Genetic Algorithm for Traveling Salesperson

November 5, 2020

Jacob Hopkins

1 Conventional GA vs Parallel GPU / CPU Accelerated GA

1.1 Imports

[172]:
```
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import random

import math
```

1.2 Reading in from file: usa13509 dataset

[173]:
```
usa13509_raw_data_lst = []
usa13509_raw_data_dict = {'X':[],'Y':[]}

with open('datasets/usa13509/usa13509.tsp', 'r') as f:
    content = f.readlines()

    for data in content[9:-1]:
        data_seperated = data.split()
        usa13509_raw_data_dict['X'].append(float(data_seperated[2]))
        usa13509_raw_data_dict['Y'].append(float(data_seperated[1]))
        usa13509_raw_data_lst.append(␣
 ↪(float(data_seperated[2]),float(data_seperated[1])) )

df = pd.DataFrame(usa13509_raw_data_dict)
```

```
usa13509_optimal_solution = 19982859

print(f'usa13509 Optimal Solution: {usa13509_optimal_solution}')
df
```

```
usa13509 Optimal Solution: 19982859
```

[173]:

	X	Y
0	817827.778	245552.778
1	810905.556	247133.333
2	810188.889	247205.556
3	806280.556	249238.889
4	805152.778	250111.111
…	…	…
13504	1194344.444	489391.667
13505	1224508.333	489466.667
13506	972433.333	489663.889
13507	1227458.333	489938.889
13508	1222636.111	490000.000

[13509 rows x 2 columns]

1.3 Plot usa13509

1.3.1 Setup of style and size for PyPlot

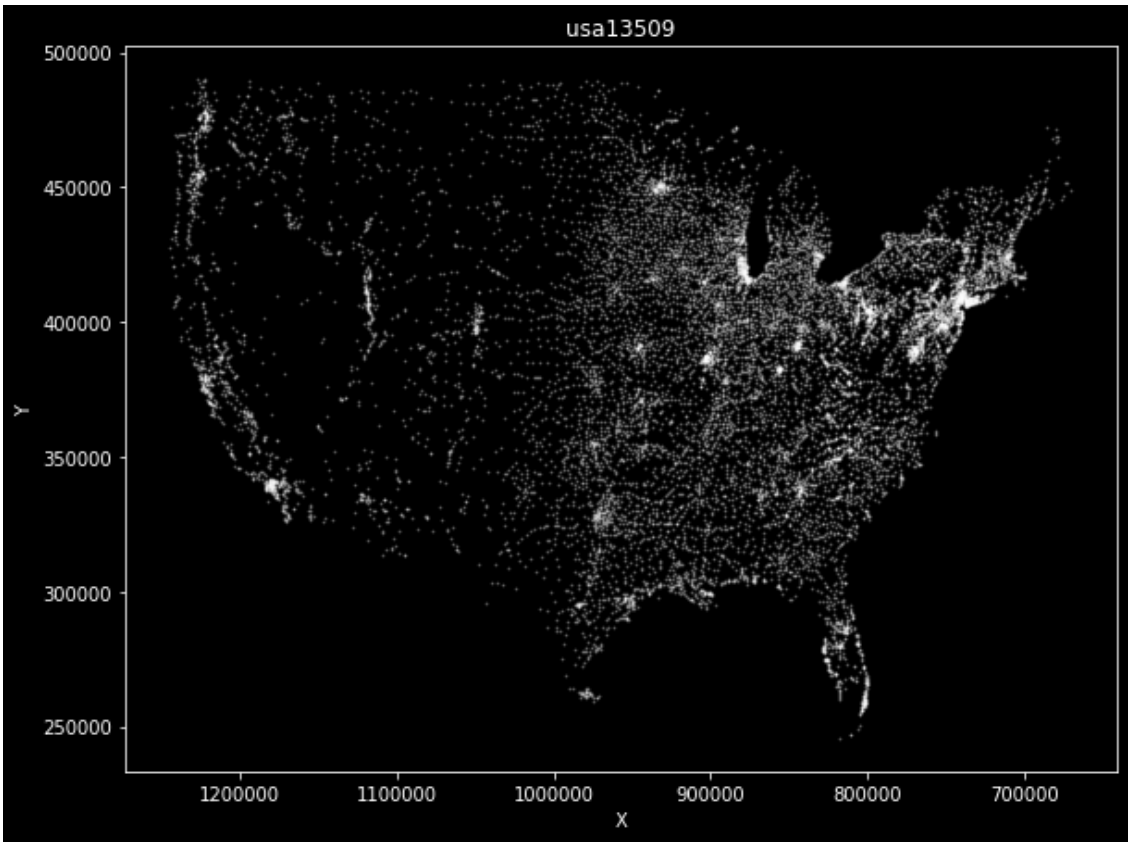
[174]:
```
plt.style.use('dark_background')

plt_scale = 1.5

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 6.4 * plt_scale
fig_size[1] = 4.8 * plt_scale
plt.rcParams["figure.figsize"] = fig_size
```

1.3.2 Plot of the usa13509 dataset

[175]:
```
plt.scatter(usa13509_raw_data_dict['X'],usa13509_raw_data_dict['Y'], alpha=0.
↪25, s=0.8, color='w')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('usa13509')
plt.gca().invert_xaxis()
plt.show()
```



1.4 TSP GA Functions

1.4.1 Random Route Function

[176]:
```
def random_route(data, same_start=False):

    return random.sample(data, len(data))
```

1.4.2 Function to Align Route to Start with Specified Point

[177]:
```
def align_route_starts_with_pt(route_a, pt):

    start = None
    for i,r in enumerate(route):
        if r == pt:
            start = i
    else:
        print('Point was not found in the route.')

    return route[start:] + route[0:start]
```

1.4.3 Function for Euclidean Distance

[178]:
```
def euclidean_distance(a,b):
    return np.sqrt(  ( abs( b[0] - a[0] ) )**2 + ( abs( b[1] - a[1] ) )**2  )
```

1.4.4 Equal Cost Route Distance Function

[179]:
```
def calculate_route_distance(route):

    route_distance = 0

    for i in range(len(route)):

        city_a = route[i]
        city_b = None

        if i + 1 < len(route):
            city_b = route[i + 1]
        else:
            city_b = route[0]

        route_distance += euclidean_distance(city_a,city_b)

    return route_distance
```

1.4.5 Route Fitness Function

[180]:
```
def evaluate_route_fitness(route):
    return 1.0 / calculate_route_distance(route);
```

1.4.6 Function for Initialization of a Population

[181]:
```
def initialize_population(population_size, dataset):
    population = []

    for _ in range(population_size):
        population.append(random_route(dataset))

    return population
```

1.4.7 Function to Sort Routes by Distance

[182]:
```
def rank_routes(population):

    return sorted( population, key = lambda candidate:candidate[1])
```

1.4.8 Selection Function

1.4.9 Crossover Function

1.4.10 Mutation Function

1.4.11 Transition Function

1.5 Conventional Genetic Algorithm

1.6 Parallel Genetic Algorithm

1.7 Test Code:

[183]:
```
POPULATION_SIZE = 100

#initialize population
population = initialize_population(POPULATION_SIZE, usa13509_raw_data_lst)

#Find the fitness and distance of each tour
population_with_distances_and_fitness = []

for candidate in population:
    dst = calculate_route_distance(candidate)
    population_with_distances_and_fitness.append( (candidate, dst, 1/dst) )


# rank the population and show the best and worst

ranked_population_with_distances_and_fitness =␣
 ↪rank_routes(population_with_distances_and_fitness)

print(ranked_population_with_distances_and_fitness[0][2])
print(ranked_population_with_distances_and_fitness[-1][2])
```

```
4.68929195816391e-10
4.596369205158039e-10
```

2 References

2.1 Paper References:

Permutation rules and genetic algorithm to solve the traveling salesman problem:
- https://www.tandfonline.com/doi/full/10.1080/25765299.2019.1615172

An improved genetic algorithm based on k-means clustering for solving traveling salesman problem:
- https://www.researchgate.net/publication/309149000_An_improved_genetic_algorithm_based_on_k-means_clustering_for_solving_traveling_salesman_problem

Parallel Genetic Algorithms with GPU Computing:
- https://www.intechopen.com/books/industry-4-0-impact-on-intelligent-logistics-and-manufacturing/parallel-genetic-algorithms-with-gpu-computing

Solution to travelling salesman problem by clusters and a modified multi-restart iterated local search metaheuristic:
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6104944/

Evolution of a salesman: A complete genetic algorithm tutorial for Python
- https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35

2.2 Database References

2.2.1 MP-TESTDATA - The TSPLIB Symmetric Traveling Salesman Problem Instances

Georg Skorobohatyj - Last update: June 1, 1995 - A large compendium of datasets, and occasionally the optimal tour as well, for tsp
- http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html

2.2.2 The Traveling Salesman Problem

The Traveling Salesman Problem is one of the most intensively studied problems in computational mathematics. These pages are devoted to the history, applications, and current research of this challenge of finding the shortest route visiting each member of a collection of locations and returning to your starting point.
- http://www.math.uwaterloo.ca/tsp/index.html

2.3 Dataset References

2.3.1 ulysses22

Odyssey of Ulysses (Groetschel and Padberg)
- http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/ulysses22.tsp

2.3.2 a280

Drilling problem (Ludwig)
- http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/a280.tsp

2.3.3 pr1002

1002-city problem (Padberg/Rinaldi)
- http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/pr1002.tsp

2.3.4 USA13509

USA13509 is one of the larger TSP instances in TSPLIB. It contains 13,509 cities in the United States (all cities having a population of at least 500 at the time the instance was contributed to TSPLIB).
- http://www.math.uwaterloo.ca/tsp/usa13509/usa13509.html

2.3.5 UK49687

Shortest possible tour to nearly every pub in the United Kingdom
- http://www.math.uwaterloo.ca/tsp/uk/index.html

2.4 Resource References

2.4.1 GPU Programming with Python

- https://linuxhint.com/gpu-programming-python/

2.4.2 Pyplot tutorial: An introduction to the pyplot interface

- https://matplotlib.org/tutorials/introductory/pyplot.html