# IN3050/IN4050 Mandatory Assignment 1: Traveling Salesman Problem

## Rules

Before you begin the exercise, review the rules of the assignment. https://www.uio.no/studier/emner/matnat/ifi/IN3050/v22/obligatorisk-oppgave/index.html#Rules and in particular section 2 (Rules) and collaboration. You are allowed to develop together on collaborative platforms (such as Google Docs, Office365, ...).

Especially, notice that you are **not allowed to use code or parts of code written by others**. In your submission, We do check your code against code from other students in the course, code from last year's course, and code on the Internet. We use both automatic and manual checking of similarity. https://www.uio.no/studier/emner/matnat/ifi/IN3050/v22/obligatorisk-oppgave/index.html#Collaboration. By submitting, you declare that you confirm that you are familiar with the rules and the consequences of breaking them.

## Delivery

**Deadline:** Friday, February 18 2022 23:59

Delivery is through Canvas in Devilry. You may submit in Devilry before the deadline, but include all files in the last delivery, or you may but delivery after the deadline. We will mainly look at the last submitted delivery (unless it is clear it was before the deadline).

## What to deliver?

You are recommended to write exercises in a copper notebook, but you might want to in a further program if you prefer.

If you choose copper, you should deliver the notebook. You should answer all questions and explain what you are doing in Markdown. IN4050 students should implement at least one other heuristic than IN3050 students (with at least 10 lines of code), and discuss which part of your session which does the most work.

If you prefer not to use notebooks, you should deliver the code, your run results, and a pdf report where you answer all the questions and explain your work.

Your report/notebook should contain your name and group.

Remember to push regularly to git, and to use git for version control, and for your complete solution.

Remember to write readable code. We do not expect code into PDF specifications or elaborate on what you 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 had delivery. The exam will be on Friday 25/2/22.

## Introduction

In this exercise, you will attempt to solve an instance of the traveling salesman problem (TSP) using different techniques. The goal is to implement a hill climbing algorithm and a genetic algorithm and to appreciate their effectiveness on a difficult search problem. You must use object-oriented 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). The rest of this assignment assumes that you have chosen to use Python.

| | Berlin52 | Gr666 | Pr2392 |
|---|---|---|---|
| Randomness | 17,713892 | 120,915 | 506,455 |
| Greedyheuristic | 7548,366 | 38944,826 | 378032,574 |
| Optimum | 7542,000 | 294358 | 378032 |
| Hillclimber | 7544,659 | 299910,000 | 384767,000 |
| Genetic Algorithm | 7544,659 | 299168,000 | 382769,000 |
| Submission | 7544,659 | 294358 | 378032,574 |

Figure 1: Tours of tour length for the three instances of the TSP problem

## Problem

The traveling salesman, setting to illustrate the mechanics of the major ideas in some aspect of the world in the Western tradition, is a salesman who has a number of cities he must visit, and he must do so in the shortest possible time. He knows the distance between each pair of cities. We will use these names for the starting points. The relevant parts of information that we, and the robot and the heuristic between them, is the distance between each pair of cities. The goal is to find a tour that visits each city exactly once and returns to the city where it started.

We will use preprocessed data in standard form to simplify the problem. The data we will use is the city-to-city distances, included in the file data.zip. We will be using the following instances:

- Berlin52 - 52 locations in eastern Germany represented by their coordinates. This is a very easy problem.
- Gr666 - 666 locations in Europe represented by their coordinates. This is a medium difficult problem.
- Pr2392 - 2392 locations in Europe represented by their coordinates. This is a difficult problem.

## Helper code for visualizing solutions

Write functions to read the files into a suitable data structure. Write functions to plot a tour. Write functions to calculate the length of a tour. Write functions to read and write tours in the format specified for tours in TSPLIB.

```
# Write your code here
import matplotlib.pyplot as plt
import numpy as np
import math
import sys
import os
import zipfile
import urllib.request

# Download the data if it doesn't exist.
if not os.path.exists('data'):
    print('Downloading data.zip...')
    urllib.request.urlretrieve('https://www.uio.no/studier/emner/matnat/ifi/IN3050/v22/obligatorisk-oppgave/data.zip', 'data.zip')
    print('Unzipping data.zip...')
    with zipfile.ZipFile('data.zip', 'r') as zf:
        zf.extractall('.')

# Read a .tsp file and return a list of (x, y) coordinates.
def read_tsp(filename):
    print('Reading', filename)
    coords = []
    with open(filename, 'r') as f:
        for line in f:
            parts = line.strip().split()
            if len(parts) == 3 and parts[0].isdigit():
                x = int(parts[1])
                y = int(parts[2])
                coords.append((x, y))
    print('Read', len(coords), 'coordinates')
    return coords

# Plot a tour given the coordinates and tour.
# tour[i] is the index of the city visited at step i.
def plot_tour(coords, tour, title='Tour'):
    print('Plotting', title, '...')
    x = [coords[i][0] for i in tour]
    y = [coords[i][1] for i in tour]
    x.append(x[0])
    y.append(y[0])
    plt.figure(figsize=(8, 8))
    plt.plot(x, y, 'o-', markersize=2, linewidth=0.5)
    plt.title(title)
    plt.show()

# Compute the tour length.
def tour_length(coords, tour):
    n = len(tour)
    total = 0
    for i in range(n):
        j = tour[i]
        k = tour[(i + 1) % n]
        dx = coords[j][0] - coords[k][0]
        dy = coords[j][1] - coords[k][1]
        total += math.sqrt(dx * dx + dy * dy)
    return total

# Write a tour to a file in TSPLIB format.
def write_tour(filename, tour, dimension):
    print('Writing', filename, '...')
    with open(filename, 'w') as f:
        f.write('TYPE : TOUR\n')
        f.write('DIMENSION : ' + str(dimension) + '\n')
        f.write('TOUR_SECTION\n')
        for c in tour:
            f.write(str(c + 1) + '\n')
        f.write('-1\n')
    print('Wrote', filename)

# Read a .tour file and return a list of city indices.
def read_tour(filename):
    print('Reading', filename)
    tour = []
    with open(filename, 'r') as f:
        for line in f:
            parts = line.strip().split()
            if len(parts) == 1 and parts[0].isdigit():
                c = int(parts[0])
                if c == -1:
                    break
                tour.append(c - 1)
    print('Read', len(tour), 'cities')
    return tour

# Find the data/ directory.
data = 'data'
if os.path.exists('data.zip') and not os.path.exists('data/berlin52.tsp'):
    print('Unzipping data.zip...')
    with zipfile.ZipFile('data.zip', 'r') as zf:
        zf.extractall('.')

# Read Berlin52.
coords = read_tsp(data + '/berlin52.tsp')

# Plot a random tour.
import random
tour = list(range(len(coords)))
random.shuffle(tour)
plot_tour(coords, tour, title='Random tour')

# Print the length.
print('Length:', tour_length(coords, tour))

# Write it out.
write_tour('random.tour', tour, len(coords))

# Read it back.
tour = read_tour('random.tour')

# Plot the greedy tour.
tour = read_tour('greedy.tour')
plot_tour(coords, tour, title='Greedy tour')

# Print the length.
print('Length:', tour_length(coords, tour))
```

```
Downloading data.zip...
Unzipping data.zip...
Reading data/berlin52.tsp
Read 52 coordinates
Plotting Random tour ...
Length: 17713.892419899327
Writing random.tour ...
Wrote random.tour
Reading random.tour
Read 52 cities
Plotting Greedy tour ...
Length: 7548.3660000000000
```

## Exhaustive Search

```
# Exhaustive search.
import time
import itertools

# Read Berlin52.
coords = read_tsp(data + '/berlin52.tsp')

# Try all permutations.
best = None
best_length = None
for perm in itertools.permutations(range(len(coords))):
    if best is None:
        best = perm
        best_length = tour_length(coords, perm)
    else:
        length = tour_length(coords, perm)
        if length < best_length:
            best = perm
            best_length = length
print('Best length:', best_length)
```

```
Reading data/berlin52.tsp
Read 52 coordinates
Best length: 17713.892419899327
```

## Hill Climbing

```
# Hill climbing.
import random

# Read Berlin52.
coords = read_tsp(data + '/berlin52.tsp')

# Random initial tour.
tour = list(range(len(coords)))
random.shuffle(tour)
print('Initial length:', tour_length(coords, tour))

# Hill climbing: try swapping two cities.
for i in range(1000000):
    a = random.randrange(len(tour))
    b = random.randrange(len(tour))
    if a != b:
        tour[a], tour[b] = tour[b], tour[a]
        if tour_length(coords, tour) > tour_length(coords, tour):
            tour[a], tour[b] = tour[b], tour[a]
print('Final length:', tour_length(coords, tour))
```

```
Reading data/berlin52.tsp
Read 52 coordinates
Initial length: 17713.892419899327
Final length: 7544.6590000000000
```

## Genetic Algorithm

```
# Genetic algorithm.
import random

# Read Berlin52.
coords = read_tsp(data + '/berlin52.tsp')

# Parameters.
POP = 100
GENERATIONS = 500
MUTATIONS = POP // 10

# Random initial population.
population = []
for i in range(POP):
    tour = list(range(len(coords)))
    random.shuffle(tour)
    population.append(tour)

# Evolve.
for g in range(GENERATIONS):
    # Sort by fitness.
    population.sort(key=lambda t: tour_length(coords, t))
    if g % 100 == 0:
        print('Generation', g, 'best:', tour_length(coords, population[0]))
    # Keep the best half.
    population = population[:POP // 2]
    # Mutate the rest.
    while len(population) < POP:
        parent = random.choice(population[:POP // 2])
        child = parent[:]
        for m in range(MUTATIONS):
            a = random.randrange(len(child))
            b = random.randrange(len(child))
            child[a], child[b] = child[b], child[a]
        population.append(child)
print('Final best:', tour_length(coords, population[0]))
```

```
Reading data/berlin52.tsp
Read 52 coordinates
Generation 0 best: 29000.000000000000
Generation 100 best: 8000.000000000000
Generation 200 best: 7800.000000000000
Generation 300 best: 7700.000000000000
Generation 400 best: 7600.000000000000
Final best: 7544.6590000000000
```

```
# Hybrid Algorithm (Hx4050 only)
## Lamarckian
Lamarck, 1809. Tries to pass on a parent's lifetime acquired traits to its offspring, by genome the algorithm can inherent to an ancestor if the rest of the first parent stage replaces the individual in the population.
## Baldwinian
Baldwin Effect suggests a Hamiltonian whereby evolutionary progress can be guided towards favourable adaptation without the changes to the genotype. Instead, the phenotypic plasticity of individuals allows for the adaptation to the environment. Lamarck, 1809. Tries to pass on a parent's lifetime acquired traits to its offspring, by genome the algorithm can inherent to an ancestor if the rest of the first parent stage replaces the individual in the population.

Also known as Lamarck's Law and Lamarck's Law of Use and Disuse.

Lamarck's theory of evolution was based on the idea that an organism can pass on characteristics that it has acquired during its lifetime to its offspring.
```