SCOA_TSP_PSO

April 10, 2020

[1]:
```python
from operator import attrgetter
import random, copy

class Graph:
    def __init__(self, amount_vertices): # (self, int)
        self.edges = {} # dict of edges
        self.vertices = set() # set of vertices
        self.amount_vertices = amount_vertices # amount of vertices


    def addEdge(self, source, destination, cost=0):
            if not self.isEdgeExist(source, destination):
          self.edges[(source, destination)] = cost
          self.vertices.add(source)
          self.vertices.add(destination)

    def isEdgeExist(self, source, destination):
        return True if (source, destination) in self.edges else False

    # return total cost path (path = list of vertex)
    def getCostPath(self, path):
        total_cost = 0
        for i in range(self.amount_vertices - 1):
            total_cost += self.edges[(path[i], path[i+1])]

        # add cost edge
        total_cost += self.edges[(path[self.amount_vertices - 1], path[0])]
        return total_cost

    def getVertices(self):
        return list(self.vertices)

    # get random unique path - return list of lists of paths
    def getRandomPaths(self, max_size, initial_vertice=-1):
        random_paths = []
        list_vertices = list(self.vertices)
```

```python
        if initial_vertice == -1:
            initial_vertice = random.choice(list_vertices)

        list_vertices.remove(initial_vertice)
        list_vertices.insert(0, initial_vertice)

        for i in range(max_size):
            while True:
                list_temp = list_vertices[1:]
                random.shuffle(list_temp) # shuffle vertex
                list_temp.insert(0, initial_vertice)
                if list_temp not in random_paths:
                    random_paths.append(list_temp)
                    break # break the while True loop

        return random_paths


class Particle:
    def __init__(self, solution, cost): # (self, path/list of vertices(list of
↪int), int)
        self.solution = solution # current solution
        self.pbest = solution # best solution (fitness)
        self.cost_current_solution = cost # cost current solution
        self.cost_pbest_solution = cost # cost pbest solution

        # velocity particle = 3 tuple
        self.velocity = []

    # setters and getters
    def setPBest(self, new_pbest):
        self.pbest = new_pbest

    def getPBest(self):
        return self.pbest

    def setVelocity(self, new_velocity): # (sequence of swap operators)
        self.velocity = new_velocity

    def getVelocity(self):
        return self.velocity

    def setCurrentSolution(self, solution):
        self.solution = solution

    def getCurrentSolution(self):
        return self.solution
```

```python
    def setCostPBest(self, cost):
        self.cost_pbest_solution = cost

    def getCostPBest(self):
        return self.cost_pbest_solution

    def setCostCurrentSolution(self, cost):
        self.cost_current_solution = cost

    def getCostCurrentSolution(self):
        return self.cost_current_solution

    # remove semua element di list velocity
    def clearVelocity(self):
        del self.velocity[:]


# PSO Algorithm
class PSO:
    def __init__(self, graph, iterations, size_population, alfa=1, beta=1,␣
↪initial_vertice=-1): # (self, Graph, int, int, float 0-1, float 0-1, int)
        self.graph = graph # the graph
        self.iterations = iterations # max of iterations
        self.size_population = size_population # size population
        self.particles = [] # list of particles
        self.alfa = alfa # probability that all swap operators in swap sequence␣
↪(pbest - x(t-1))
        self.beta = beta # probability that all swap operators in swap sequence␣
↪(gbest - x(t-1))

        # initialized with a group of random particles (solutions)
        solutions = self.graph.getRandomPaths(self.size_population,␣
↪initial_vertice)

        print("\nInitial solution each particle:")
        # creates the particles and initialization of swap sequences in all the␣
↪particles
        for solution in solutions:
            print("{solution} = {cost}".format(solution=solution, cost=graph.
↪getCostPath(solution)))
            # create a new particle
            particle = Particle(solution=solution, cost=graph.
↪getCostPath(solution))
            # add the particle
            self.particles.append(particle)
```

```
    def setGBest(self, new_gbest):
        self.gbest = new_gbest

    def getGBest(self):
        return self.gbest

    # print particles information
    def showParticles(self):
        print("\nParticles:\n")
        for particle in self.particles:
            print(
                "pbest: %s \t|\t cost pbest: %3d \t|\t current solution: %s \t|\t cost current solution: %3d" \
                % (str(particle.getPBest()), particle.getCostPBest(), str(particle.getCurrentSolution()), particle.getCostCurrentSolution())
            )

    def run(self):
        # for each time step (iteration)
        for t in range(self.iterations):
            # updates gbest (best particle of the population)
            self.gbest = min(self.particles, key=attrgetter("cost_pbest_solution"))

            # for each particle in the swarm
            for particle in self.particles:
                particle.clearVelocity()
                temp_velocity = []
                solution_gbest = copy.copy(self.gbest.getPBest()) # gets solution of the gbest
                solution_pbest = particle.getPBest()[:] # copy of the pbest solution
                solution_particle = particle.getCurrentSolution()[:] # copy of the current solution of the particle

                # generates all swap operators to calculate (pbest - x(t-1))
                for i in range(self.graph.amount_vertices):
                    if solution_particle[i] != solution_pbest[i]:
                        # generates swap operator
                        swap_operator = (i, solution_pbest.index(solution_particle[i]), self.alfa)

                        # append swap operator in the list of velocity
                        temp_velocity.append(swap_operator)
```

```
                        # makes the swap
                        temp = solution_pbest[swap_operator[0]]
                        solution_pbest[swap_operator[0]] = solution_pbest[swap_operator[1]]
                        solution_pbest[swap_operator[1]] = temp

                # generates all swap operators to calculate (gbest - x(t-1))
                for i in range(self.graph.amount_vertices):
                    if solution_particle[i] != solution_gbest[i]:
                        # generates swap operator
                        swap_operator = (i, solution_gbest.index(solution_particle[i]), self.beta)

                        # append swap operator in the list of velocity
                        temp_velocity.append(swap_operator)

                        # makes the swap
                        temp = solution_gbest[swap_operator[0]]
                        solution_gbest[swap_operator[0]] = solution_gbest[swap_operator[1]]
                        solution_gbest[swap_operator[1]] = temp

                # updates velocity
                particle.setVelocity(temp_velocity)

                # generates new solution for particle
                for swap_operator in temp_velocity:
                    if random.random() <= swap_operator[2]: # (random.random() generate a random number [0.0, 1.0))
                        # makes the swap
                        temp = solution_particle[swap_operator[0]]
                        solution_particle[swap_operator[0]] = solution_particle[swap_operator[1]]
                        solution_particle[swap_operator[1]] = temp

                particle.setCurrentSolution(solution_particle) # updates the current solution
                cost_current_solution = self.graph.getCostPath(solution_particle)
                particle.setCostCurrentSolution(cost_current_solution) # updates the cost of the current solution

                # check if current solution is pbest solution
                if cost_current_solution < particle.getCostPBest():
                    particle.setPBest(solution_particle)
                    particle.setCostPBest(cost_current_solution)
```

```python
if __name__ == "__main__":

    # manual input
    amount_vertices = int(input("amount of vertices: "))
    graph = Graph(amount_vertices=amount_vertices)
    n = int(amount_vertices*(amount_vertices-1)/2)
    print("Enter edges (source destination cost) as much {n} time:".format(n=n))
    for i in range(n):
        while True:
            src, dest, cost = [int(x) for x in input().split()]
            if not graph.isEdgeExist(src, dest):
                graph.addEdge(src, dest, cost)
                graph.addEdge(dest, src, cost)
                break
            print("Edge already exists! insert another edge.\n")

    initial_vertice = int(input("Enter the initial vertex: "))
    while True:
        if initial_vertice in graph.getVertices():
            break
        print("Vertex doesn't exist! re-input.")
        initial_vertice = int(input("Enter the initial vertex: "))

    iterations = int(input("Enter the maximum iteration: "))
    size_population = int(input("Enter population size: "))
    alfa = float(input("Enter the swap operator probability for (pbest -␣
↪x(t-1)): "))
    beta = float(input("Enter the swap operator probability for (gbest -␣
↪x(t-1)): "))

    pso = PSO(graph, iterations=iterations, size_population=size_population,␣
↪alfa=alfa, beta=beta, initial_vertice=initial_vertice)
    pso.run()
    pso.showParticles()

    # shows the global best particle
    print('\ngbest: %s | cost: %d\n' % (pso.getGBest().getPBest(), pso.
↪getGBest().getCostPBest()))
```

```
amount of vertices: 5
Enter edges (source destination cost) as much 10 time:
0 1 1
0 2 3
0 3 4
0 4 5
```

```
1 2 1
1 3 4
1 4 8
2 3 5
2 4 1
3 4 2
Enter the initial vertex: 0
Enter the maximum iteration: 100
Enter population size: 10
Enter the swap operator probability for (pbest - x(t-1)): 0.9
Enter the swap operator probability for (gbest - x(t-1)): 1

Initial solution each particle:
[0, 1, 3, 4, 2] = 11
[0, 4, 2, 3, 1] = 16
[0, 2, 4, 1, 3] = 20
[0, 2, 3, 1, 4] = 25
[0, 3, 4, 2, 1] = 9
[0, 2, 3, 4, 1] = 19
[0, 1, 4, 3, 2] = 19
[0, 2, 1, 3, 4] = 15
[0, 3, 1, 2, 4] = 15
[0, 3, 2, 4, 1] = 19

Particles:

pbest: [0, 1, 2, 4, 3]  |       cost pbest:   9        |        current
solution: [0, 1, 2, 4, 3]      |        cost current solution:   9
pbest: [0, 1, 2, 4, 3]  |       cost pbest:   9        |        current
solution: [0, 1, 2, 4, 3]      |        cost current solution:   9
pbest: [0, 1, 2, 4, 3]  |       cost pbest:   9        |        current
solution: [0, 1, 2, 4, 3]      |        cost current solution:   9
pbest: [0, 1, 2, 4, 3]  |       cost pbest:   9        |        current
solution: [0, 1, 2, 4, 3]      |        cost current solution:   9
pbest: [0, 3, 4, 2, 1]  |       cost pbest:   9        |        current
solution: [0, 3, 4, 2, 1]      |        cost current solution:   9
pbest: [0, 1, 2, 4, 3]  |       cost pbest:   9        |        current
solution: [0, 1, 2, 4, 3]      |        cost current solution:   9
pbest: [0, 1, 2, 4, 3]  |       cost pbest:   9        |        current
solution: [0, 1, 2, 4, 3]      |        cost current solution:   9
pbest: [0, 3, 4, 2, 1]  |       cost pbest:   9        |        current
solution: [0, 1, 2, 4, 3]      |        cost current solution:   9
pbest: [0, 3, 4, 2, 1]  |       cost pbest:   9        |        current
solution: [0, 3, 4, 2, 1]      |        cost current solution:   9
pbest: [0, 3, 4, 2, 1]  |       cost pbest:   9        |        current
solution: [0, 3, 4, 2, 1]      |        cost current solution:   9

gbest: [0, 1, 2, 4, 3] | cost: 9
```

```
[ ]:
```