Tugas Pemrograman 01 – Searching
CII-2M3 Pengantar Kecerdasan Buatan
Semester Genap 2021/2022

Universitas Telkom

Kelompok 6
Hilman Taris Muttaqin
(1301204208)
Herjanto Janawisuta
(1301200421)

Definisi Tugas

Lakukan analisis dan desain algoritma Genetika Algoritma (GA) serta mengimplementasikannya ke dalam suatu program komputer untuk mencari nilai x dan y sehingga diperoleh nilai minimum dari fungsi:

$$h(x,y) = \frac{(\cos x + \sin y)^2}{x^2 + y^2}$$

dengan domain (batas nilai) untuk x dan y

$$-5 \leq x \leq 5 \text{ dan } -5 \leq y \leq 5$$

Analisis dan desain:

Desain kromosom dan metode dekode-nya

- Chromosome Length
  Panjang kromosom adalah 10

```
chromosome_length = 10
```

- Generate Population
  Fungsi ini akan men-generate populasi menggunakan tipe data integer untuk masing-masing gen pada suatu kromosom (Integer encoding).

```
def generate_population(population, kromosom):
    # return random integer k gen in single chromosome and generate p population of chromosome
    return [{CHROMOSOME: [random.randint(0, 9) for _ in range(kromosom)], FITNESS: 0} for _ in range(population)]
```

- Domain Chromosome
  Sebelum sebuah kromosom di decode, kromosom harus dibelah untuk mendapatkan domain *x* dan *y*. Fungsi ini untuk membelah kromosom tersebut.

```
def get_domain_from_kromosom(kromosom):
    # return split gens in chromosome
    # e.g. [1, 2, 4, 2, 3, 3]
    # result = [[1, 2, 4], [2, 3, 3]]
    pembagi = int(chromosome_length / 2)
    return [kromosom[:pembagi], kromosom[pembagi:]]
```

- Decode Populasi
  Representasi data gen yang digunakan adalah dengan tipe data integer. Sehingga butuh rumus untuk digunakan sebagai fungsi decoding dari 3 gen pada kromosom menjadi floating point sesuai dengan limit atas dan limit bawah x dan y.
  (Rumus encoding untuk integer value)

$$x = r_{min} + \frac{r_{max} - r_{min}}{\Sigma_{i=1}^{N} 9 * 10^{-i}} (g_1 * 10^{-1} + g_1 * 10^{-2} + \cdots + g_N * 10^{-N})$$

```python
def decode(kromosom, limit):
    # return decode result from x or y in single chromosome
    # result in floating point data type
    pengali = 0
    pembagi = 0

    for i in range(len(kromosom)):
        gen = kromosom[i]
        pengali += gen * 10**-(i+1)
        pembagi += 9 * 10**-(i+1)

    return limit[LIMIT_BAWAH] + (((limit[LIMIT_ATAS] - limit[LIMIT_BAWAH]) / pembagi) * pengali)
```

- Objective Function

  Menggunakan rumus $h(x, y) = (\cos x + \sin y) / 2\ x\ 2 + y\ 2$. Fungsi ini mengevaluasi kromosom. 1 solusi bisa saja memiliki beberapa fungsi objektif.

```python
def h(x, y):
    # this function called objective function
    # this function provided by this task in task description
    return ((cos(x) + sin(y))**2) / (x**2 + y**2)
```

Ukuran populasi

- Population Size

  Ukuran populasi adalah 10 per generasi

```python
max_population = 10
```

Metode pemilihan orangtua

- Parent selection

  Seleksi orang tua ini merupakan proses untuk memilih suatu individu untuk bisa membuat keturunan pada generasi selanjutnya. Individu dipilih berdasarkan nilai fitnessnya. Fungsi ini digunakan untuk memilih orang tua dengan metode *stokastik roulette wheels.*

```python
def stochastic_roulette_wheel(population):
    max_fitness_chromosome = max(population, key = lambda x: x[FITNESS])
    while True:
        indv = random.uniform(0, 1) * max_population
        indv_chromosome = population[int(indv)]
        r = random.uniform(0, 1)
        if r < indv_chromosome[FITNESS] / max_fitness_chromosome[FITNESS]:
            return population[int(indv)]
```

Metode operasi genetik (pindah silang dan mutasi)

- Crossover

  Fungsi ini mengacak dan menyatukan gen-gen pada setiap parent untuk mendapatkan mutasi pada keturunan baru. Tetapi crossover tidak selalu dapat terjadi, probability crossover di antara 60% - 70%. Fungsi crossover yang digunakan adalah *single point crossover* karena panjang dari kromosomnya tidak begitu panjang.

```
def single_point_crossover(parent1, parent2):

    r = random.uniform(0, 1)

    if r < CROSSOVER_PROBABILITY:
        random_position = random.randint(0, chromosome_length - 1)

        offspring1_chromosome = parent1[CHROMOSOME][:random_position] + parent2[CHROMOSOME][random_position:]
        offspring2_chromosome = parent2[CHROMOSOME][:random_position] + parent1[CHROMOSOME][random_position:]

        offspring1 = {CHROMOSOME: offspring1_chromosome, FITNESS: 0}
        offspring2 = {CHROMOSOME: offspring2_chromosome, FITNESS: 0}

        result = [offspring1, offspring2]
    else:
        result = [parent1, parent2]
```

- Mutation

  Fungsi untuk mensimulasikan efek kesalahan yang terjadi dengan probabilitas yang rendah selama duplikasi.

```
def mutation(offspring_chromosome):

    for index, gen in enumerate(offspring_chromosome):
        r = random.uniform(0, 1)
        if r < MUTATION_PROBABILITY:
            new_gen = random.randint(0, 9)
            offspring_chromosome[index] = new_gen

    return offspring_chromosome
```

Probabilitas operasi genetik (Pc dan Pm)

- Crossover Probability (Pc)

  Probabilitas crossover adalah 70%

```
CROSSOVER_PROBABILITY = 0.7
```

- Crossover Mutation (Pc)

  Probabilitas mutasi adalah 1%

```
MUTATION_PROBABILITY = 0.01
```

Metode pergantian generasi (seleksi survivor)

- Survivor Selection
  Untuk Metode yang digunakan untuk seleksi kita memakai dua metode yaitu *Steady-State Local Fitness-based Selection* dan *Generational Replacement.*
- *Steady-State Local Fitness-based Selection*

```python
# max population replacement in this case is 4
POPULATION_REPLACEMENT = 2

def steady_state(population, parents, offsprings):
    new_population = population


    # combine parents and offsprings
    next_gen_candidate = parents + offsprings

    next_gen = []
    for i in range(POPULATION_REPLACEMENT):
        if len(next_gen_candidate) != 0:
            max_fitness_indv = max(next_gen_candidate, key = lambda x: x[FITNESS])
            next_gen.append(max_fitness_indv)
            next_gen_candidate.remove(max_fitness_indv)

    remaining_generations = 0
    for index, pop in enumerate(new_population):
        if remaining_generations <= 1:
            for parent in parents:
                if pop == parent and len(next_gen) != 0:
                    new_population[index] = next_gen[remaining_generations]
                    remaining_generations += 1


    return new_population
```

- *Generational Replacement.*

```python
# if the population size is even, elisitism the population twice
# if the popoulation size is odd, elisitism the population in single value
def elisitism(population):

    if len(population) % 2 != 0:
        return [max(population, key = lambda x: x[FITNESS])]
    else:
        new_population = []
        while len(new_population) < 2:
            max_fitness_chromosome = max(population, key = lambda x: x[FITNESS])
            new_population.append(max_fitness_chromosome)
        return new_population
```

Kriteria penghentian evolusi (loop)

- *Steady State*

  Untuk penghentian evolusi, menggunakan loop sebesar maks generasi dimana maks nya 100.

```
generation_count = 100
```

```
# generate initial chromosom
populations = generate_population(max_population, chromosome_length)

best_chromosome = max(populations, key = lambda x: x[FITNESS])
print("Kromosom terbaik : ", best_chromosome[CHROMOSOME])
print("Fitness value = ", best_chromosome[FITNESS])

for i in range(generation_count):

    # hitung nilai fitness
    for chromosome in populations:
        domain = get_domain_from_kromosom(chromosome[CHROMOSOME])
        current_x = decode(domain[X], limit_x)
        current_y = decode(domain[Y], limit_y)
        chromosome[FITNESS] = fitness(current_x, current_y)

    # seleksi orang tua
    parents = []
    parents.append(stochastic_roulette_wheel(populations))
    parent2 = stochastic_roulette_wheel(populations)
    while parent2 == parents[0]:
        parent2 = stochastic_roulette_wheel(populations)
    parents.append(parent2)
```

```
# crossover
offsprings = single_point_crossover(parents[0], parents[1])

# mutation
for offspring in offsprings:
    offspring[CHROMOSOME] = mutation(offspring[CHROMOSOME])
    domain = get_domain_from_kromosom(offspring[CHROMOSOME])
    current_x = decode(domain[X], limit_x)
    current_y = decode(domain[Y], limit_y)
    offspring[FITNESS] = fitness(current_x, current_y)

# survivor selection
populations = steady_state(populations, parents, offsprings)

print(f"\n\n=================== gen {i} ===================")

best_chromosome = max(populations, key = lambda x: x[FITNESS])
print("Kromosom terbaik : ", best_chromosome[CHROMOSOME])
print("Fitness value = ", best_chromosome[FITNESS])
```

```python
print(f"\n\n================== FINAL GENERATION ==================")
for chromosome in populations:
    domain = get_domain_from_kromosom(chromosome[CHROMOSOME])
    current_x = decode(domain[X], limit_x)
    current_y = decode(domain[Y], limit_y)
    print(f"Kromosom: {chromosome[CHROMOSOME]}")
    print(f"x: {current_x}")
    print(f"y: {current_y}")
    print(f"Fitness: {chromosome[FITNESS]}")
    print("=============")
```

- Best Chromosome Output untuk *Steady state*

```
================== FINAL GENERATION ==================
Kromosom: [8, 3, 4, 4, 6, 4, 8, 4, 6, 4]
x: 3.344683446834468
y: -0.15355153551535583
Fitness: 8.742430811614245
=============
Kromosom: [5, 5, 2, 1, 8, 2, 8, 9, 4, 9]
x: 0.5218552185521848
y: -2.105071050710507
Fitness: 120117.96587450503
=============
Kromosom: [5, 5, 2, 1, 8, 2, 8, 9, 4, 9]
x: 0.5218552185521848
y: -2.105071050710507
Fitness: 120117.96587450503
=============
Kromosom: [8, 1, 2, 7, 4, 4, 5, 4, 4, 0]
x: 3.127481274812748
y: -0.45595455954559583
Fitness: 4.815870772049654
=============
Kromosom: [1, 5, 2, 1, 8, 2, 8, 6, 4, 8]
x: -3.4781847818478187
y: -2.1351713517135167
Fitness: 5.205592603186577
=============
Kromosom: [6, 7, 5, 9, 6, 0, 5, 7, 4, 0]
x: 1.7596675966759667
y: -4.425994259942599
Fitness: 38.11219093843569
```

```
=============
Kromosom: [5, 5, 2, 1, 8, 2, 8, 9, 4, 9]
x: 0.5218552185521848
y: -2.105071050710507
Fitness: 120117.96587450503
=============
Kromosom: [5, 5, 2, 1, 8, 2, 8, 5, 4, 8]
x: 0.5218552185521848
y: -2.145171451714517
Fitness: 6509.354528010062
=============
Kromosom: [8, 4, 6, 3, 0, 9, 9, 7, 5, 2]
x: 3.463084630846309
y: 4.975299752997529
Fitness: 10.02661858680858
=============
Kromosom: [9, 6, 0, 6, 8, 2, 8, 6, 4, 8]
x: 4.606896068960689
y: -2.1351713517135167
Fitness: 28.554658077119125
=============
```

- *Generational Replacement.*

```python
populations = generate_population(max_population, chromosome_length)

for i in range(generation_count):

    # hitung nilai fitness
    for chromosome in populations:
        domain = get_domain_from_kromosom(chromosome[CHROMOSOME])
        current_x = decode(domain[X], limit_x)
        current_y = decode(domain[Y], limit_y)
        chromosome[FITNESS] = fitness(current_x, current_y)

    # elisitism
    new_population = elisitism(populations)

    while len(new_population) < max_population:

        # seleksi orang tua
        parents = []
        parents.append(stochastic_roulette_wheel(populations))
        parent2 = stochastic_roulette_wheel(populations)
        while parent2 == parents[0]:
            parent2 = stochastic_roulette_wheel(populations)
        parents.append(parent2)

        # crossover
        offsprings = single_point_crossover(parents[0], parents[1])
```

```python
    # mutation
    for offspring in offsprings:
        offspring[CHROMOSOME] = mutation(offspring[CHROMOSOME])
        domain = get_domain_from_kromosom(offspring[CHROMOSOME])
        current_x = decode(domain[X], limit_x)
        current_y = decode(domain[Y], limit_y)
        offspring[FITNESS] = fitness(current_x, current_y)

    # survivor selection
    populations = steady_state(populations, parents, offsprings)

    print(f"\n\n=================== gen {i} ===================")

    best_chromosome = max(populations, key = lambda x: x[FITNESS])
    print("Kromosom terbaik : ", best_chromosome[CHROMOSOME])
    print("Fitness value = ", best_chromosome[FITNESS])

print(f"\n\n================== FINAL GENERATION ==================")
for chromosome in populations:
    domain = get_domain_from_kromosom(chromosome[CHROMOSOME])
    current_x = decode(domain[X], limit_x)
    current_y = decode(domain[Y], limit_y)
    print(f"Kromosom: {chromosome[CHROMOSOME]}")
    print(f"x: {current_x}")
    print(f"y: {current_y}")
    print(f"Fitness: {chromosome[FITNESS]}")
    print("============")
```

- Best Chromosome Output untuk *Generational Replacement.*

```
================== FINAL GENERATION ==================
Kromosom: [0, 5, 6, 8, 9, 5, 2, 8, 2, 0]
x: -4.431094310943109
y: 0.2820528205282047
Fitness: 37169699.3930348
============
Kromosom: [0, 5, 6, 8, 9, 5, 2, 8, 2, 0]
x: -4.431094310943109
y: 0.2820528205282047
Fitness: 37169699.3930348
============
Kromosom: [0, 5, 6, 8, 6, 5, 2, 8, 2, 0]
x: -4.431394313943139
y: 0.2820528205282047
Fitness: 19081989.022374146
============
Kromosom: [0, 5, 6, 8, 9, 5, 2, 8, 2, 0]
x: -4.431094310943109
y: 0.2820528205282047
Fitness: 37169699.3930348
============
Kromosom: [0, 5, 6, 8, 9, 5, 2, 8, 2, 7]
x: -4.431094310943109
y: 0.2827528275282747
Fitness: 10050511.034631765
============
```

```
Kromosom: [0, 5, 6, 8, 9, 5, 2, 8, 2, 0]
x: -4.431094310943109
y: 0.2820528205282047
Fitness: 37169699.3930348
============
Kromosom: [0, 5, 6, 8, 9, 5, 2, 8, 2, 7]
x: -4.431094310943109
y: 0.2827528275282747
Fitness: 10050511.034631765
============
Kromosom: [0, 5, 6, 8, 9, 4, 2, 8, 2, 0]
x: -4.431094310943109
y: -0.7179571795717958
Fitness: 23.027123709723675
============
Kromosom: [0, 5, 6, 8, 6, 5, 2, 8, 2, 0]
x: -4.431394313943139
y: 0.2820528205282047
Fitness: 19081989.022374146
============
Kromosom: [0, 5, 6, 8, 9, 5, 2, 8, 2, 0]
x: -4.431094310943109
y: 0.2820528205282047
Fitness: 37169699.3930348
============
```

Peran Anggota Kelompok

- Hilman Taris Muttaqin
    1. Merekam dan mengedit video presentasi
    2. Membuat fungsi Decode
    3. Membuat fungsi Fitness
    4. Membuat fungsi Stochastic Roulette Wheel
    5. Membuat fungsi Steady State Selection Survivor
    6. Membuat fungsi General Replacement Selection Survivor
    7. Membuat main untuk Metode Steady State
    8. Membuat main untuk Metode General Replacement
- Herjanto Janawisuta
    1. Membuat laporan
    2. Membuat fungsi Generate Population
    3. Membuat fungsi objektif
    4. Membuat fungsi Domain Chromosome
    5. Membuat fungsi Crossover
    6. Membuat fungsi Mutation

Berikut Link Video Presentasi Kelompok 6
https://drive.google.com/file/d/1x--bAraG19MMS3WmvMuYxBj6ENarfo9H/view?usp=sharing