Systems Biology – Exercises

Week 12: Swarm Intelligence

Swarm Intelligence

- Since the concept of emergence, we have seen characteristics form out of simple elements into complex behavior. We will see parts of cellular automata (mathematical concept) and genetic algorithm (error and fitness) in swarm intelligence.
- Many algorithms that fall in the swarm intelligence category have a metaphoric name such as Ant Colony Optimization, Artificial Bee Colony, Flock of Birds and other. We of course would expect this, since this is a Systems Biology exercise class. There is also an approach, which conceptualizes swarm intelligence as a particle swarm.

Optimization using Swarm Intelligence

- Mathematically similar to a cellular automaton, we use individual computer agents to scout their environment. These individual entities are very simple and would almost always fail in finding a satisfactory solution. However, using numerous same computer agents distributed over a solution space, optimization problems can be addressed.
- We first want to optimize (find a minimum) of the following Function 1.

$$f(x) = 5(x - 20)^2 \qquad (1)$$

Simple Function to Optimize

Code 1. Simple Function to Optimize.

```
5 def errorFunction(x):
6     return sum([5 * (x[i] - 20)**2 for i in range(len(x))])
```

Computer Agents as a Python Class

- Using the next code, we implement the computer agent as a class. Don’t forget to import the random and math package.

Exercise 1

- In the following code 2, provide four more object parameters: individualVelocity as an empty list, individualBestPosition as an empty list, individualBestError initialized with −1, individualError initialized with −1. Follow the example of the line above the comment ### Exercise ←

Code 2 (on the next page).
Class of the Computer Agent (Particle)

```python
# A class of a computer agent, here also called "particle"
class individualParticle:
    def __init__(self, x0):
        self.individualPosition = []
        ### Exercise<--
        ###
        ###

        for i in range(0, numStartingLocation):
            self.individualVelocity.append(random.uniform(-1, 1))
            self.individualPosition.append(x0[i])

    # Evaluate the current fitness of the computer agents
    def evaluate(self, costFunction):
        self.individualError = costFunction(self.individualPosition)

        # The current position of the agent is compared with the individual best and updated if necessary
        if self.individualError < self.individualBestError or self.individualBestError == -1:
            self.individualBestPosition = self.individualPosition
            self.individualBestError = self.individualError

    # Calculate the new agent's velocity
    def update_velocity(self, groupsBestPosition):
        w = 0.5       # Inertia weight of previous velocity
        c1 = 1        # Cognitive constant (distance from the the individual's known best position)
        c2 = 2        # Social constant (distance from the the group's known best position)

        # Random numbers to compensate for cognitive and social constants
        for i in range(0, numStartingLocation):
            r1 = random.random()
            r2 = random.random()
            # Updating cognitive velocity and social velocity
            cognitiveVelocity = c1 * r1 * (self.individualBestPosition[i] - self.individualPosition[i])
            socialVelocity = c2 * r2 * (groupsBestPosition[i] - self.individualPosition[i])
            self.individualVelocity[i] = w * self.individualVelocity[i] + cognitiveVelocity + socialVelocity

    # Update the position of each agent based on the new velocity updates
    def positionUpdate(self, bounds):
        for i in range(0, numStartingLocation):
            self.individualPosition[i] = self.individualPosition[i] + self.individualVelocity[i]
            # Compensate maximum position?
            if self.individualPosition[i] > bounds[i][1]:
                self.individualPosition[i] = bounds[i][1]
            # Compensat minimum position?
            if self.individualPosition[i] < bounds[i][0]:
                self.individualPosition[i] = bounds[i][0]
```

Exercise 2

- Given the Code 1, what would be the expected minimum, and what would be the corresponding x?

Code 3. Particle Swarm Optimization.

```
64 def particleSwarmOptimization(costFunction, x0, bounds, num_particles, maxiter):
65     # Initialize the swarm
66     global numStartingLocation
67     numStartingLocation = len(x0)
68     print(numStartingLocation)
69     bestGroupError = -1
70     groupsBestPosition = []
71     swarm = []
72 
73     for i in range(0, num_particles):
74         swarm.append(individualParticle(x0))
75 
76     # Loop to start the optimization process
77     for i in range(0, maxiter):
78         # Evaluate each agent's fitness
79         for j in range(0, num_particles):
80             swarm[j].evaluate(costFunction)
81 
82             # Which agent has the best position (minimum error) in the swarm?
83             if swarm[j].individualError < bestGroupError or bestGroupError == -1:
84                 groupsBestPosition = list(swarm[j].individualPosition)
85                 bestGroupError = float(swarm[j].individualError)
86 
87         # Update velocities and positions inside the swarm
88         for j in range(0, num_particles):
89             swarm[j].update_velocity(groupsBestPosition)
90             swarm[j].positionUpdate(bounds)
91 
92     # Print the final results with error
93     print('After running the swarm of computer agents, the group\'s best porision is ' +
94        str(groupsBestPosition) + " with an error of " + str(bestGroupError))
```

Initialization of the Optimization using Swarm Particles

- We initialize the whole process with starting locations and bounds for the solution space in the code 4

Code 4. Initial Parameters

```
initialStartingPosition = [-15, 15]
minMaxBounds = [(-100, 100), (-100, 100)]
particleSwarmOptimization(errorFunction, initialStartingPosition, minMaxBounds,
                          num_particles = 150, maxiter = 30)
```

Exercise 3

- Let's look at a slightly more complex problem to solve (optimize). Consider Equation 2 and think of why particles with low velocity have difficulties finding the global minimum even after an increased number of iterations. What ways can you find to change the code to come to a better solution? Explain how.

$$g(x) = 50\sin(x) + x^2 + 100 \qquad (2)$$

Exercise 4

- How about Function 3? What ways can you find to change the code to come to a better solution? Explain how.
- We have to think about the possible solution space, properties of the graph and how we can reflect the parameters in our code.

$$h(x) = 15 \sin\left(\frac{x}{5}\right)^2 \cdot 75 \cos\left(\frac{x}{13}\right) \cdot 3 \sin\left(\frac{x}{6}\right) + x^2 + 300 \qquad (3)$$

Homework

- Due next Wednesday (17:00, 22nd, Dec. 2021) electronically to manaba+R
- File format: YourStudentID_W12.* (ID without hyphen, e.g., 12345678901_W12.*).
- Your code must include your own comments for all code sections. Go line-by-line. Comments in your program must be full sentences and reflect your understanding of the code.

Q1. Solve the exercises 1-4 given above. Implement Functions 2 and 3 as Python code. (Use the implementation of Function 1 as reference)

- Submit the complete Swarm Intelligence Python code for Functions 1, 2 and 3 respectively (the name of the code could be YourStudentID_W12_Q1_F1.py, YourStudentID_W12_Q1_F2.py, YourStudentID_W12_Q1_F3.py)
- Answer the question of exercises 2-4 using the comments at the end of the submitted source code.

2. Plot all three functions (f(x), g(x), and h(x) in equations (1), (2) and (3)) using Python's matplotlib.

- Submit the Python code which is used for plotting the graph. (→YourStudentID_W12_Q2.py)

Systems Biology – Exercises

Week 13:
Artificial Immune System