

TUGAS PEMROGRAMAN 1
PENGANTAR KECERDASAN BUATAN
GENETIC ALGORITHM



Disusun oleh:

1301218548 – Irgi Ahmad Maulana

1301218586 - Muhamad Azmi Rizkifar

FAKULTAS INFORMATIKA
PROGRAM STUDI S1 INFORMATIKA
UNIVERSITAS TELKOM

2021/2022

Analisis, desain dan implementasi Genetic Algorithm (GA) ke dalam suatu program komputer untuk menemukan nilai minimum dari fungsi :

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

Dengan domain untuk x dan y :

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

Desain kromosom dan metode dekode-nya

Evolusi biasanya dibentuk dari populasi individu (kromosom) yang dibuat secara acak, dimana desain kromosom yang digunakan pada algoritma yang kami rancang dibuat dengan menggunakan fungsi random yang ada pada library python, dengan melakukan random nilai antara 1 dan 0 sebagai bagian dari kromosom sejumlah n kromosom. Setelah didapat nilai random antara 1 dan 0, maka terbentuklah nilai biner yang selanjutnya akan di proses dekode. Dalam proses pengkodean algoritmanya, kami menggunakan metode *Individual Representation Binary encoding* yang cara kerjanya membagi kromosom menjadi dua bagian x dan y dimana indeks pertama dari kromosom hingga indeks tengah dari panjang kromosom menjadi bagian x dan sisanya menjadi bagian y . Kemudian bagian x akan diubah menjadi bilangan desimal dengan menggunakan perhitungan rumus :

$$x = r_{min} + \frac{r_{max} - r_{min}}{\sum_{i=1}^N 2^{-i}} (g_1 * 2^{-1} + g_2 * 2^{-2} + \dots + g_N * 2^{-N})$$

Dan untuk bagian y akan diubah menjadi bilangan desimal dengan menggunakan rumus :

$$y = r_{min} + \frac{r_{max} - r_{min}}{\sum_{i=1}^N 2^{-i}} (g_1 * 2^{-1} + g_1 * 2^{-2} + \dots + g_N * 2^{-N})$$

Implementasi :

```

def generatePopulation(chromosomeLength, n):
    # Inisiasi variabel populasi
    population = []

    # Looping sebanyak n populasi
    for i in range(n):
        # Inisiasi variabel kromosom
        chromosome = []

        # Loop sebanyak length dari kromosom
        for j in range(chromosomeLength):
            # Random nilai biner antara 1 dan 0 untuk melakukan pengisian nilai kromosom
            chromosome.append(random.randint(0,1))

        # Memasukkan nilai kromosom ke dalam populasi
        population.append(chromosome)

    # Mengembalikan nilai populasi yang berisikan n kromosom
    return population

```

```

def chromosomeDecode(chromosome, intervalX, intervalY):
    half_chromosome = len(chromosome)//2

    #menghitung bagian kiri genotype
    left_x = 0
    chromosome_x = chromosome[:half_chromosome]
    for i in range (1, half_chromosome+1):
        left_x += chromosome_x[i-1] * (2**-i)

    #menghitung bagian kanan genotype
    right_y = 0
    chromosome_y = chromosome[half_chromosome:]
    for i in range (1, half_chromosome+1):
        right_y += chromosome_y[i-1] * (2**-i)

    sum_bot = sum([2 ** -(i) for i in range (1, half_chromosome + 1)])

    #rumus menghitung genotype
    x = intervalX["min"] + (left_x * (intervalX["max"] - intervalX["min"])) / sum_bot
    y = intervalY["min"] + (right_y * (intervalY["max"] - intervalY["min"])) / sum_bot

    return x,y

```

Ukuran populasi

Kami menggunakan ukuran populasi sebanyak 20 dan 50 pada setiap generasi untuk membandingkan kecepatan pencarian generasi terbaik serta nilai minimum dari fungsi yang telah ditentukan berdasarkan soal. Karena semakin besar ukuran populasi, maka semakin banyak pula individu/kromosom yang bermutasi dan bisa menghasilkan lebih banyak nilai fitness terbaik.

Metode pemilihan orangtua

Menggunakan metode roulette wheel selection, cara yang digunakan pada roulette wheel selection antara lain :

1. Individu diberi probabilitas terpilih yang berbanding lurus dengan fitnessnya
2. Dua individu kemudian dipilih secara acak berdasarkan probabilitas dan menghasilkan keturunan.

Algoritma:

```
function RouletteWheelSelection (fitness)
```

```
    total = 0
```

```
    for indv = 1 to n do
```

```
        total += fitness(indv)
```

```
    r = random_uniform()
```

```
    indv = 1
```


```
    while( r > 0 ) do
```

```
        r -= fitness(indv)/total
```

```
        indv ++
```

```
    return indv - 1
```

Implementasi :




```
def twoParentSelection(population, fitnessPopulation):
    # Inisiasi variable temporary
    temp = 0

    for chromosome in range(len(population)):
        temp += fitnessPopulation[chromosome]

    rng = random.random()

    chromosome = 0
    while rng > 0 :
        rng -= fitnessPopulation[chromosome] / temp
        chromosome += 1

    return chromosome - 1
```



```
def parentSelection(population):
    n = len(population)

    fitnessPopulation = []
    for chromosome in range(n):
        x, y = chromosomeDecode(population[chromosome], intervalX, intervalY)
        fitnessPopulation.append(fitnessFunction(x, y))

    # Normalisasi nilai fitness
    min_ = min(fitnessPopulation)
    max_ = max(fitnessPopulation)

    for i in range(n):
        fitnessPopulation[i] = (fitnessPopulation[i] - min_) / (max_ - min_)

    # Melakukan Roulette Wheel untuk mendapatkan 2 parent
    parent = []

    while len(parent) != 2:
        chromosome = twoParentSelection(population, fitnessPopulation)
        parent.append(population[chromosome])

    return parent
```

Metode operasi genetik (pindah silang dan mutasi)

Crossover

Teknik ini Memungkinkan proses evolusi untuk bergerak menuju wilayah *search space* yang menjanjikan, setelah itu teknik ini mencocokkan sub-solusi orang tua yang baik untuk membangun keturunan yang lebih baik. Disini kami menggunakan teknik single point crossover. Single point crossover bekerja dengan cara mengambil Sebuah titik pada kromosom kedua orang tua yang diambil secara acak, dan ditetapkan sebagai 'titik pindah silang'. Bit di sebelah kanan titik itu ditukar di antara dua kromosom induk. hal tersebut menghasilkan dua keturunan, masing-masing membawa beberapa informasi genetik dari kedua orang tuanya.

Algoritma

```
def singlePointCrossover(parent):
    chromosomeLength = len(parent[0])

    intersectionPoint = random.randint(0, chromosomeLength)

    child = [0, 0]
    child[0] = parent[0][:intersectionPoint] + parent[1][intersectionPoint:]
    child[1] = parent[1][:intersectionPoint] + parent[0][intersectionPoint:]

    return child
```

Mutation

Mutasi adalah operator genetika yang digunakan untuk mempertahankan keragaman genetik dari satu generasi populasi kromosom algoritma genetika ke generasi berikutnya. Teknik mutasi yang kami gunakan adalah *bit string mutation*. Teknik ini memungkinkan bit-bit yang telah dibuat akan di flip atau di inverse secara *random*.

Algoritma

```
def chromosomeMutation(child, probability):
    chromosomeLength = len(child[0])

    for i in range(2):
        for j in range(chromosomeLength):
            if probability ≥ random.random():
                child[i][j] = [0, 1][not child[i][j]]

    return child
```

Probabilitas operasi genetik (P_c dan P_m)

Probabilitas crossover yang kami berikan bernilai 0.6 sedangkan probabilitas mutation adalah sebesar 0.05 karena menurut hasil observasi, semakin kecil nilai PM maka akan semakin cepat mendapatkan nilai *best fitness*.

Metode pergantian generasi (seleksi survivor)

Metode yang digunakan oleh kami yaitu metode general replacement, metode ini menghasilkan *n-off springs*, dimana n adalah ukuran populasi, dan seluruh populasi diganti dengan yang baru di akhir iterasi. Metode ini harus menambahkan mekanisme untuk memastikan individu terbaik bertahan (*elitism*)

Kriteria penghentian evolusi (loop)

Pada program ini kami membatasi proses regenerasi sampai generasi ke-n. Dari n generasi tersebut, didapatkan nilai menaik dan atau linear tergantung dari individu dan populasi yang *ter-generate*.

Kesimpulan

Kesimpulan yang didapatkan dari masalah yang diselesaikan menunjukkan bahwa setiap *properties* yang ada pada *genetic algorithm* berpengaruh untuk mendapatkan hasil yang diinginkan yaitu meminimalkan fungsi, *properties* ini berpengaruh pada kecepatan untuk mendapatkan nilai dari fungsi minimal tersebut, *properties* yang dimaksud antara lain.

1. Individu dan Populasi
Semakin banyak individu dan populasi yang ada, maka program akan lebih cepat untuk mendapatkan *best fitness value*. Jumlah Individu berpengaruh juga pada nilai *fitness* yang didapatkan
2. Mutasi
Mutasi yang lebih kecil akan mendapatkan nilai minimum yang lebih cepat
3. Crossover
Nilai crossover yang lebih besar akan berpengaruh pada kecepatan pencarian untuk nilai *fitness* yang terbaik

Selain *properties* didapatkan kesimpulan juga bahwa mencari nilai minimum dari sebuah fungsi, kita harus melakukan invers pada fungsi dengan menggunakan rumus:

$$f = 1/(h + a)$$

h : fungsi yang disediakan

f : nilai fitness

a : nilai konstanta yang kecil

Hasil Nilai GA Terbaik

Population Size	: 20
Chromosome Size	: 10
Generation Size	: 20
Elitism Size	: 2
Crossover Probability	: 0.6
Mutation Probability	: 0.05

x = -0.16129032258064502, y = -4.67741935483871 Generasi 1, Best fitness: 28.84655628971235	x = 3.064516129032258, y = 4.35483870967742 Generasi 9, Best fitness: 35.01572950262775	x = 3.064516129032258, y = 4.35483870967742 Generasi 17, Best fitness: 35.01572950262775
x = 4.67741935483871, y = 1.451612903225806 Generasi 2, Best fitness: 28.84655628971235	x = -2.4193548387096775, y = 5.0 Generasi 10, Best fitness: 35.01572950262775	x = 3.064516129032258, y = 3.387096774193548 Generasi 18, Best fitness: 35.01572950262775
x = -2.4193548387096775, y = -4.032258064516129 Generasi 3, Best fitness: 28.84655628971235	x = 2.741935483870968, y = 5.0 Generasi 11, Best fitness: 35.01572950262775	x = 3.064516129032258, y = 4.35483870967742 Generasi 19, Best fitness: 35.01572950262775
x = 2.741935483870968, y = 4.67741935483871 Generasi 4, Best fitness: 28.84655628971235	x = 3.387096774193548, y = 4.67741935483871 Generasi 12, Best fitness: 35.01572950262775	x = 3.064516129032258, y = 4.03225806451613 Generasi 20, Best fitness: 35.01572950262775
x = 0.48387096774193505, y = 3.387096774193548 Generasi 5, Best fitness: 32.355498523293726	x = 3.064516129032258, y = 3.387096774193548 Generasi 13, Best fitness: 35.01572950262775	
x = 2.741935483870968, y = 4.67741935483871 Generasi 6, Best fitness: 35.01572950262775	x = 3.064516129032258, y = 3.387096774193548 Generasi 14, Best fitness: 35.01572950262775	
x = 2.741935483870968, y = 4.67741935483871 Generasi 7, Best fitness: 35.01572950262775	x = 2.741935483870968, y = 4.67741935483871 Generasi 15, Best fitness: 35.01572950262775	
x = -2.096774193548387, y = 4.67741935483871 Generasi 8, Best fitness: 35.01572950262775	x = -1.129032258064516, y = 2.741935483870968 Generasi 16, Best fitness: 35.01572950262775	

