Homework 1

Rebecca Roskill
rcr2144
MECS 4510
Prof. Hod Lipson
Submitted Sunday, September 19th, 2021
Grace hours used: 0
Grace hours remaining: 96

1 Methods

1.1 Random Search

The random choice function implemented in the Python random library was used to randomly select a starting point and each additional point until all points in the data set were visited. The euclidean distance between the points were visited to calculate the cumulative path traveled. 10000 trials were conducted.

2 Results

2.1 Random Search

Over the course of the 10,000 trials conducted, the following best paths were discovered.

Trial	Best path length
0	∞
1000	470.15
2000	470.15
3000	466.45
4000	466.45
5000	466.45
6000	466.45
7000	466.45
8000	466.45
9000	466.45
10000	466.45

Table 1: Random Search results per 1000 trials

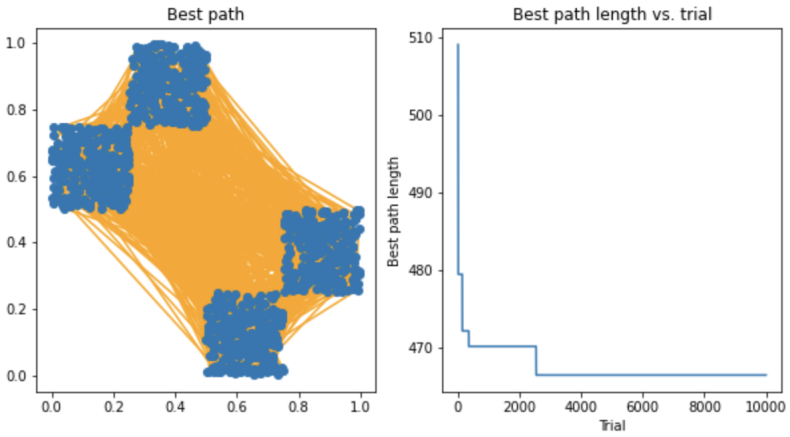

Figure 1: Best path discovered by random search (left) and performance (right).

3 Appendix

3.1 Random Search

```
from google.colab import drive
drive.mount('/content/drive')

# Load dataset from .txt file
with open('/content/drive/My Drive/Senior Year/Evolutionary
    Algos/tsp.txt', 'r') as f:
  lines = f.readlines()

points = [[float(x) for x in line[:-1].split(',')] for line in
    lines]

# Plot dataset
from matplotlib import pyplot as plt

fig = plt.figure(figsize=(10,10))
x = [p[0] for p in points]
y = [p[1] for p in points]
plt.scatter(x, y)

import random
import numpy as np

fig = plt.figure(figsize=(5,5))

'''
Random search algorithm to solve TSP
   points: list of (x,y) coordinates to visit
   fig (optional): mpl figure for plotting, None for no
       plotting
'''
def random_search(points):
  path = []
  points = [p for p in points]
  distance = 0
  p1 = random.choice(points)
  points.remove(p1)
  path += [p1]
  while points:
    p2 = random.choice(points)
    points.remove(p2)
    distance += np.sqrt(np.square(p2[1]-p1[1]) +
        np.square(p2[0]-p1[0]))
    p1 = p2
    path += [p1]
```

```
  return (distance, path)

def plot_path(path, fig):
  if fig:
    x = [p[0] for p in path]
    y = [p[1] for p in path]
    plt.scatter(x, y, zorder=2)
    for i in range(1, len(path)):
      x1, y1 = path[i - 1]
      x2, y2 = path[i]
      plt.plot([x1, x2], [y1, y2], c='orange', zorder=1)

distance, path = random_search(points)
plot_path(path, fig)
distance

# Run n_trials trials
n_trials = 10000

best_dist = [float('inf')]
best_path = None

for i in range(n_trials):
  if i % (n_trials/10) == 0:
    print(i, ' & ', round(best_dist[-1], 2), ' \\\\')
  distance, path = random_search(points)
  if distance < best_dist[-1]:
    best_dist += [distance]
    best_path = path
  else:
    best_dist += [best_dist[-1]]

# Plot best path found in n_trials
fig = plt.figure(figsize=(10,5))
plt.subplot(121)
plot_path(best_path, fig)
plt.title('Best path')
plt.subplot(122)
plt.plot(range(n_trials + 1), best_dist)
plt.title('Best path length vs. trial')
plt.xlabel('Trial')
plt.ylabel('Best path length')
```