Hongwei Hu 001677683

Class: INFO 6205

Final: Particle swarm optimization

Date: April 28, 2017

PSO: Travelling in America (NP Problem)

1. Problem statement: I used PSO to calculate the shortest route while driving in America to visit all states (except Alaska(AK) and Hawaii(HI)). The start point and end point are both Boston.
2. Solution:
   1. Use one particle to represent a route.
   2. Store best route of one particle in exploration as pBest.
   3. Store best route of the whole swarm in exploration as gBest.
   4. While exploring (learning): similar to Genetic Algorithm
      1. Inertia: keep (w)% of the route unchanged: add self-adjustment process to decide which part to remain (Cognition Part 1).
      2. Cognition Part 2: learn from its experience: learn from pBest: pick up (c1)% of pBest to substitute original route in same indexes.
      3. Social: learn from social experience: learn from gBest: pick up (c2)% of gBest to substitute original route in same indexes.
3. Methodology:
   1. State information: Use the coordinates of the capital cities to represent states.
   2. Create classes for Objects:
      1. Package: pso.basic:
         1. City:
            1. Stores all information for a city (name, state, longitude, latitude, etc.)
            2. Stores the distances from this city to all other cities.
         2. Cities:
            1. Stores all City objects in an Array: the index0 is for both the start point and end point of the travel:
            2. Updates distances between any two cities.
         3. Route:
            1. Randomly generates a route: start and end points are both Boston: stores all cities (not include Boston) in the route to an ArrayList.
            2. Calculates the total distance of current route: stores in totalDistance.
            3. Supports calculating distance between any two cities in a route.
      2. Package pso.core:
         1. Particle:
            1. Extends Route: add a new attribute: pBest to store the best route of this Particle in exploration.
            2. Supports exploring (explore()) to find the shortest route:

learnFromSelf(w)

learnFromExperience(pBest, c1)

learnFromSocial(gBest, c2)

* + - 1. Swarm:
         1. Stores All Particles in an ArrayList.
         2. Add gBest to store the best route of the whole swarm in exploration.
         3. Parallel explore for each Particle
  1. Exploration (Optimization Process):
     1. Self-adjustment: learnFromSelf(w):
        1. Keep part (w) of Route unchanged.
        2. Find which part is better: pick same number of cities from both front and rear of the route, compare them and keep the better part unchanged.
        3. Shuffle the rest route.
     2. Learn From pBest: learnFromExperience(pBest, c1):
        1. Pick part (c1) from pBest.
        2. Delete same elements from original route.
        3. Insert this piece (c1) of route to original route in same index.
     3. Learn From gBest: learnFromSocial(gBest, c2):
        1. Pick part (c2) from gBest.
        2. Delete same elements from original route.
        3. Insert this piece (c2) of route to original route in same index.
     4. If learning result is worse than (valMax \* currentRoute), give up learning result.
  2. Parallel Exploration (Swarm.java):

Use swarm.parallelStream().forEach() to parallel explore the shortest route in each Epoch.

* 1. Converge Step (Particle.java):

Use valMax to limit learning process to get all particles converged.

* 1. UI (Package pso.graph):
     1. Supports set all parameters for swarms.
     2. Only shows the final shortest route in Map.

1. Leaning Parameters (w, c1, c2, valMax, #Particles, #Epochs):

Choice of (w, c1, c2, valMax) should be based on size of problems:

* 1. For 10 cities: 0.1 – 0.3 – 0.4(0.3)
  2. For 48 cities (valMax: 1.05):
     1. 0.2 – 0.8 – 0.05
     2. 0.2 – 0.7 – 0.1
     3. 0.5 – 0.7 – 0.1
     4. 0.2 – 0.95 – 0.05
  3. The parameter combinations will significantly affect results of PSO.

1. Result:

Best Route: refer to “Picture/5-Shortest-X”

1. Challenges:

I design the solution based on PSO and GA. The result looks good, but there are some flaws:

* 1. Parallel: Collections.shuffle() will degrade the performance of parallelStream(): when w is low (shuffle more), parallel calculation will be hardly improved. However, for a higher w (keep more part unchanged), running time will be reduced significantly by using parallel approach.
  2. Learn from Social: Because the start and end points are both a same city, the route is a circle. Therefore, gBest could start from both sides of the circle. It’s hard for a route starting from one side to learn a gBest starting from the other side. The learning result could be worse. That’s also why for all good parameter combinations, c2 is all a small value.