigin(x):
    x = np.asarray(x)
    n = x.size
    return 10 * n + ((x**2 - 10 * np.cos(2 * math.pi * x)).sum())

class CellularOptimizer:
    def __init__(
        self,
        obj_fn,
        dim,
        grid_shape=(20, 20),
        bounds=(-5.12, 5.12),
        init_scale=1.0,
        neighborhood="moore",
        move_rate=0.5,
        mutation_std=0.1,
        n_iters=200,
        use_parallel=False,
        seed=None
    ):
        if seed is not None:
            np.random.seed(seed)
            random.seed(seed)
        self.obj_fn = obj_fn
        self.dim = dim
        self.grid_shape = grid_shape
        self.n_cells = grid_shape[0] * grid_shape[1]
        self.bounds = np.array(bounds, dtype=float)
        self.init_scale = init_scale
        self.move_rate = move_rate
        self.mutation_std = mutation_std
        self.n_iters = n_iters
        self.neighborhood = neighborhood
        self.use_parallel = use_parallel

        low, high = self.bounds
        self.positions = np.random.uniform(low, high, size=(grid_shape[0], grid_shape[1], dim)) *
init_scale
        self.fitness = np.full((grid_shape[0], grid_shape[1]), np.inf, dtype=float)
        self.best_pos = None
```

```
    self.best_fit = np.inf

def _get_neighbors_idx(self, i, j):
    rows, cols = self.grid_shape
    neighbors = []
    for di in (-1, 0, 1):
        for dj in (-1, 0, 1):
            ni = (i + di) % rows
            nj = (j + dj) % cols
            neighbors.append((ni, nj))
    return neighbors

def _evaluate_one(self, pos):
    return self.obj_fn(pos)

def evaluate_fitness(self):
    flat_positions = self.positions.reshape((-1, self.dim))
    if self.use_parallel:
        with Pool(min(cpu_count(), 8)) as p:
            flat_f = p.map(self._evaluate_one, flat_positions)
        flat_f = np.asarray(flat_f, dtype=float)
    else:
        flat_f = np.asarray([self._evaluate_one(x) for x in flat_positions], dtype=float)
    self.fitness = flat_f.reshape(self.grid_shape)
    min_idx = np.unravel_index(np.argmin(self.fitness), self.grid_shape)
    if self.fitness[min_idx] < self.best_fit:
        self.best_fit = float(self.fitness[min_idx])
        self.best_pos = self.positions[min_idx].copy()

def step(self):
    rows, cols = self.grid_shape
    new_positions = self.positions.copy()
    for i in range(rows):
        for j in range(cols):
            neighbors = self._get_neighbors_idx(i, j)
            # find best neighbor (lowest fitness)
            best_n = min(neighbors, key=lambda ij: self.fitness[ij])
            best_pos = self.positions[best_n]
            curr_pos = self.positions[i, j]
            # move toward best neighbor
            direction = best_pos - curr_pos
            new_pos = curr_pos + self.move_rate * direction
            # mutation (Gaussian)
            new_pos = new_pos + np.random.normal(0, self.mutation_std, size=self.dim)
            # clip to bounds
            new_pos = np.clip(new_pos, self.bounds[0], self.bounds[1])
```

```
                new_positions[i, j] = new_pos
        self.positions = new_positions

    def run(self, verbose=False, record_history=False):
        history = []
        self.evaluate_fitness()
        history.append(self.best_fit)
        for t in range(1, self.n_iters + 1):
            self.step()
            self.evaluate_fitness()
            history.append(self.best_fit)
            if verbose and (t % max(1, self.n_iters // 10) == 0 or t == 1):
                print(f"Iter {t}/{self.n_iters}  best={self.best_fit:.6f}")
        if record_history:
            return self.best_pos, self.best_fit, np.asarray(history)
        return self.best_pos, self.best_fit

if __name__ == "__main__":
    # Example usage: optimize Rastrigin in 10D
    dim = 10
    grid_shape = (20, 20)
    opt = CellularOptimizer(
        obj_fn=rastrigin,
        dim=dim,
        grid_shape=grid_shape,
        bounds=(-5.12, 5.12),
        init_scale=1.0,
        move_rate=0.4,
        mutation_std=0.2,
        n_iters=300,
        use_parallel=False,
        seed=42
    )
    best_pos, best_fit, history = opt.run(verbose=True, record_history=True)
    print("Best fitness:", best_fit)
    print("Best position (first 10 dims):", best_pos[:10])

    # optional plotting of convergence
    try:
        import matplotlib.pyplot as plt
        plt.plot(history)
        plt.yscale("log")
        plt.xlabel("Iteration")
        plt.ylabel("Best fitness (log scale)")
        plt.title("Cellular optimizer convergence")
        plt.grid(True)
```

```
    plt.show()
except Exception:
    pass
```

Output:

```
Iter 1/300  best=71.099406
Iter 30/300  best=23.824319
Iter 60/300  best=21.343670
Iter 90/300  best=14.794851
Iter 120/300  best=10.387709
Iter 150/300  best=10.387709
Iter 180/300  best=10.387709
Iter 210/300  best=10.387709
Iter 240/300  best=10.387709
Iter 270/300  best=10.387709
Iter 300/300  best=10.387709
Best fitness: 10.387708598559811
Best position (first 10 dims): [-0.04856735  0.09811317  0.06555259 -0.12510798  0.00300656 -0.10706224
 -0.00423161  0.04226049 -0.08398669  0.04320281]
```