PENYELESAIAN MASALAH TSP MENGGUNAKAN GA

NAJMA ULYA AGUSTINA-5025211239

APA ITU GA (GENETIC ALGORITHM)?

Genetic Algorithm (GA) adalah salah satu teknik optimasi yang terinspirasi dari proses evolusi di alam. GA adalah metode pencarian heuristik yang digunakan untuk menemukan solusi terbaik dalam sebuah masalah yang kompleks dengan mencari solusi secara iteratif, seperti algoritma optimasi lainnya.

CARA KERJA GA

DALAM MENYELESAIKAN MASALAH TSP

REPRESENTASI KROMOSOM

Setiap kromosom dalam GA merepresentasikan satu solusi dalam ruang pencarian. Dalam TSP, kromosom bisa direpresentasikan sebagai urutan kota yang harus dikunjungi.

INISIALISASI POPULASI AWAL

Populasi awal solusi acak dibuat. Setiap kromosom merepresentasikan satu rute yang mungkin melalui setiap kota.

CARA KERJA GA

DALAM MENYELESAIKAN MASALAH TSP

EVALUASI KINERJA

Setiap kromosom dinilai berdasarkan jarak total yang harus ditempuh untuk menyelesaikan TSP.

SELEKSI

ndividu-individu yang paling cocok dipilih untuk dijadikan orangtua (parent) untuk reproduksi. Seleksi dapat dilakukan menggunakan metode roulette wheel, turnamen, atau metode lainnya.

STEP 5

CROSSOVER

Proses penggabungan informasi dari kedua orangtua untuk menghasilkan offspring baru dengan urutan kota yang berbeda. Crossover dapat dilakukan dengan menggunakan metode one-point, two-point, atau metode lainnya.

STEP 6

MUTASI

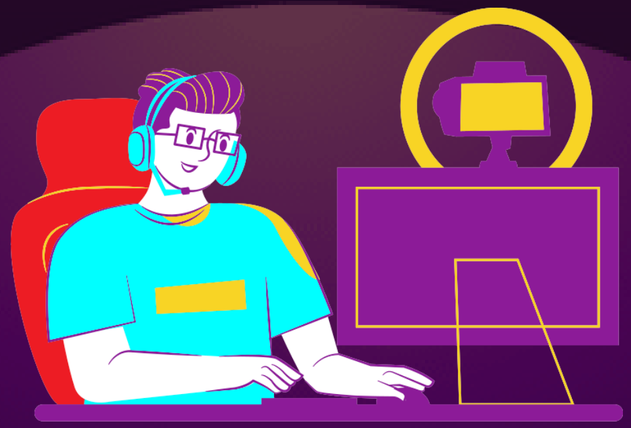
Proses perubahan gen pada offspring baru secara acak untuk menghasilkan variasi genetik baru pada populasi. Mutasi dapat dilakukan dengan probabilitas rendah untuk mempertahankan keragaman populasi.

STEP 7

EVALUASI KINERJA OFFSPRING

Setiap offspring dinilai berdasarkan jarak total yang harus ditempuh untuk menyelesaikan TSP.

STEP8 DAN 9



SELEKSI GEN BERIKUTNYA, ULANGI LANGKAH 5-8

Individu yang paling cocok dipilih untuk membentuk populasi generasi berikutnya. Ulangi langkah 5 hingga 8 sampai solusi terbaik ditemukan atau terpenuhi kriteria penghentian yang ditentukan.

IMPLEMENTASI

Kecerdasan-Buatan > tsp_ganew.py > ...

```python
import random

# Inisialisasi kromosom
def create_chromosome(num_cities):
    chromosome = list(range(num_cities))
    random.shuffle(chromosome)
    return chromosome

# Evaluasi fitness
def fitness(chromosome, distances):
    distance = 0
    for i in range(len(chromosome)-1):
        distance += distances[chromosome[i]][chromosome[i+1]]
    distance += distances[chromosome[-1]][chromosome[0]]
    return 1/distance

# Seleksi orang tua dengan turnamen
def tournament_selection(population, distances):
    parents = []
    for i in range(len(population)//2):
        tournament = random.sample(population, 5)
        tournament_fitness = [fitness(chromosome, distances) for chromosome in tournament]
        winner = tournament[tournament_fitness.index(max(tournament_fitness))]
        parents.append(winner)
        population.remove(winner)
    return parents
```

```python
# Crossover dengan order crossover (OX)
def order_crossover(parent1, parent2):
    child = [-1] * len(parent1)
    start = random.randint(0, len(parent1)-1)
    end = random.randint(start, len(parent1)-1)
    child[start:end+1] = parent1[start:end+1]
    for i in range(len(parent2)):
        if parent2[i] not in child:
            for j in range(len(child)):
                if child[j] == -1:
                    child[j] = parent2[i]
                    break
    return child

# Mutasi dengan swap mutation
def swap_mutation(chromosome):
    index1 = random.randint(0, len(chromosome)-1)
    index2 = random.randint(0, len(chromosome)-1)
    chromosome[index1], chromosome[index2] = chromosome[index2], chromosome[index1]
    return chromosome
```

```python
# Algoritma genetika
def genetic_algorithm(distances, population_size, num_generations):
    num_cities = len(distances)
    population = [create_chromosome(num_cities) for i in range(population_size)]
    for
        (variable) new_population: list
                                    pulation, distances)
        new_population = []
        for j in range(len(parents)//2):
            parent1 = parents[2*j]
            parent2 = parents[2*j+1]
            child1 = order_crossover(parent1, parent2)
            child2 = order_crossover(parent2, parent1)
            child1 = swap_mutation(child1)
            child2 = swap_mutation(child2)
            new_population.append(child1)
            new_population.append(child2)
        population = new_population
    best_chromosome = max(population, key=lambda chromosome: fitness(chromosome, distances))
    return best_chromosome
```

```python
distances = [[0, 10, 15, 20],
             [10, 0, 35, 25],
             [15, 35, 0, 30],
             [20, 25, 30, 0]]
best_chromosome = genetic_algorithm(distances, 100, 1000)
print(best_chromosome)
```


THANK YOU
I HOPE YOU LEARNED SOMETHING NEW!