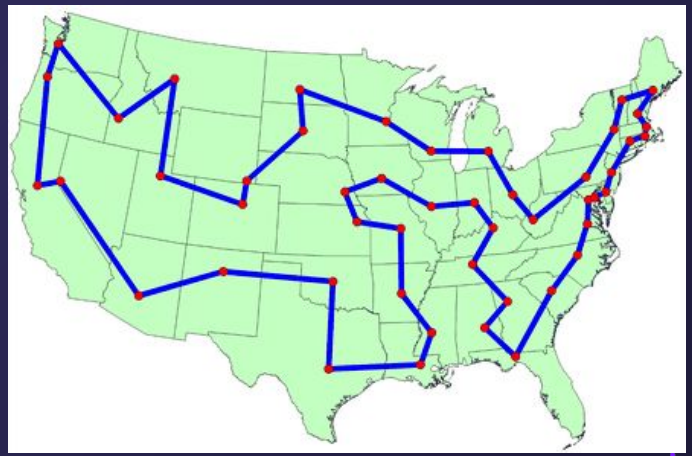
Tugas Individu 2

Travelling Salesman Problem (TSP)
using Genetic Algorithm

Salsabila Fatma Aripa - 5025211057

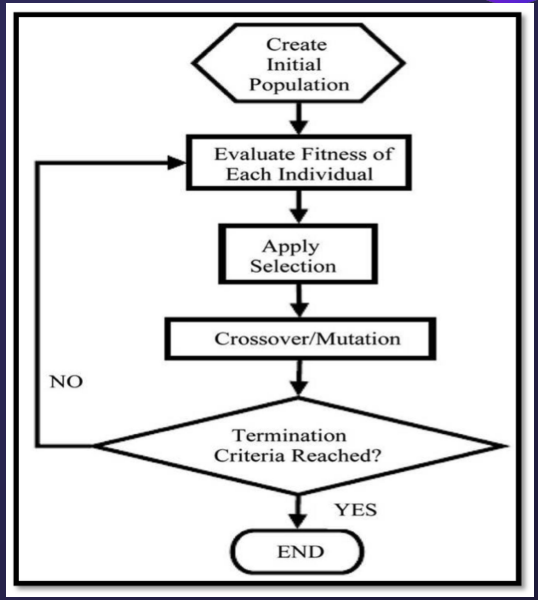
Travelling Salesman Problem (TSP)

Travelling salesman problem (TSP) adalah sebuah permasalahan optimisasi kombinatorial yang mencari jalur terpendek atau tercepat yang melalui beberapa titik (kota) yang harus dikunjungi hanya sekali dan kembali ke titik awal.



Genetic Algorithm (GA)

Genetic Algorithm (GA) adalah teknik optimisasi yang terinspirasi oleh evolusi biologis dan digunakan untuk menemukan solusi yang baik untuk masalah optimisasi. GA adalah algoritma yang berdasarkan pada prinsip seleksi alami, pewarisan sifat, mutasi dan crossover untuk menghasilkan solusi yang semakin baik dari generasi ke generasi.



Genetic Algorithm (GA)

Genetic Algorithm (GA) adalah salah satu metode yang sering digunakan untuk menyelesaikan TSP. GA adalah sebuah algoritma pencarian heuristik yang terinspirasi dari proses evolusi alami dalam seleksi alamiah.

Implementasi

```
CITY_COORDINATES = [[5, 80], [124, 31], [46, 54], [86, 148], [21, 8],
                    [134, 72], [49, 126], [36, 34], [26, 49], [141, 6],
                    [124, 122], [80, 92], [70, 69], [76, 133], [23, 65]]

TOTAL_CHROMOSOME = len(CITY_COORDINATES) - 1

POPULATION_SIZE = 300
MAX_GENERATION = 100
MUTATION_RATE = 0.001
WEAKNESS_THRESHOLD = 900
```

Implementasi

```
class Genome():
    def __init__(self):
        self.chromosome = []
        self.fitness = 0

    def __str__(self):
        return "Chromosome: {0} Fitness: {1}\n".format(self.chromosome,
self.fitness)

    def __repr__(self):
        return str(self)
```

Implementasi

```python
def distance(a, b) -> float:
    dis = math.sqrt(((a[0] - b[0])**2) + ((a[1] - b[1])**2))
    return numpy.round(dis, 2)

def get_fittest_genome (genomes: List[Genome]) -> Genome:
    genome_fitness  = [genome.fitness for genome in genomes]
    return genomes[genome_fitness .index(min(genome_fitness ))]

def eval_chromosome (chromosome: List[int]) -> float:
    arr = [0] * (len(chromosome) + 2)
    arr[1:-1] = chromosome

    fitness = 0
    for i in range(len(arr) - 1):
        p1 = CITY_COORDINATES [arr[i]]
        p2 = CITY_COORDINATES [arr[i + 1]]
        fitness += distance(p1, p2)
    return numpy.round(fitness, 2)

def tournament_selection (population:List[Genome], k:int) -> List[Genome]:
    selected_genomes  = random.sample(population, k)
    selected_parent  = get_fittest_genome (selected_genomes )
    return selected_parent
```

Implementasi

```
def order_crossover(parents: List[Genome]) -> Genome:
    child_chro = [-1] * TOTAL_CHROMOSOME

    subset_length = random.randrange(2, 5)
    crossover_point = random.randrange(0, TOTAL_CHROMOSOME - subset_length)

    child_chro[crossover_point:crossover_point+subset_length] =
parents[0].chromosome[crossover_point:crossover_point+subset_length]

    j, k = crossover_point + subset_length, crossover_point + subset_length
    while -1 in child_chro:
        if parents[1].chromosome[k] not in child_chro:
            child_chro[j] = parents[1].chromosome[k]
            j = j+1 if (j != TOTAL_CHROMOSOME-1) else 0

        k = k+1 if (k != TOTAL_CHROMOSOME-1) else 0

    child = Genome()
    child.chromosome = child_chro
    child.fitness = eval_chromosome(child.chromosome)
    return child
```

Implementasi

```python
def scramble_mutation(genome: Genome) -> Genome:
    subset_length = random.randint(2, 6)
    start_point = random.randint(0, TOTAL_CHROMOSOME - subset_length)
    subset_index = [start_point, start_point + subset_length]

    subset = genome.chromosome[subset_index[0]:subset_index[1]]
    random.shuffle(subset)

    genome.chromosome[subset_index[0]:subset_index[1]] = subset
    genome.fitness = eval_chromosome(genome.chromosome)
    return genome

def reproduction(population: List[Genome]) -> Genome:
    parents = [tournament_selection(population, 20), random.choice(population)]

    child = order_crossover(parents)

    if random.random() < MUTATION_RATE:
        scramble_mutation(child)

    return child
```

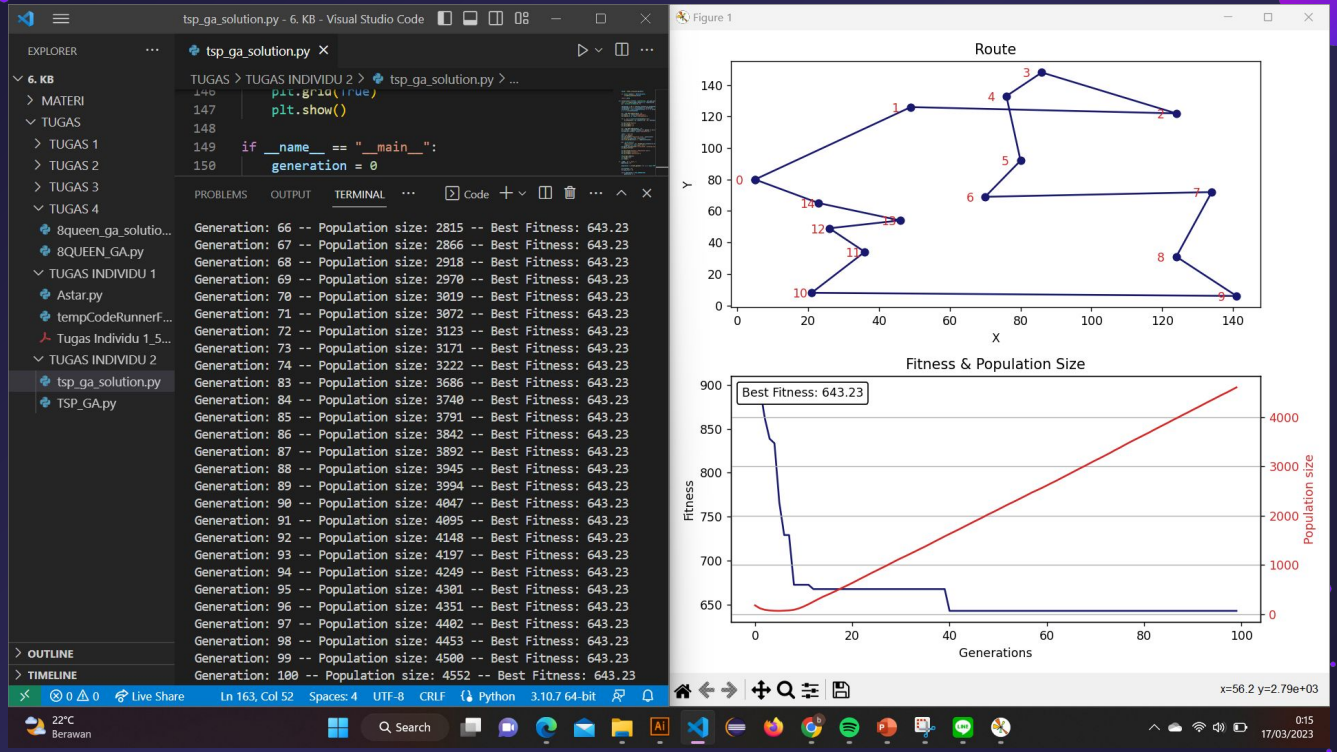
Implementasi

```
def visualize(all_fittest: List[Genome], all_pop_size: List[int]):
    fig = plt.figure(tight_layout=True, figsize=(10, 6))
    gs = gridspec.GridSpec(2, 1)

    chromosome = [0] * (len(all_fittest[-1].chromosome) + 2)
    chromosome[1:-1] = all_fittest[-1].chromosome
    coordinates = [CITY_COORDINATES[i] for i in chromosome]
    x, y = zip(*coordinates)
```

Implementasi

```
if __name__ == "__main__":
    generation = 0

    population = [create_genome() for x in range (POPULATION_SIZE)]

    all_fittest = []
    all_pop_size = []

    while generation != MAX_GENERATION:
        generation += 1
        print("Generation: {0} -- Population size: {1} -- Best Fitness: {2}"
            .format(generation, len(population), get_fittest_genome (population).fitness))

        childs = []
        for x in range(int(POPULATION_SIZE * 0.2)):
            child = reproduction(population)
            childs.append(child)
        population.extend(childs)

        for genome in population:
            if genome.fitness > WEAKNESS_THRESHOLD:
                population.remove(genome)

        all_fittest.append(get_fittest_genome (population))
        all_pop_size.append(len(population))

    visualize(all_fittest, all_pop_size)
```

Hasil / Output



TERIMA KASIH