TSP - Genetic Algorithm

June 12, 2021

1 TSP with GA

1.1 Data Preparation

[1]:
```
def read_file(file_name):
    text_file = open(file_name, 'r')
    data = [[int(n) for n in line.split()] for line in text_file]
    return data
```

1.2 Prepare compare approach (Greedy Algorithm)

[2]:
```
def find_min(graph, node, path):
    min_cost = int(1e9)
    return_node = node
    for i in range (len(graph[node - 1])):
        if graph[node - 1][i] != 0 and i + 1 not in path:
            if min_cost > graph[node - 1][i]:
                min_cost = graph[node - 1][i]
                return_node = i + 1
    return (return_node, min_cost)
```

[3]:
```
def tsp_greedy(graph, start):
    path = [start]
    now = start
    cost = 0
    while(len(path) != len(graph)):
        future, future_cost = find_min(graph, now, path)
        cost += future_cost
        path.append(future)
        now = future
    cost += graph[now - 1][start - 1]
    path.append(start)
    print(path)
    return cost
```

1.2.1 Test case for tsp_greedy

[4]:
```
data = read_file('/Users/sangryupark/Downloads/dataset/gr17_d.txt')
print(tsp_greedy(data, 1))
```

```
[1, 13, 4, 7, 8, 6, 17, 14, 15, 3, 11, 5, 10, 2, 9, 12, 16, 1]
2187
```

1.3 Genetic Algorithm

[5]:
```
import random
```

[6]:
```
## Create genes for population
def create_gene(graph, start):
    gene = [start]
    while len(gene) != len(graph):
        candidate = random.randint(1, len(graph))
        if candidate not in gene:
            gene.append(candidate)
    gene.append(start)
    return gene
```

[7]:
```
## Test create_gene
a = [[0,3,4,2,7], [3,0,4,6,3], [4,4,0,5,8], [2,6,5,0,6], [7,3,8,6,0]]
print(create_gene(a, 1))
```

```
[1, 2, 5, 3, 4, 1]
```

[8]:
```
## Calculate cost for path
def calculate_cost(graph, path):
    cost = 0
    for i in range(len(path) - 1):
        cost += graph[path[i] - 1][path[i + 1] - 1]
    return cost
```

[9]:
```
## Test for calculate_cost / Expected return : 80
a = [[0, 10, 15, 20], [10, 0, 35, 25], [15, 35, 0, 30], [20, 25, 30, 0]]
test_path = [1,2,4,3,1]
print(calculate_cost(a, test_path))
```

```
80
```

[10]:
```
## Select save_rate amount of population and add new population
def sort_route_select(graph, start, population, save_rate):
    # sort the population
    for i in range(len(population) - 1):
        temp_cost = calculate_cost(graph, population[i])
        for j in range(i + 1, len(population)):
```

```
            if temp_cost > calculate_cost(graph, population[j]):
                temp = population[i]
                population[i] = population[j]
                population[j] = temp

    #Create new population from sorted : Top Ranked population + new population␣
↪from random
    save_pop = int(len(population) * save_rate)
    new_pop = len(population) - save_pop
    new_population = population[:save_pop]

    for l in range(new_pop):
        random_select = random.randint(0, save_pop)
        new_population.append(population[random_select])

    return new_population
```

[11]:
```
## Test for sort_route_select
data = read_file('/Users/sangryupark/Downloads/dataset/gr17_d.txt')
population = []
for i in range(10):
    population.append(create_gene(data, 1))
result = sort_route_select(data, 1, population, 0.2)
for j in range(len(result)):
    print(calculate_cost(data, result[j]))
```

```
4074
3841
4074
3841
4074
4074
3841
4074
3841
4074
```

[12]:
```
## Breed from population
def breed(parent1, parent2, start):
    child = []
    ch1 = parent1[1 : len(parent1) - 1]
    ch2 = parent2[1 : len(parent2) - 1]
    p1 = []
    p2 = []

    a_point = random.randint(0, len(ch1))
    b_point = random.randint(0, len(ch1))
```

```
    while a_point == b_point:
        b_point = random.randint(0, len(ch1) - 1)

    start = min(a_point, b_point)
    end = max(a_point, b_point)

    for i in range(start,end):
        p1.append(ch1[i])

    p2 = [item for item in ch2 if item not in p1]
    child = p1 + p2
    child.insert(0, 1)
    child.append(1)
    return child
```

[13]:
```
## test for breed
p1 = [1,3,2,4,5,1]
p2 = [1,2,3,4,5,1]
print(breed(p1, p2, 1))
```

```
[1, 3, 2, 4, 5, 1]
```

[14]:
```
## breed for population
def breed_population(population, save_rate, start):
    breed_result = []
    pool = random.sample(population, len(population))
    save = int(len(population) * save_rate)
    new = len(population) - save

    for i in range(save):
        breed_result.append(population[i])

    for j in range(new):
        child = breed(pool[j], pool[len(population) - j - 1], start)
        breed_result.append(child)

    return breed_result
```

[15]:
```
## test for breed_population
data = read_file('/Users/sangryupark/Downloads/dataset/gr17_d.txt')
population = []
for i in range (10):
    population.append(create_gene(data, 1))
print(population)
print(breed_population(population, 0.2, 1))
```

```
[[1, 4, 5, 16, 9, 12, 2, 15, 6, 10, 11, 14, 17, 3, 7, 13, 8, 1], [1, 10, 2, 8,
16, 12, 3, 13, 17, 15, 14, 9, 6, 11, 7, 4, 5, 1], [1, 6, 3, 11, 7, 2, 14, 5, 4,
8, 16, 13, 9, 17, 10, 15, 12, 1], [1, 2, 9, 11, 16, 17, 12, 10, 7, 6, 4, 3, 8,
15, 5, 13, 14, 1], [1, 13, 11, 14, 16, 6, 7, 12, 4, 8, 9, 5, 10, 15, 3, 2, 17,
1], [1, 2, 7, 14, 4, 3, 15, 12, 13, 9, 16, 10, 8, 17, 11, 5, 6, 1], [1, 7, 9, 6,
16, 11, 2, 4, 14, 3, 8, 12, 15, 10, 13, 17, 5, 1], [1, 17, 5, 3, 6, 7, 10, 8, 9,
11, 4, 2, 16, 12, 13, 14, 15, 1], [1, 14, 12, 4, 6, 13, 10, 9, 8, 15, 16, 5, 11,
7, 2, 17, 3, 1], [1, 3, 14, 6, 5, 4, 17, 11, 7, 16, 12, 10, 8, 15, 2, 9, 13, 1]]
[[1, 4, 5, 16, 9, 12, 2, 15, 6, 10, 11, 14, 17, 3, 7, 13, 8, 1], [1, 10, 2, 8,
16, 12, 3, 13, 17, 15, 14, 9, 6, 11, 7, 4, 5, 1], [1, 16, 13, 11, 14, 6, 7, 12,
4, 8, 9, 5, 10, 15, 3, 2, 17, 1], [1, 14, 12, 4, 6, 13, 10, 9, 8, 15, 16, 5, 11,
7, 2, 17, 3, 1], [1, 17, 10, 15, 2, 7, 14, 4, 3, 12, 13, 9, 16, 8, 11, 5, 6, 1],
[1, 6, 7, 10, 8, 9, 3, 14, 5, 4, 17, 11, 16, 12, 15, 2, 13, 1], [1, 4, 5, 16, 9,
12, 2, 15, 10, 8, 3, 13, 17, 14, 6, 11, 7, 1], [1, 16, 12, 3, 13, 17, 15, 14, 9,
6, 11, 7, 4, 5, 2, 10, 8, 1], [1, 4, 17, 11, 7, 16, 12, 10, 8, 5, 3, 6, 9, 2,
13, 14, 15, 1], [1, 15, 12, 13, 9, 16, 10, 8, 17, 11, 5, 6, 3, 7, 2, 14, 4, 1]]
```

[16]:
```python
## random mutation
def random_mutation(path):
    r = random.randint(1, len(path) - 2)
    r1 = random.randint(1, len(path) - 2)

    while r == r1:
        r1 = random.randint(1, len(path) - 2)
    temp = path[r]
    path[r] = path[r1]
    path[r1] = temp

    return path
```

[17]:
```python
## test for random_mutation
path = [1,2,3,4,1]
print(random_mutation(path))
```

```
[1, 3, 2, 4, 1]
```

[18]:
```python
## combine all methods to create next generation
def next_generation(graph, population, start, save_rate, save_possible):
    rank_population = sort_route_select(graph, start, population, save_rate)
    children = breed_population(population, save_rate, start)
    next_population = []
    for i in range(len(children)):
        possible = random.randint(0, 100)
        if possible < save_possible:
            mutated = random_mutation(children[i])
            next_population.append(mutated)
        else:
```

```
            next_population.append(children[i])
    return next_population
```

[19]:
```
## test for next_generation
data = read_file('/Users/sangryupark/Downloads/dataset/gr17_d.txt')
population = []
for i in range(3):
    population.append(create_gene(data, 1))
print(population)
print(next_generation(data, population, 1, 0.2, 5))
```

```
[[1, 16, 14, 15, 7, 2, 10, 17, 5, 13, 12, 8, 4, 3, 6, 11, 9, 1], [1, 15, 11, 13,
3, 14, 8, 4, 2, 6, 10, 17, 16, 7, 12, 5, 9, 1], [1, 12, 8, 14, 6, 9, 3, 11, 16,
7, 5, 4, 15, 10, 17, 2, 13, 1]]
[[1, 10, 17, 5, 13, 12, 8, 4, 3, 6, 14, 9, 11, 16, 7, 15, 2, 1], [1, 16, 7, 15,
11, 13, 3, 14, 8, 4, 2, 6, 10, 17, 12, 5, 9, 1], [1, 6, 9, 3, 11, 16, 7, 5, 4,
15, 10, 17, 2, 13, 14, 12, 8, 1]]
```

[20]:
```
def tsp_genetic(graph, start, pop_size, run_time, save_rate, save_possible):
    population = []
    count = 0

    for i in range(pop_size):
        population.append(create_gene(graph, start))

    #print("Initial Population cost : ", calculate_cost(graph, population[0]))

    while count < run_time:
        new_population = next_generation(graph, population, start, save_rate,␣
 ↪save_possible)
        population = new_population
        count += 1

    cost_index = 0
    min_cost = int(1e9)

    for k in range(len(population)):
        cost = calculate_cost(graph, population[k])
        if cost < min_cost:
            cost_index = k
            min_cost = cost

    #print(population[cost_index])
    return min_cost
```

[21]:
```
## test for tsp_genetic
data = read_file('/Users/sangryupark/Downloads/dataset/att48_d.txt')
```

```
print(tsp_genetic(data, 1, 100, 2000, 0.2, 1))
print(tsp_greedy(data, 1))
```

```
44038
[1, 9, 38, 31, 44, 18, 7, 28, 36, 30, 6, 37, 19, 27, 43, 17, 46, 33, 15, 12, 11,
23, 14, 25, 13, 21, 47, 20, 40, 3, 22, 16, 41, 34, 29, 5, 48, 39, 32, 24, 10,
42, 26, 4, 35, 45, 2, 8, 1]
40551
```

1.4 Plot per generation

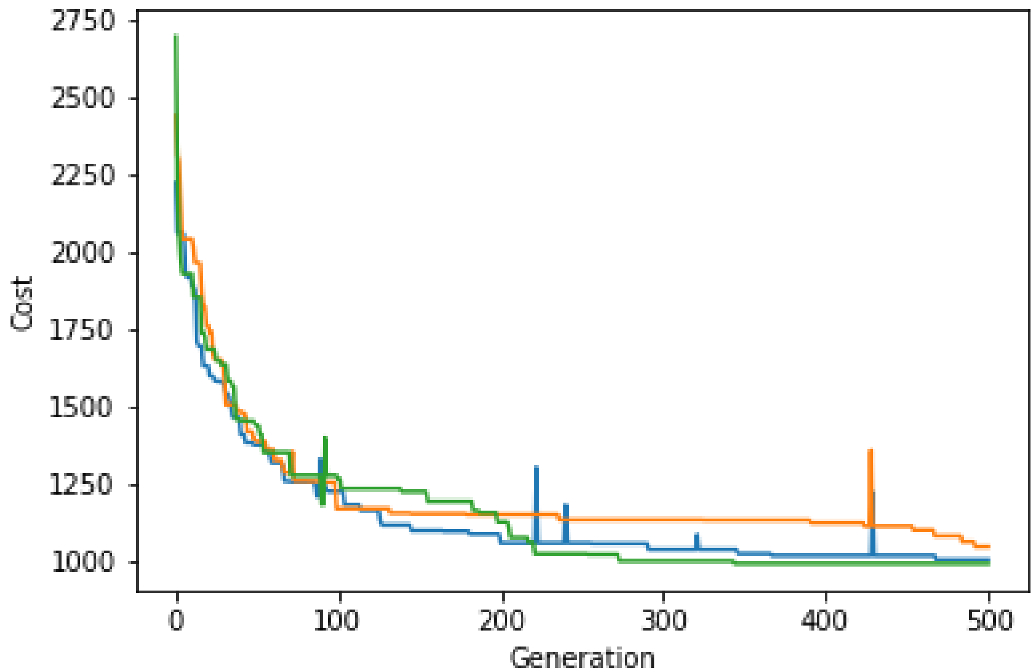
[22]:
```
import matplotlib.pyplot as plt
```

[23]:
```
def sort(graph, population):
    # sort the population
    for i in range(len(population) - 1):
        temp_cost = calculate_cost(graph, population[i])
        for j in range(i + 1, len(population)):
            if temp_cost > calculate_cost(graph, population[j]):
                temp = population[i]
                population[i] = population[j]
                population[j] = temp
```

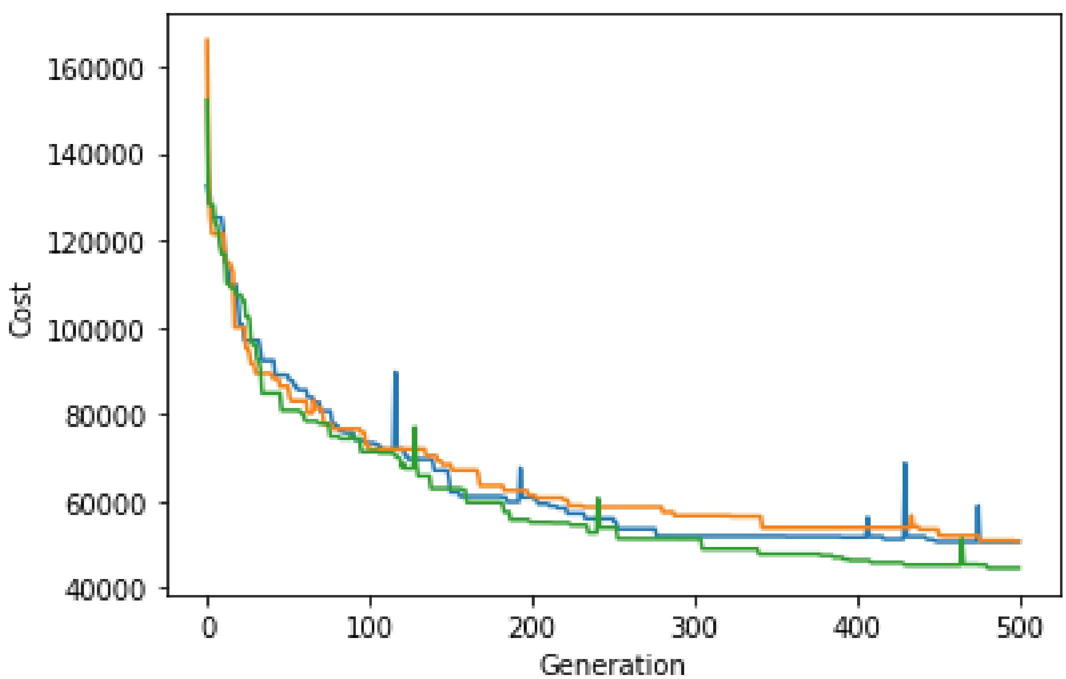
[24]:
```
def plot_tsp_result(graph, start, pop_size, run_time, save_rate, save_possible):
    population = []
    plot_data1 = []
    plot_data2 = []
    plot_data3 = []
    for i in range(pop_size):
        population.append(create_gene(graph, start))
    sort(graph, population)
    plot_data1.append(calculate_cost(graph, population[0]))

    for j in range(run_time):
        population = next_generation(graph, population, start, save_rate,␣
 ↪save_possible)
        sort(graph, population)
        plot_data1.append(calculate_cost(graph, population[0]))


    population2 = []
    for m in range(pop_size):
        population2.append(create_gene(graph, start))

    sort(graph, population2)
    plot_data2.append(calculate_cost(graph, population2[0]))
```

```
    for k in range(run_time):
        population2 = next_generation(graph, population2, start, save_rate,␣
↪save_possible)
        sort(graph, population2)
        plot_data2.append(calculate_cost(graph, population2[0]))


    population3 = []
    for n in range(pop_size):
        population3.append(create_gene(graph, start))

    sort(graph, population3)
    plot_data3.append(calculate_cost(graph, population3[0]))

    for l in range(run_time):
        population3 = next_generation(graph, population3, start, save_rate,␣
↪save_possible)
        sort(graph, population3)
        plot_data3.append(calculate_cost(graph, population3[0]))

    plt.plot(plot_data1)
    plt.plot(plot_data2)
    plt.plot(plot_data3)
    plt.xlabel('Generation')
    plt.ylabel('Cost')
    plt.show()
```

[25]:
```
data = read_file('/Users/sangryupark/Downloads/dataset/fri26_d.txt')
plot_tsp_result(data, 1, 100, 500, 0.2, 1)
```




```
[26]: data2 = read_file('/Users/sangryupark/Downloads/dataset/att48_d.txt')
      plot_tsp_result(data2, 1, 100, 500, 0.2, 1)
```



2 Average GA minimum cost

[39]:
```
import time, math
```

[28]:
```
def average(graph):
    result = 0
    result_list =[]
    start = time.time()
    for i in range(10):
        temp_result = tsp_genetic(graph, 1, 100, 500, 0.2, 1)
        result += temp_result
        result_list.append(temp_result)
    print("time : ", time.time() - start)
    print(result / 10)
    return result_list
```

[29]:
```
data_48 = read_file('/Users/sangryupark/Downloads/dataset/att48_d.txt')
data_42 = read_file('/Users/sangryupark/Downloads/dataset/dantzig42_d.txt')
data_5 = [[0,3,4,2,7], [3,0,4,6,3], [4,4,0,5,8], [2,6,5,0,6], [7,3,8,6,0]]
data_26 = read_file('/Users/sangryupark/Downloads/dataset/fri26_d.txt')
data_17 = read_file('/Users/sangryupark/Downloads/dataset/gr17_d.txt')
```

[30]:
```
result_48 = average(data_48)
```

```
time :  213.80094289779663
52428.6
```

[31]:
```
result_42 = average(data_42)
```

```
time :  182.85994505882263
1077.9
```

[32]:
```
result_5 = average(data_5)
```

```
time :  32.87572407722473
19.0
```

[33]:
```
result_26 = average(data_26)
```

```
time :  118.61300206184387
1083.1
```

[34]:
```
result_17 = average(data_17)
```

```
time :  83.4503059387207
2137.7
```

[41]:
```
def accuracy(result, min_cost):
    accuracy_list = []
    for i in range(len(result)):
        error = math.fabs(result[i] - min_cost) / min_cost
        accurate = (1 - error) * 100
        accuracy_list.append(accurate)
    return accuracy_list
```

[55]:
```
def average_accurate(accuracy) :
    total = 0
    for i in accuracy:
        total += i
    return total / 10
```

[46]:
```
print(result_17)
accuracy_17 = accuracy(result_17, 2085)
print(accuracy_17)
```

```
[2100, 2085, 2085, 2155, 2085, 2272, 2172, 2132, 2148, 2143]
[99.28057553956835, 100.0, 100.0, 96.64268585131894, 100.0, 91.03117505995203,
95.8273381294964, 97.74580335731416, 96.97841726618705, 97.21822541966428]
```

[56]:
```
print(average_accurate(accuracy_17))
```

```
97.47242206235012
```

[50]:
```
accuracy_26 = accuracy([1000, 1099, 1169, 1109, 1085, 1142, 1025, 1137, 1021,␣
 ↪1067], 937)
print(accuracy_26)
```

```
[93.27641408751333, 82.71077908217717, 75.2401280683031, 81.64354322305229,
84.20490928495197, 78.12166488794023, 90.60832443970116, 78.65528281750267,
91.03521878335113, 86.12593383137673]
```

[57]:
```
print(average_accurate(accuracy_26))
```

```
84.16221985058698
```

[51]:
```
accuracy_42 = accuracy([1095, 1163, 1175, 914, 952, 1302, 1137, 1136, 1076,␣
 ↪1072], 699)
print(accuracy_42)
```

```
[43.34763948497854, 33.61945636623748, 31.902718168812594, 69.241773962804,
63.805436337625174, 13.733905579399142, 37.33905579399141, 37.482117310443485,
46.065808297567955, 46.63805436337625]
```

[58]:
```
print(average_accurate(accuracy_42))
```

42.3175965665236

[53]:
```
accuracy_46 = accuracy([51353, 54614, 55962, 47746, 48390, 53720, 48643, 49476,␣
 ↪53001, 52082], 47746)
print(accuracy_46)
```

[92.44544045574499, 85.6155489465086, 82.79227579273656, 100.0,
98.6511959116994, 87.48795710635446, 98.1213085912956, 96.3766598249068,
88.99384241611862, 90.91861098311901]

[59]:
```
print(average_accurate(accuracy_46))
```

92.1402840028484