```python
import math
import random

# Constants
Ct = 0.88 # Pushing constant
Z = 63 # Tower height [m]
z0 = 0.3 # Field roughness
rr = 22 #  Blade length [m]
u0 = 12 #  Wind speed [m/s]
square_length = 200 # [m]
N = 10 # grid size [-]
ro = 1.23 # Air density [kg/m^3]
Cp = 0.4 # Power coefficient

debug = False

a_val = 1.0 * ((1 - math.sqrt(1 - Ct)) / 2)
alfa = 1.0 * (0.5 / math.log(Z / z0))
r1 = 1.0 * (rr * math.sqrt((1 - a_val) / (1 - 2 * a_val)))

# global vars sections
test_grid =[[1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
         [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
         [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
         [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
         [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
         [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
         [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
         [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
         [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
         [1, 0, 0, 0, 0, 1, 0, 0, 0, 1]]

#  _____________________ y
# |
# | (x1, y1)
# |
# |
# |            (x2, y2)
# |
# |
# x


def compute_influenced_wind(wind_speed, x1, y1, x2, y2):

   # first check if the turbines are in range
   # assume that (x1, y1) are always on the left of (x2, y2)
   # if they are not, return wind_speed
   if y1 >= y2:
      return wind_speed

   x = (y2 - y1)
   r = alfa * x + rr
   beta = math.atan(1.0 * r / x) * 180 / math.pi
   gamma = math.atan(1.0 * math.fabs(x1 - x2) / math.fabs(y1 - y2)) * 180 / math.pi

   # beta is the angle for which the first turbine influences the second
   # gamma is the angle between the first and the second turbine

   if debug:
      print("The angle between (x1, y1) = (%s, %s) and (x2, y2) = (%s, %s) is " \
         "%s; action angle is %s" % (x1, y1, x2, y2, gamma, beta))
```

```python
    # if gamma > beta, second turbine is influenced by the first,
    # otherwise, wind speed remains the same
    if gamma > beta:
        return wind_speed
    else:
        return wind_speed * (1.0 - 2.0 * a_val / (1.0 + alfa * (x / r1)**2))

def compute_turbine_wind(wind_speed, turbines_winds):
    return wind_speed * (1 - math.sqrt(sum([(1 - 1.0 * vi / wind_speed)**2 for vi in
turbines_winds])))

def compute_turbine_power(wind_speed):
    return 0.5 * ro * math.pi * rr**2 * Cp * wind_speed**3 / 1000 # [kW]

def compute_total_power(turbines_winds):
    return sum([compute_turbine_power(wind_speed) for wind_speed in turbines_winds])

def compute_cost(no_turbines):
    return no_turbines * (2.0 / 3.0 + 1.0 / 3.0 * math.e ** (-0.00174 * no_turbines**2))

# objective = cost / power
def compute_objective(no_turbines, turbines_winds):
    return 1.0 * compute_cost(no_turbines) / compute_total_power(turbines_winds)

def compute_influenced_turbines_winds(wind_speed, turbines_grid):
    turbines_influenced_winds = {}
    turbines_winds = {}
    for x in range(N):
        for y in range(N):
            # first turbine on column has wind speed equal to wind_speed
            if turbines_grid[x][y] == 1:
                turbines_influenced_winds[(x, y)] = [wind_speed]
                break

    for x1 in range(N):
        for y1 in range(N):
            if turbines_grid[x1][y1] == 0:
                continue

            turbines_winds[(x1, y1)] = compute_turbine_wind(wind_speed,
turbines_influenced_winds[(x1, y1)])
            if debug:
                print("Turbine (%s, %s) has wind speed of %s" % (x1, y1,
turbines_winds[(x1, y1)]))

            for x2 in range(N):
                for y2 in range(y1 + 1, N, 1):
                    if turbines_grid[x2][y2] == 0:
                        continue
                    influenced_winds = turbines_influenced_winds.get((x2, y2), [])
                    influenced_winds.append(compute_influenced_wind(wind_speed,
                        x1 * square_length, y1 * square_length, x2 * square_length, y2 *
square_length))
                    turbines_influenced_winds[(x2, y2)] = influenced_winds

    return turbines_winds

# Monte Carlo
def generate_random_coordinates(no_coordinates):
    coordinates = {}
```

```
    for i in range(no_coordinates):
        while True:
            (x, y) = (random.randint(0, N - 1), random.randint(0, N - 1))
            if not coordinates.get((x, y)):
                coordinates[(x, y)] = 1
                break

    return coordinates.keys()

def monte_carlo():
    no_iterations = 100

    good_turbines_grid = None
    good_no_turbines = -1
    good_objective = 100000
    good_power = -1
    good_cost = -1

    for i in range(no_iterations):
        no_turbines = random.randint(20, 40)
        turbines_grid = [[0 for x in range(N)] for x in range(N)]
        coordinates = generate_random_coordinates(no_turbines)
        for (x, y) in coordinates:
            turbines_grid[x][y] = 1

        turbines_winds = compute_influenced_turbines_winds(u0, turbines_grid)
        power = compute_total_power(turbines_winds.values())
        cost = compute_cost(no_turbines)
        objective = compute_objective(no_turbines, turbines_winds.values())
        print(f"Power is {power}, cost is {cost}, objective {objective}, turbines
{no_turbines}")

        if objective < good_objective:
            good_objective = objective
            good_turbines_grid = turbines_grid
            good_no_turbines = no_turbines
            good_power = power
            good_cost = cost

    print('\n'.join([''.join(['{:2}'.format(item) for item in row]) for row in
good_turbines_grid]))
    print(f"The power produced {good_power} [kW], Cost {good_cost} [u.r.], Objective
function value {good_objective}, Number of turbines {good_no_turbines}")


if __name__ == "__main__":
    monte_carlo()
```