Assignment1

February 20, 2020

1 IN3050/IN4050 Mandatory Assignment 1: Traveling Salesman Problem

1.1 Rules

Before you begin the exercise, review the rules at this website: https://www.uio.no/english/studies/examinations/compulsory-activities/mn-ifi-mandatory.html (This is an individual assignment. You are not allowed to deliver together or copy/share source-code/answers with others.)

1.2 Delivering

Deadline: *Friday, February 21, 2020*

1.3 What to deliver?

Deliver one single zipped folder (.zip, .tgz or .tar.gz) which includes: * PDF report containing: * Your name and username (!) * Instructions on how to run your program. * Answers to all questions from assignment. * Brief explanation of what you've done. * *Your PDF may be generated by exporting your Jupyter Notebook to PDF, if you have answered all questions in your notebook* * Source code * Source code may be delivered as jupyter notebooks or python files (.py) * The european cities file so the program will run right away. * Any files needed for the group teacher to easily run your program on IFI linux machines.

Important: if you weren't able to finish the assignment, use the PDF report to elaborate on what you've tried and what problems you encountered. Students who have made an effort and attempted all parts of the assignment will get a second chance even if they fail initially. This exercise will be graded PASS/FAIL.

1.4 Introduction

In this exercise, you will attempt to solve an instance of the traveling salesman problem (TSP) using different methods. The goal is to become familiar with evolutionary algorithms and to appreciate their effectiveness on a difficult search problem. You may use whichever programming language you like, but we strongly suggest that you try to use Python, since you will be required to write

the second assignment in Python. You must write your program from scratch (but you may use non-EA-related libraries).

	Barcelona	Belgrade	Berlin	Brussels	Bucharest	Budapest
Barcelona	0	1528.13	1497.61	1062.89	1968.42	1498.79
Belgrade	1528.13	0	999.25	1372.59	447.34	316.41
Berlin	1497.61	999.25	0	651.62	1293.40	1293.40
Brussels	1062.89	1372.59	651.62	0	1769.69	1131.52
Bucharest	1968.42	447.34	1293.40	1769.69	0	639.77
Budapest	1498.79	316.41	1293.40	1131.52	639.77	0

Figure 1: First 6 cities from csv file.

1.5 Problem

The traveling salesman, wishing to disturb the residents of the major cities in some region of the world in the shortest time possible, is faced with the problem of finding the shortest tour among the cities. A tour is a path that starts in one city, visits all of the other cities, and then returns to the starting point. The relevant pieces of information, then, are the cities and the distances between them. In this instance of the TSP, a number of European cities are to be visited. Their relative distances are given in the data file, *european_cities.csv*, found in the zip file with the mandatory assignment.

(You will use permutations to represent tours in your programs. If you use Python, the **itertools** module provides a permutations function that returns successive permutations, this is useful for exhaustive search)

1.6 Exhaustive Search

First, try to solve the problem by inspecting every possible tour. Start by writing a program to find the shortest tour among a subset of the cities (say, **6** of them). Measure the amount of time your program takes. Incrementally add more cities and observe how the time increases.

[62]:
```
import time
import pandas as pd
import itertools as it
import numpy as np
import numpy.random as rnd


# Read distance data to Pandas data frame and convert it to Numpy ndarray
city_distances = pd.read_csv('european_cities.csv', sep=';').to_numpy()


def get_shortest_permutation(cities, weights=city_distances):
    # Variables for storing current shortest path
```

```
    smallest_distance = 9999999
    shortest_permutation = None

    for perm in it.permutations(cities[1:]):
        # Initializing a variable for summing up the distance.
        distance = weights[cities[0], perm[0]] + weights[perm[-1], cities[0]]
        # Summing up the distances
        for i in range(1, len(perm)):
            diff = weights[perm[i-1], perm[i]]
            distance += diff
        # If this is the shortest solution so far
        if distance < smallest_distance:
            # Update values for shortest
            smallest_distance = distance
            shortest_permutation = perm

    return (cities[0], ) + shortest_permutation, smallest_distance
```

[63]:
```
# number of cities
n = 10
city_indices = range(n)

t0 = time.time()
perm, dist = get_shortest_permutation(city_indices, city_distances)
t = time.time() - t0

print("Shortest path: ", perm)
print("Shortest distance: ", dist)
print("Time: ", t)
```

```
Shortest path:  (0, 1, 9, 4, 5, 2, 6, 8, 3, 7)
Shortest distance:  7486.31
Time:  1.1170105934143066
```

What is the shortest tour (i.e., the actual sequence of cities, and its length) among the first 10 cities (that is, the cities starting with B,C,D,H and I)? How long did your program take to find it? Calculate an approximation of how long it would take to perform exhaustive search on all 24 cities?

[64]:
```
# List with different amount of cities to solve for
ns = range(2, 12)
# Create Pandas data frame to store solutions conviniently
columns = ['Shortest path', 'Shortest distance', 'Computation time']
exhaustive_solutions = pd.DataFrame(columns=columns)

for n in ns:
    city_indices = range(n)
```

```
    t0 = time.time()
    perm, dist = get_shortest_permutation(city_indices, city_distances)
    t = time.time() - t0

    exhaustive_solutions.loc[n] = [perm, dist, t]

exhaustive_solutions
```

[64]:

	Shortest path	Shortest distance	Computation time
2	(0, 1)	3056.26	0.000017
3	(0, 1, 2)	4024.99	0.000021
4	(0, 1, 2, 3)	4241.89	0.000025
5	(0, 1, 4, 2, 3)	4983.38	0.000062
6	(0, 1, 4, 5, 2, 3)	5018.81	0.000270
7	(0, 1, 4, 5, 2, 6, 3)	5487.89	0.001771
8	(0, 1, 4, 5, 2, 6, 3, 7)	6667.49	0.013083
9	(0, 1, 4, 5, 2, 6, 8, 3, 7)	6678.55	0.121051
10	(0, 1, 9, 4, 5, 2, 6, 8, 3, 7)	7486.31	1.132358
11	(0, 5, 1, 4, 9, 10, 2, 6, 8, 3, 7)	8339.36	12.247979

[76]:
```
# I am assuming a linear relationship between number of permutations and␣
 ↪computation time
x = [np.math.factorial(i) for i in exhaustive_solutions.index.values]
y = exhaustive_solutions['Computation time']
beta = np.polyfit(x, y, 1)

# Extrapolating the model to estimate time for 24 cities with 24! permutations
time_all_cities = beta[0]*np.math.factorial(24) + beta[1]
print("Estimated time for all cities is %.2e years" % (time_all_cities/60/60/
 ↪24))
```

```
Estimated time for all cities is 2.38e+13 years
```

1.7 Hill Climbing

Then, write a simple hill climber to solve the TSP. How well does the hill climber perform, compared to the result from the exhaustive search for the first **10 cities**? Since you are dealing with a stochastic algorithm, you should run the algorithm several times to measure its performance. Report the length of the tour of the best, worst and mean of 20 runs (with random starting tours), as well as the standard deviation of the runs, both with the **10 first cities**, and with all **24 cities**.

[30]:
```
def hill_climb(position, distances, accuracy = 100):
        count = 0
        current_distance = np.sum(distances[position, np.roll(position, 1)])
        while count < accuracy:
            i = rnd.randint(len(position), size=2)
            new_position = position.copy()
```

```
            new_position[i[0]] = position[i[1]]
            new_position[i[1]] = position[i[0]]
            new_distance = np.sum(distances[new_position, np.roll(new_position,␣
↪1)])

            if new_distance < current_distance:
                position = new_position
                current_distance = new_distance
                count = 0
            else:
                count += 1
        return position, current_distance
```

[44]:
```
rnd.seed(3050)
# Number of cities (requires: 2 <= n <= 24)
n = 24
# Number of starting seeds
n_starts = 20
# Pandas data frame for storing solution data
columns = ['Start', 'Solution', 'Distance', 'Comp time']
hill_climb_solutions = pd.DataFrame(columns=columns)

for i in range(n_starts):
    position = np.arange(n)
    rnd.shuffle(position)
    t0 = time.time()
    pos, dist = hill_climb(position, city_distances, 10000)
    t = time.time() - t0
    hill_climb_solutions.loc[i] = [position, pos, dist, t]

hill_climb_solutions.sort_values(by='Distance')
```

[44]:

	Start	\
4	[8, 23, 7, 4, 9, 6, 14, 16, 15, 12, 18, 10, 13…	
12	[3, 0, 21, 19, 22, 16, 18, 14, 9, 15, 13, 11, …	
7	[13, 9, 8, 7, 14, 1, 6, 20, 4, 12, 10, 0, 5, 2…	
14	[21, 0, 12, 19, 23, 18, 16, 8, 11, 5, 14, 13, …	
3	[7, 9, 15, 6, 3, 11, 21, 16, 8, 23, 10, 12, 20…	
8	[1, 16, 20, 3, 14, 18, 5, 22, 4, 8, 7, 23, 10,…	
9	[0, 17, 7, 9, 22, 4, 10, 3, 8, 16, 19, 1, 6, 1…	
0	[16, 10, 23, 14, 17, 8, 9, 1, 21, 12, 11, 6, 1…	
17	[15, 17, 8, 14, 12, 6, 0, 19, 22, 10, 18, 4, 1…	
5	[13, 0, 22, 23, 9, 20, 10, 14, 5, 16, 6, 18, 1…	
11	[22, 9, 16, 1, 10, 0, 19, 4, 3, 2, 15, 20, 8, …	
2	[10, 12, 22, 3, 20, 0, 2, 4, 19, 9, 16, 21, 23…	
6	[10, 22, 4, 11, 12, 18, 14, 9, 8, 15, 6, 13, 2…	
10	[20, 18, 5, 16, 12, 0, 3, 21, 6, 17, 22, 14, 4…	

```
18  [6, 1, 7, 8, 14, 13, 2, 23, 20, 5, 21, 18, 22,…
13  [3, 7, 2, 15, 4, 8, 1, 10, 11, 20, 21, 17, 22,…
1   [21, 23, 5, 2, 0, 3, 20, 13, 15, 19, 10, 22, 1…
19  [13, 5, 8, 2, 4, 21, 14, 3, 23, 19, 0, 9, 22, …
16  [12, 15, 17, 10, 8, 19, 5, 6, 13, 11, 3, 4, 23…
15  [9, 4, 7, 13, 21, 11, 2, 1, 10, 19, 22, 6, 12,…
```

	Solution	Distance	Comp time
4	[19, 21, 6, 8, 3, 11, 7, 16, 12, 0, 18, 13, 15…	12513.66	0.267901
12	[23, 10, 14, 19, 21, 6, 2, 8, 3, 16, 11, 7, 12…	12633.83	0.302105
7	[7, 21, 19, 14, 10, 4, 9, 20, 1, 18, 22, 5, 23…	13438.74	0.269371
14	[17, 15, 13, 18, 0, 12, 16, 3, 11, 7, 19, 14, …	13551.37	0.249314
3	[23, 17, 2, 8, 11, 7, 12, 0, 16, 3, 13, 18, 20…	13757.12	0.253812
8	[16, 3, 8, 6, 21, 19, 14, 10, 23, 2, 11, 7, 12…	13778.21	0.269408
9	[11, 16, 13, 18, 20, 9, 4, 1, 5, 22, 15, 6, 21…	13782.02	0.249848
0	[17, 8, 6, 2, 23, 5, 1, 20, 9, 4, 10, 14, 19, …	13917.67	0.271247
17	[16, 11, 7, 12, 0, 21, 19, 14, 10, 4, 9, 20, 1…	14103.74	0.257933
5	[3, 15, 13, 18, 20, 9, 10, 14, 19, 21, 6, 23, …	14167.94	0.252321
11	[14, 19, 21, 6, 17, 22, 1, 20, 9, 4, 23, 2, 8,…	14412.99	0.245901
2	[21, 7, 11, 16, 12, 0, 13, 15, 3, 8, 6, 2, 5, …	14419.77	0.247504
6	[20, 9, 4, 3, 11, 7, 16, 12, 0, 18, 13, 15, 22…	14425.23	0.277266
10	[22, 15, 13, 18, 0, 12, 16, 3, 6, 21, 19, 14, …	14460.66	0.246494
18	[2, 3, 16, 11, 7, 12, 0, 13, 8, 6, 21, 23, 5, …	14805.97	0.268355
13	[6, 8, 3, 11, 7, 19, 14, 10, 23, 17, 15, 13, 1…	14851.03	0.251374
1	[17, 2, 8, 3, 16, 22, 5, 23, 10, 14, 19, 21, 6…	15164.59	0.261657
19	[12, 13, 15, 17, 22, 5, 23, 6, 21, 19, 14, 10,…	15589.67	0.246457
16	[18, 0, 12, 13, 21, 19, 14, 10, 9, 4, 5, 22, 1…	15611.14	0.257361
15	[18, 13, 16, 11, 3, 8, 2, 14, 19, 21, 6, 7, 12…	16098.98	0.236633

1.8 Genetic Algorithm

Next, write a genetic algorithm (GA) to solve the problem. Choose mutation and crossover operators that are appropriate for the problem (see chapter 4.5 of the Eiben and Smith textbook). Choose three different values for the population size. Define and tune other parameters yourself and make assumptions as necessary (and report them, of course).

For all three variants: As with the hill climber, report best, worst, mean and standard deviation of tour length out of 20 runs of the algorithm (of the best individual of last generation). Also, find and plot the average fitness of the best fit individual in each generation (average across runs), and include a figure with all three curves in the same plot in the report. Conclude which is best in terms of tour length and number of generations of evolution time.

[53]:
```
def evolutionary_alg(cities, pop_size, n_cycles, distances, parent_rate = 1,␣
↪replace_parents=True):
    # No odd numbers please
    assert pop_size % 2 == 0
    n_parents = int(pop_size * parent_rate)
```

```
    n_parents -= n_parents%2
    n_pairs = n_parents // 2

    n_cities = len(cities)

    # Create random starting population
    population = np.empty((pop_size, n), dtype=int)
    for i in range(pop_size):
        population[i] = rnd.permutation(cities)

    # Find distance of all paths in population and total distance
    find_distance = lambda path: np.sum(distances[path, np.roll(path, 1)])
    scores = np.apply_along_axis(find_distance, 1, population)

    for i in range(n_cycles):
        # Choose parents
        inv_scores = np.power(1/scores, 3)
        portions = inv_scores/inv_scores.sum()
        i_parents = rnd.choice(pop_size, size=n_parents,␣
↪replace=replace_parents, p=portions)
        # Get parents from index and reshape to 3D array for easier iteration␣
↪over pairs
        parents = population[i_parents].reshape(n_pairs, 2, n_cities)

        # Get children
        children = create_children_pmx(parents, n_pairs, n_cities)
        # Mutate children
        mutate_children_swap(children)

        # Select new population
        population, scores = select_new_population(population, children,␣
↪scores, distances)
    i_solution = np.argmin(scores)
    return population[i_solution], scores[i_solution]


def select_new_population(population, children, prev_scores, distances,␣
↪elite_size = 3):
    total_population = np.concatenate((population, children))

    find_distance = lambda path: np.sum(distances[path, np.roll(path, 1)])
    children_scores = np.apply_along_axis(find_distance, 1, children)
    scores = np.concatenate((prev_scores, children_scores))

    i_sort = scores.argsort()
    i_elite = i_sort[:elite_size]
    i_not_elite = i_sort[elite_size:]
```

```
    print(scores[i_elite])

    inv_scores = np.power(1/scores[i_not_elite], 3)
    portions = inv_scores/inv_scores.sum()
    i_norm_pop = rnd.choice(i_not_elite, population.shape[0] - elite_size,␣
↪replace=False, p=portions)
    i_pop = np.concatenate((i_norm_pop, i_elite))
    return total_population[i_pop], scores[i_pop]


def mutate_children_swap(children):
    for child in children:
        i = rnd.choice(child.size, 2, replace=False)
        child[i[0]], child[i[1]] = child[i[1]], child[i[0]]


def mutate_children_insert(children):
    raise NotImplementedError


def create_children_pmx(parents, n_pairs, n_cities):
    segment_size = n_cities // 2
    # Starting indices of each segment
    start_segments = rnd.randint(0, n_cities, size=n_pairs)
    # Copy genes from parent 1
    children = parents[:, 1].copy()
    # For each pair of parents
    for i in range(n_pairs):
        # Indices of segment to copy from parent 0, with rollover for out of␣
↪bounds indices
        i_segment = (np.arange(segment_size) + start_segments[i]) % n_cities

        children[i, i_segment] = parents[i, 0, i_segment]

        # For each index in segment
        for j in i_segment:
            # If the replaced value is not in the segment it was replaced by
            if parents[i, 1, j] not in parents[i, 0, i_segment]:
                k = np.where(parents[i, 1] == parents[i, 0, j])[0][0]
                while k in i_segment:
                    k = np.where(parents[i, 1] == parents[i, 0, k])[0][0]
                children[i, k] = parents[i, 1, j]
    return children
```

```
'''
def create_children_pmx_vec(parents, n_pairs, n_cities):
    segment_size = n_cities // 2
    i_segment = np.add(np.mgrid[0:n_pairs, 0:segment_size][1], rnd.randint(0,␣
 ↪n_cities, size=n_pairs).reshape((n_pairs, 1))) % n_cities
    children = parents[:, 1]
    print(children, "\n-")
    print(i_segment, "\n-")
    print(children[:, i_segments], "\n-")
'''

rnd.seed(3151)
# Number of cities
n = 24
cities = np.arange(n)
# Size of population
pop_size = 1000
# Number of generations
n_cycles = 500s


evolutionary_alg(cities, pop_size, n_cycles, city_distances)
```

```
[23703.23 24031.   24237.62]
[23703.23 24031.   24237.62]
[23659.77 23703.23 24031.  ]
[23659.77 23703.23 24031.  ]
[23659.77 23703.23 23961.6 ]
[21532.43 23471.   23611.17]
[21532.43 23032.93 23372.93]
[21532.43 22617.13 23032.93]
[21532.43 22617.13 23032.93]
[21532.43 22092.15 22617.13]
[21532.43 22092.15 22189.01]
[21532.43 22092.15 22189.01]
[21532.43 21962.85 22092.15]
[21532.43 21962.85 22092.15]
[21532.43 21962.85 22092.15]
[21532.43 21962.85 22092.15]
[20642.49 21532.43 21962.85]
[20642.49 21532.43 21962.85]
[20642.49 21330.95 21532.43]
[20642.49 21330.95 21532.43]
[20642.49 21330.95 21532.43]
[20642.49 20850.14 21330.95]
[20642.49 20850.14 21330.95]
[20642.49 20850.14 21330.95]
```

[20642.49 20666.48 20850.14]
[20642.49 20666.48 20850.14]
[19834.15 20642.49 20666.48]
[19834.15 20642.49 20666.48]
[19834.15 20642.49 20666.48]
[19834.15 20642.49 20666.48]
[19834.15 20642.49 20666.48]
[19834.15 20642.49 20666.48]
[19834.15 20642.49 20666.48]
[19834.15 20642.49 20666.48]
[19834.15 20069.37 20642.49]
[19834.15 20069.37 20642.49]
[19834.15 20069.37 20642.49]
[19834.15 20069.37 20642.49]
[19834.15 20069.37 20642.49]
[19834.15 20069.37 20642.49]
[19834.15 20069.37 20627.96]
[19834.15 20069.37 20627.96]
[19834.15 20069.37 20627.96]
[19834.15 20069.37 20627.96]
[19834.15 20069.37 20627.96]
[19834.15 20069.37 20627.96]
[19834.15 20069.37 20627.96]
[19834.15 20069.37 20627.96]
[19834.15 20069.37 20627.96]
[19834.15 20069.37 20111.74]
[19777.11 19834.15 20069.37]
[19043.34 19777.11 19834.15]
[19043.34 19777.11 19834.15]
[19043.34 19777.11 19834.15]
[19043.34 19777.11 19834.15]
[17820.29 19043.34 19777.11]
[17820.29 19043.34 19777.11]
[17820.29 19043.34 19777.11]
[17820.29 19043.34 19777.11]
[17820.29 19043.34 19777.11]
[17820.29 19043.34 19777.11]
[17820.29 19043.34 19777.11]
[17820.29 18713.01 19043.34]
[17820.29 18713.01 19043.34]
[17820.29 18713.01 19043.34]
[17820.29 18713.01 19043.34]
[17820.29 18713.01 19043.34]
[17820.29 18713.01 19043.34]
[17820.29 18713.01 19043.34]
[17820.29 18713.01 19043.34]
[17820.29 18713.01 19043.34]
[17820.29 18713.01 19043.34]

[17820.29 18713.01 19043.34]
[17820.29 18713.01 19043.34]
[17820.29 18713.01 19043.34]
[17820.29 18713.01 19043.34]
[17820.29 18137.47 18713.01]
[17820.29 17985.17 18137.47]
[17389.16 17820.29 17985.17]
[17389.16 17820.29 17985.17]
[17389.16 17820.29 17985.17]
[17389.16 17820.29 17985.17]
[17389.16 17539.75 17820.29]
[17389.16 17539.75 17820.29]
[17389.16 17539.75 17820.29]
[17389.16 17539.75 17820.29]
[17389.16 17539.75 17820.29]
[17389.16 17539.75 17820.29]
[17389.16 17539.75 17820.29]
[16126.98 17389.16 17539.75]
[16126.98 17389.16 17539.75]
[16126.98 17177.67 17389.16]
[16126.98 17092.17 17177.67]
[16126.98 17092.17 17177.67]
[16126.98 17092.17 17177.67]
[16126.98 17092.17 17177.67]
[16126.98 16855.09 17092.17]
[16126.98 16575.85 16855.09]
[16126.98 16575.85 16855.09]
[15480.73 16126.98 16575.85]
[15480.73 16126.98 16575.85]
[15480.73 16126.98 16575.85]
[15480.73 16126.98 16157.41]
[15480.73 16059.78 16126.98]
[15480.73 15662.04 16059.78]
[15480.73 15662.04 16059.78]
[15480.73 15662.04 16059.78]
[15480.73 15662.04 16059.78]
[15480.73 15662.04 15775.88]
[15480.73 15662.04 15775.88]
[15480.73 15662.04 15775.88]
[15480.73 15662.04 15775.88]
[14406.43 15480.73 15662.04]
[14406.43 15480.73 15662.04]
[14406.43 15480.73 15662.04]
[14406.43 15480.73 15662.04]
[14406.43 15480.73 15662.04]
[14406.43 15480.73 15662.04]
[14406.43 14828.83 15480.73]
[14406.43 14828.83 15480.73]

```
[14406.43 14828.83 15346.28]
[14406.43 14828.83 15346.28]
[14406.43 14828.83 15346.28]
[14406.43 14828.83 15346.28]
[14406.43 14828.83 15346.28]
[14406.43 14828.83 15346.28]
[14406.43 14828.83 15346.28]
[14406.43 14828.83 15346.28]
[14406.43 14828.83 15346.28]
[14406.43 14828.83 15346.28]
[12737.2  14406.43 14730.35]
[12737.2  14406.43 14730.35]
[12737.2  14406.43 14730.35]
[12737.2  14406.43 14525.38]
[12737.2  14406.43 14525.38]
[12737.2  14406.43 14525.38]
[12737.2  14406.43 14525.38]
[12737.2  14406.43 14525.38]
[12737.2  14406.43 14525.38]
[12737.2  14406.43 14525.38]
[12737.2  14406.43 14525.38]
[12737.2  14406.43 14525.38]
[12737.2  14406.43 14525.38]
[12737.2  14406.43 14525.38]
[12737.2  14406.43 14525.38]
[12737.2  14406.43 14525.38]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14406.43]
```

```
[12737.2  14343.27 14406.43]
[12737.2  14343.27 14356.82]
[12737.2  14343.27 14356.82]
[12737.2  14343.27 14356.82]
[12737.2  13397.51 14343.27]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13550.91]
[12737.2  13397.51 13512.83]
[12737.2  13397.51 13512.83]
[12737.2  13397.51 13512.83]
[12737.2  13397.51 13512.83]
[12737.2  13230.41 13397.51]
[12737.2  13230.41 13336.21]
[12737.2  13029.76 13230.41]
[12737.2  13029.76 13230.41]
[12737.2  13029.76 13150.21]
[12737.2  13029.76 13150.21]
[12737.2  13029.76 13150.21]
[12737.2  13029.76 13150.21]
[12737.2  13029.76 13150.21]
[12737.2  13029.76 13150.21]
[12737.2  13029.76 13150.21]
[12737.2  13029.76 13150.21]
[12737.2  13000.26 13029.76]
[12737.2  13000.26 13029.76]
[12737.2  12904.11 13000.26]
[12584.66 12737.2  12904.11]
[12584.66 12737.2  12904.11]
[12584.66 12737.2  12904.11]
[12584.66 12737.2  12904.11]
[12584.66 12737.2  12904.11]
[12584.66 12737.2  12885.26]
[12584.66 12666.2  12737.2 ]
[12584.66 12666.2  12737.2 ]
[12584.66 12666.2  12737.2 ]
[12584.66 12666.2  12737.2 ]
[12584.66 12666.2  12737.2 ]
```

```
[12584.66 12666.2  12737.2 ]
[12584.66 12666.2  12737.2 ]
[12584.66 12666.2  12737.2 ]
[12584.66 12666.2  12737.2 ]
[12584.66 12666.2  12737.2 ]
[12584.66 12666.2  12737.2 ]
[12486.8  12584.66 12666.2 ]
[12486.8  12584.66 12666.2 ]
[12486.8  12584.66 12616.24]
[12486.8  12584.66 12616.24]
[12486.8  12584.66 12616.24]
[12486.8  12584.66 12616.24]
[12486.8  12584.66 12616.24]
[12486.8  12584.66 12616.24]
[12486.8  12584.66 12616.24]
[12486.8  12584.66 12616.24]
[12384.22 12486.8  12584.66]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12486.8  12487.51]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
```

```
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12486.8 ]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12416.51]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
```

```
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12402.42 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12402.42]
[12384.22 12384.22 12384.22]
[12384.22 12384.22 12384.22]
[12384.22 12384.22 12384.22]
[12384.22 12384.22 12384.22]
[12384.22 12384.22 12384.22]
[12384.22 12384.22 12384.22]
[12384.22 12384.22 12384.22]
[12384.22 12384.22 12384.22]
```

```
[12384.22 12384.22 12384.22]
[12384.22 12384.22 12384.22]
[12384.22 12384.22 12384.22]
[12384.22 12384.22 12384.22]
[12384.22 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
[12287.07 12384.22 12384.22]
```

```
[12287.07 12384.22 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
[12287.07 12287.07 12384.22]
```

```
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12384.22]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
 [12287.07 12287.07 12287.07]
```

```
[53]: (array([ 8,  3, 16, 11,  7, 12,  0, 18, 13, 15, 17,  2, 23, 22,  5,  1, 20,
              9,  4, 10, 14, 19, 21,  6]), 12287.07)
```

Among the first 10 cities, did your GA find the shortest tour (as found by the exhaustive search)? Did it come close?

For both 10 and 24 cities: How did the running time of your GA compare to that of the exhaustive search?

How many tours were inspected by your GA as compared to by the exhaustive search?

[65]:
```
np.arange(5)
```

[65]:
```
array([0, 1, 2, 3, 4])
```

1.9 Hybrid Algorithm (IN4050 only)

1.9.1 Lamarckian

Lamarck, 1809: Traits acquired in parents' lifetimes can be inherited by offspring. In general the algorithms are referred to as Lamarckian if the result of the local search stage replaces the individual in the population. ### Baldwinian Baldwin effect suggests a mechanism whereby evolutionary progress can be guided towards favourable adaptation without the changes in individual's fitness arising from learning or development being reflected in changed genetic characteristics. In general the algorithms are referred to as Baldwinian if the original member is kept, but has as its fitness the value belonging to the outcome of the local search process.

(See chapter 10 and 10.2.1 from Eiben and Smith textbook for more details. It will also be lectured in Lecure 4)

1.9.2 Task

Implement a hybrid algorithm to solve the TSP: Couple your GA and hill climber by running the hill climber a number of iterations on each individual in the population as part of the evaluation. Test both Lamarckian and Baldwinian learning models and report the results of both variants in the same way as with the pure GA (min, max, mean and standard deviation of the end result and an averaged generational plot). How do the results compare to that of the pure GA, considering the number of evaluations done?

[1]:
```
# Implement algorithm here
```