Implement Simulated Annealing

Jacob Strong

I tried to answer how is run time affected as the number of cities grows? Also how was output quality affected.

I ran six experiments each with 20 trials. Each trial tracked the initial distance, the final distance and the number of iterations.

I found that increasing time was roughly linear. With 10 cities taking less than 10 seconds, 100 cities taking roughly a minute, and 1000 cities taking several minutes. I think that this would hold true unless your temperature reduction became dependent on the number of cities.

Because for each set we do not know the best distance, I measured the improvement from the initial randomized route and the final optomized route. With smaller numbers of cities, the improvement was always higher. I imagine that this is because my temperature function had nothing to do with the number of cities. If the temperature were changed to be dependent on the number of cities, I am sure that could be changed.

There were some problems with my algorithm. There were cases where the final result would be greater that the initial result. I can only imagine this is due to probability and bad solutions getting accepted. I did make a change to the way my temperature was calculated and found that in almost all cases it eliminated this problem.

Below I have pasted my code as well as my results.

```python
'''
Code from http://www.theprojectspot.com/tutorial-post/simulated-annealing-
algorithm-for-beginners/6
Translated from java to python
'''

import numpy as np
import random
import copy
import matplotlib.pyplot as plt
import matplotlib.animation as animation

class City:
    x = 0
    y = 0

    def __init__(self, size):
        self.x = random.randint(1, size + 1)
        self.y = random.randint(1, size + 1)

    def getX(self):
        return self.x

    def getY(self):
        return self.y

    def distanceTo(self, city):
        xDist = np.abs(self.getX() - city.getX())
        yDist = np.abs(self.getY() - city.getY())
        return np.sqrt( xDist*xDist +  yDist*yDist)

    def __repr__(self):
        return "(" + str(self.getX()) + ", " + str(self.getY()) + ")"

class Map:
    cities = []

    def __init__(self, numCities, size):
        self.cities = []
        for x in range(numCities):
            self.addCity(size)

    # def __init__(self):

    def addCities(self, numCities, size):
```

```python
        for x in range(numCities):
            self.addCity(size)

    def addCity(self, size):
        self.cities.append(City(size))

    def printCities(self):
        # for x in self.cities:
        #     print('x: ' + str(x.getX()) + ' ' + str(x.getY()))
        print(self.cities)

    def getCity(self, index):
        return self.cities[index]

    def numberOfCities(self):
        return len(self.cities)


class Tour:
    tour = []
    distance = 0

    def __init__(self, tour = []):
        self.tour = tour

    def getTour(self):
        return self.tour

    def generateIndividual(self, map):
        self.tour = []
        for cityIndex in range(map.numberOfCities()):
            self.tour.append(map.getCity(cityIndex))
        random.shuffle(self.tour)

    def getCity(self, position):
        return self.tour[position]

    def setCity(self, position, city):
        self.tour[position] = city
        self.distance = 0

    def getDistance(self):
        if self.distance == 0:
            tourDistance = 0
```

```
            for x in range(self.tourSize()):
                fromCity = self.getCity(x)
                destCity = None
                if x + 1 < self.tourSize():
                    destCity = self.getCity(x + 1)
                else:
                    destCity = self.getCity(0)
                tourDistance += fromCity.distanceTo(destCity)
            self.distance = tourDistance
        return self.distance

    def tourSize(self):
        return len(self.tour)

    def __repr__(self):
        ret = "|"
        for x in range(self.tourSize()):
            ret += str(self.getCity(x)) + "|"
        return ret

def acceptanceProbability(energy, newEnergy, temperature):
    if newEnergy < energy:
        return 1.0
    return np.exp((float(energy) - float(newEnergy)) / float(temperature))

def plot(tour, size, name):
    x = []
    y = []

    for i in range(tour.tourSize()):
        city = tour.getCity(i)
        x.append(city.getX())
        y.append(city.getY())
    plt.plot(x, y)
    plt.xlabel('East - West')
    plt.ylabel('North - South')
    plt.title('Map of current best route')
    plt.savefig(name)
    plt.clf()


def trial():
    initTemp = 1
    temp = initTemp
    coolingRate = 0.0003
```

```python
    size = 200
    map = Map(10, size)

    currentSolution = Tour()
    currentSolution.generateIndividual(map)
    print(str(currentSolution.getDistance()) + ', ', end='')

    best = currentSolution

    iterations = 0
    plot(best, size, 'before.png')

    while temp > 1:
        iterations += 1
        newSolution = Tour(currentSolution.getTour())

        pos1 = random.randint(0, newSolution.tourSize() - 1)
        pos2 = random.randint(0, newSolution.tourSize() - 1)
        city1 = newSolution.getCity(pos1)
        city2 = newSolution.getCity(pos2)

        newSolution.setCity(pos1, city2)
        newSolution.setCity(pos2, city1)

        currentEnergy = currentSolution.getDistance()
        newEnergy = newSolution.getDistance()

        if acceptanceProbability(currentEnergy, newEnergy, temp) > random.random():
            currentSolution = Tour(newSolution.getTour())

        if currentSolution.getDistance() < best.getDistance():
            best = Tour(currentSolution.getTour())

        temp *= 1-coolingRate
        # temp = -coolingRate*(iterations*iterations) + initTemp

    plot(best, size, 'after.png')
    print(str(best.getDistance()) + ', ', end = '')
    # print(best)
    print(str(iterations) + ', ')

for x in range(20):
    print(  str(x) + ', ', end = '')
```

```
    trial()
```

Initial temp = 100000 Cooling Rate = 0.003

Trial #	Initial Dist	Final Dist	Itterations	10 - 20x20	
0	137.0535	75.83036	3066		0
1	147.0714	92.65154	3066		0
2	166.6881	93.3179	3066		0
3	132.6423	80.95791	3066		0
4	102.7675	58.78903	3066		0
5	122.0251	78.43806	3066		0
6	154.6623	66.55147	3066		0
7	109.2701	74.27045	3066		0
8	87.52026	58.67221	3066		0
9	125.4761	115.6428	3066		0
10	75.8186	64.05082	3066		0
11	80.46045	58.88709	3066		0
12	115.2806	112.6159	3066		0
13	114.5069	68.24768	3066		0
14	91.18816	90.08593	3066		0
15	110.9848	121.0761	3066		1
16	113.1181	124.2781	3066		1
17	83.6736	103.7608	3066		1
18	81.97641	59.86077	3066		0
19	103.319	124.1508	3066		1

Initial Avg	Final Avg	Improve	Ratio Imp	Num Worse	
112.7752	86.10678	26.66838	0.236474		4

Trial #	Initial Dist	Final Dist	Itterations	100 - 200x200	
0	10483.85	10438.14	3066		0
1	9873.428	9736.356	3066		0
2	11543.54	9810.08	3066		0
3	10005.15	10404.96	3066		1
4	10664.83	9031.654	3066		0
5	10452.93	10631.19	3066		1
6	10032.42	10821.64	3066		1
7	10846.74	8797.663	3066		0
8	10578.64	8815.68	3066		0
9	10045.07	9303.17	3066		0
10	10190.58	10749.82	3066		1
11	11031.47	10189.14	3066		0
12	10628.89	8920.259	3066		0
13	10484.55	9126.311	3066		0

14	10185.72	10388.34	3066		1
15	10002.41	8789.53	3066		0
16	9405.808	10433.27	3066		1
17	10681.16	8622.093	3066		0
18	10462.86	9091.786	3066		0
19	9816.218	9172.584	3066		0

Initial Avg	Final Avg	Improve	Ratio Imp	Num Worse	
10370.81	9663.683	707.13	0.068185		6

Trial #	Initial Dist	Final Dist	Itterations	1000 - 1000x1000	
0	528685.5	503310.6	3066		0
1	517473.7	499792.8	3066		0
2	529261.9	517923.9	3066		0
3	528401.3	518936.6	3066		0
4	521049	514085.2	3066		0
5	522149.4	511690.8	3066		0
6	518586.2	525435.6	3066		1
7	518394.3	496741.5	3066		0
8	531700.6	497853.6	3066		0
9	529958.9	517901.4	3066		0
10	532131.6	510992.7	3066		0
11	514492.3	512092.5	3066		0
12	509501.2	494876.1	3066		0
13	513576.3	505124.9	3066		0
14	518465.2	495255.3	3066		0
15	523874.8	510672.5	3066		0
16	529145	516428	3066		0
17	516810.3	509884	3066		0
18	507649.2	501975.9	3066		0
19	517500.5	502511	3066		0

Initial Avg	Final Avg	Improve	Ratio Imp	Num Worse	
521440.4	508174.2	13266.12	0.025441		1

Using temp = -coolingRate*(iterations*iterations) + initTemp

Trial #	Initial Dist	Final Dist	Itterations	10 - 20x20	
0	85.12906	75.0199	1000		0
1	83.88016	84.51531	1000		1
2	118.7738	84.58254	1000		0

3	118.073	79.6977	1000	0
4	108.6803	77.78949	1000	0
5	93.8304	89.85523	1000	0
6	88.3042	67.01431	1000	0
7	110.6068	76.69462	1000	0
8	144.2773	86.44993	1000	0
9	107.1015	70.37247	1000	0
10	118.7708	114.1922	1000	0
11	90.26841	93.01604	1000	1
12	98.50196	66.55267	1000	0
13	91.08195	69.39208	1000	0
14	98.52461	75.8234	1000	0
15	124.9679	89.25145	1000	0
16	104.6113	78.06612	1000	0
17	98.24687	84.20958	1000	0
18	108.2952	78.72468	1000	0
19	152.6871	102.1988	1000	0

Initial Avg	Final Avg	Improve	Ratio Imp	Num Worse
107.2306	82.17093	13266.12	0.233699	2

Trial #	Initial Dist	Final Dist	Itterations	100 - 200x200
0	11218.63	9196.376	1000	0
1	10473.26	9215.985	1000	0
2	10189.41	9786.626	1000	0
3	10953.03	9185.291	1000	0
4	10013.58	9297.73	1000	0
5	10324.12	9594.264	1000	0
6	10551.64	9691.848	1000	0
7	10836	9861.827	1000	0
8	10430.41	9724.661	1000	0
9	11462.64	9489.855	1000	0
10	10034.46	8662.033	1000	0
11	9641.074	8962.279	1000	0
12	10883.71	10297.5	1000	0
13	9841.282	9262.769	1000	0
14	10505.92	9474.226	1000	0
15	10746.07	9353.547	1000	0
16	10627.9	9406.761	1000	0
17	9968.031	8892.878	1000	0
18	10047.15	9240.414	1000	0
19	9903.5	9197.769	1000	0

Initial Avg	Final Avg	Improve	Ratio Imp	Num Worse	
10432.59	9389.732	13266.12	0.099962		0

Trial #	Initial Dist	Final Dist	Itterations	1000 - 1000x1000	
0	519331.5	503532.3	1000		0
1	523581.1	506842.2	1000		0
2	527102.7	513642.4	1000		0
3	518056.3	508358.4	1000		0
4	538488.6	519496.5	1000		0
5	518237	515767	1000		0
6	519166.8	502912.3	1000		0
7	530807.3	523208.1	1000		0
8	528162.3	514657.6	1000		0
9	507671	500750.7	1000		0
10	530150.1	503112.8	1000		0
11	511295.2	499181.2	1000		0
12	517241.1	504185.3	1000		0
13	531469.3	515374.7	1000		0
14	513686.8	509429.8	1000		0
15	527785.4	516636.4	1000		0
16	520957	512789.4	1000		0
17	523464.5	514463.4	1000		0
18	490028.1	488947.7	1000		0
19	530262.1	512599	1000		0

Initial Avg	Final Avg	Improve	Ratio Imp	Num Worse	
521347.2	509294.4	13266.12	0.023119		0