

GENETIC ALGORITHM



agungsetiabudi@ub.ac.id

Definisi *Genetic Algorithm*

Genetic Algorithm (GA) adalah metode optimisasi yang terinspirasi dari proses seleksi alam dalam evolusi biologis. GA digunakan untuk menemukan solusi optimal atau mendekati optimal dalam berbagai masalah, terutama ketika ruang pencarian sangat besar dan tidak ada metode deterministik yang efisien untuk menemukan solusi.

Tahap-Tahap Genetic Algorithm:

1. Inisialisasi Populasi
2. Evaluasi Fitness
3. Seleksi
4. Crossover (Rekombinasi)
5. Mutasi
6. Penggantian
7. Pengulangan

Tahap-Tahap Genetic Algorithm:

1. Inisialisasi Populasi
2. Evaluasi Fitness
3. Seleksi
4. Crossover (Rekombinasi)
5. Mutasi
6. Penggantian
7. Pengulangan

1. Inisialisasi Populasi

- Inisialisasi populasi adalah langkah pertama dalam algoritma genetik yang bertujuan untuk menciptakan populasi awal dari individu yang akan dievaluasi dan berkembang.
- Tahap ini sangat penting karena kualitas dan keragaman populasi awal dapat memengaruhi efektivitas algoritma dalam mencari solusi optimal. Berikut adalah beberapa aspek yang perlu dipertimbangkan dalam inisialisasi populasi:
 - Ukuran populasi
 - Metode pengacakan
 - Diversifikasi

1. Inisialisasi Populasi

- **Ukuran Populasi:** Ukuran populasi yang lebih besar biasanya dapat menghasilkan solusi yang lebih beragam dan mengurangi risiko terjebak dalam solusi lokal.
- **Metode Pengacakan:** Pengacakan harus dilakukan dalam batasan ruang solusi agar individu yang dihasilkan relevan dengan masalah yang ingin diselesaikan.
- **Diversifikasi:** Penting untuk memastikan bahwa populasi awal memiliki keragaman yang cukup untuk memungkinkan eksplorasi ruang solusi. Keragaman ini membantu mencegah populasi terjebak di dalam area sempit solusi.

1. Inisialisasi Populasi (Contoh)

Misalkan kita ingin memecahkan masalah optimasi fungsi sederhana:
 $f(x) = x^2$ dalam rentang $[-10, 10]$.

- 1. Tentukan ukuran populasi:** misal kita pilih 5
- 2. Inisialisasi acak:** menentukan 5 nilai acak dalam rentang yang ditentukan.

1. Inisialisasi Populasi (Python)

```
import random

def initialize_population(size, lower_bound, upper_bound):
    population = []
    for _ in range(size):
        individual = random.randint(lower_bound, upper_bound)
        population.append(individual)
    return population
# Inisialisasi populasi
population_size = 5
lower_bound = -10
upper_bound = 10
population = initialize_population(population_size, lower_bound, upper_bound)
print("Populasi Awal:", population)
```

1. Inisialisasi Populasi (Matlab)

```
function population = initialize_population(size, lower_bound, upper_bound)
    population = zeros(1, size); % Inisialisasi array populasi dengan nol
    for i = 1:size
        population(i) = randi([lower_bound, upper_bound]);
    end
end
% Inisialisasi populasi
population_size = 5;
lower_bound = -10;
upper_bound = 10;
population = initialize_population(population_size, lower_bound, upper_bound);
% Menampilkan hasil
disp('Populasi Awal:');
disp(population);
```

1. Inisialisasi Populasi (Output Program)

Populasi Awal: [3, -7, 2, 8, -1]

2. Evaluasi Fitness

- Evaluasi fitness adalah langkah krusial dalam algoritma genetik, di mana setiap individu dalam populasi dinilai untuk menentukan seberapa baik mereka menyelesaikan masalah yang ingin dipecahkan.
- Fitness berfungsi sebagai indikator kualitas individu, dan nilai fitness ini akan memengaruhi probabilitas individu tersebut terpilih untuk reproduksi di langkah selanjutnya.
- Aspek Penting dalam Evaluasi Fitness: 1) **Fungsi Fitness**, 2) **Skala Penilaian**, 3) **Keterhubungan dengan tujuan Masalah**.

Tahap-Tahap Genetic Algorithm:

1. Inisialisasi Populasi
2. **Evaluasi Fitness**
3. Seleksi
4. Crossover (Rekombinasi)
5. Mutasi
6. Penggantian
7. Pengulangan

2. Evaluasi Fitness

- **Fungsi fitness:** metode matematis yang digunakan untuk menghitung nilai fitness dari setiap individu. Fungsi ini harus dirancang sesuai dengan tujuan masalah yang ingin diselesaikan.
- **Skala penilaian:** Pernyataan fitness tergantung pada jenis masalah. Misalnya, dalam masalah minimisasi, individu dengan nilai fitness lebih rendah dianggap lebih baik.
- **Keterhubungan dengan tujuan Masalah:** Fungsi fitness harus relevan dengan tujuan akhir. Misalnya, jika tujuan adalah memaksimalkan keuntungan, fungsi fitness harus menghitung nilai keuntungan yang dihasilkan oleh individu tersebut.

2. Evaluasi Fitness (Contoh)

Misalkan kita masih menggunakan contoh sebelumnya, yaitu optimasi fungsi sederhana: $f(x) = x^2$ dalam rentang $[-10, 10]$. Dalam konteks ini, semakin kecil nilai x^2 , semakin baik individu tersebut.

2. Evaluasi Fitness (Python)

```
def evaluate_fitness(population):
    fitness = [] # Inisialisasi daftar fitness
    for individual in population:
        fitness_value = individual ** 2 # Menghitung nilai fitness
        fitness.append(fitness_value) # Menambahkan nilai fitness ke daftar
    return fitness

# Contoh populasi
population = [3, -7, 2, 8, -1]

# Evaluasi fitness
fitness_values = evaluate_fitness(population)

# Menampilkan hasil
print("Nilai Fitness:", fitness_values)
```

2. Evaluasi Fitness (Matlab)

```
function fitness = evaluate_fitness(population)
    fitness = zeros(1, length(population)); % Inisialisasi array fitness
    for i = 1:length(population)
        fitness(i) = population(i)^2; % Menghitung nilai fitness
    end
end

% Contoh populasi
population = [3, -7, 2, 8, -1];

% Evaluasi fitness
fitness_values = evaluate_fitness(population);

% Menampilkan hasil
disp('Nilai Fitness:');
disp(fitness_values);
```

2. Evaluasi Fitness (Hasil program)

Nilai Fitness: [9, 49, 4, 64, 1]

Tahap-Tahap Genetic Algorithm:

1. Inisialisasi Populasi
2. Evaluasi Fitness
3. **Seleksi**
4. Crossover (Rekombinasi)
5. Mutasi
6. Penggantian
7. Pengulangan

3. Seleksi

- Seleksi adalah tahap di mana individu-individu dalam populasi dipilih untuk reproduksi berdasarkan nilai fitness mereka.
- Tujuan dari seleksi adalah untuk memastikan bahwa individu yang lebih baik (dengan nilai fitness yang lebih tinggi) memiliki peluang lebih besar untuk menghasilkan keturunan.
- Sehingga generasi berikutnya diharapkan dapat lebih baik daripada generasi sebelumnya.

Metode Seleksi

- **Seleksi Roulette Wheel:** Individu dipilih berdasarkan proporsi nilai fitness mereka. Seperti roda roulette, individu dengan fitness lebih tinggi memiliki peluang lebih besar untuk dipilih.
- **Tournament Selection:** Beberapa individu dipilih secara acak, dan individu dengan fitness terbaik di antara mereka dipilih sebagai pemenang.
- **Seleksi Rank:** Individu diurutkan berdasarkan nilai fitness, dan probabilitas seleksi ditentukan berdasarkan posisi individu dalam urutan.

Contoh Seleksi

- Misalkan kita masih menggunakan contoh sebelumnya, yaitu optimasi fungsi sederhana: $f(x) = x^2$ dalam rentang $[-10, 10]$.
- Sebelumnya sudah didapatkan populasi dan fitness sebagai berikut:

Populasi Awal: [3, -7, 2, 8, -1]

Nilai Fitness: [9, 49, 4, 64, 1]

Contoh Program Seleksi (Python)

```
import random

def tournament_selection(population, fitness, tournament_size=2):
    selected_indices = random.sample(range(len(population)), tournament_size) # Memilih individu acak
    best_index = selected_indices[0] # Asumsikan individu pertama adalah yang terbaik
    for index in selected_indices:
        if fitness[index] < fitness[best_index]: # Menggunakan minimisasi sebagai contoh
            best_index = index
    return population[best_index] # Mengembalikan individu terbaik

# Contoh populasi dan fitness
population = [3, -7, 2, 8, -1]
fitness_values = [9, 49, 4, 64, 1] # Fitness sesuai dengan  $x^2$ 

# Melakukan seleksi
selected_individual = tournament_selection(population, fitness_values)
print("Individu Terpilih:", selected_individual)
```

Contoh Program Seleksi (Python)

```
function selected_individual = tournament_selection(population, fitness, tournament_size)
    selected_indices = randperm(length(population), tournament_size); % Memilih indeks acak
    best_index = selected_indices(1); % Asumsikan individu pertama adalah yang terbaik
    for i = 1:tournament_size
        if fitness(selected_indices(i)) < fitness(best_index) % Menggunakan minimisasi
            best_index = selected_indices(i);
        end
    end
    selected_individual = population(best_index); % Mengembalikan individu terbaik
end
% Contoh populasi dan fitness
population = [3, -7, 2, 8, -1];
fitness_values = [9, 49, 4, 64, 1]; % Fitness sesuai dengan x^2
tournament_size = 2;
% Melakukan seleksi
selected_individual = tournament_selection(population, fitness_values, tournament_size);
disp('Individu Terpilih:');
disp(selected_individual);
```

Hasil Program

Individu Terpilih: 2

Tahap-Tahap Genetic Algorithm:

1. Inisialisasi Populasi
2. Evaluasi Fitness
3. Seleksi
4. **Crossover (Rekombinasi)**
5. Mutasi
6. Penggantian
7. Pengulangan

4. Crossover (Rekombinasi)

- Crossover, atau rekombinasi, adalah tahap dalam algoritma genetik di mana dua individu (atau kromosom) bergabung untuk menghasilkan satu atau lebih keturunan.
- Proses ini bertujuan untuk menggabungkan informasi genetik dari kedua induk untuk menciptakan individu baru yang memiliki potensi untuk lebih baik dalam menyelesaikan masalah yang dihadapi.

Kromosom

- Kromosom dalam konteks algoritma genetik adalah representasi dari individu dalam populasi yang berisi informasi genetik.
- Kromosom dapat direpresentasikan dalam dua format umum: **biner** dan **real**. Pilihan antara kedua format ini tergantung pada jenis masalah yang dipecahkan dan bagaimana informasi dapat diwakili secara efektif.

Jenis Kromosom

- **Kromosom Biner:** Kromosom biner adalah representasi individu yang terdiri dari string biner (0 dan 1). Setiap bit dalam string dapat merepresentasikan gen. Kromosom biner sering digunakan dalam masalah optimasi yang memerlukan representasi diskrit.
- **Kromosom Real:** Kromosom real menggunakan nilai numerik langsung, sering kali dalam bentuk floating-point, untuk merepresentasikan individu. Ini berguna untuk masalah di mana solusi tidak harus diskrit, seperti optimasi kontinyu.

Contoh Crossover Biner

- Pilih titik crossover secara acak dalam kromosom.
- Tukar bagian kromosom setelah titik crossover antara kedua induk untuk menghasilkan dua keturunan.

Kode Program Crossover (python)

```
import random
def single_point_crossover_binary(parent1, parent2):
    # Memilih titik crossover acak
    crossover_point = random.randint(1, len(parent1) - 1)
    # Membuat keturunan dengan menggabungkan induk
    offspring1 = parent1[:crossover_point] + parent2[crossover_point:]
    offspring2 = parent2[:crossover_point] + parent1[crossover_point:]
    return offspring1, offspring2
# Contoh induk biner
parent1 = '10101'
parent2 = '11000'
# Melakukan crossover
offspring1, offspring2 = single_point_crossover_binary(parent1, parent2)
# Menampilkan hasil
print("Induk 1:", parent1)
print("Induk 2:", parent2)
print("Keturunan 1:", offspring1)
print("Keturunan 2:", offspring2)
```

Kode Program Crossover (Matlab)

```
function [offspring1, offspring2] = single_point_crossover_binary(parent1, parent2)
    % Memilih titik crossover acak
    crossover_point = randi([1, length(parent1)-1]);
    % Membuat keturunan dengan menggabungkan induk
    offspring1 = [parent1(1:crossover_point), parent2(crossover_point+1:end)];
    offspring2 = [parent2(1:crossover_point), parent1(crossover_point+1:end)];
end

% Contoh induk biner
parent1 = '10101';
parent2 = '11000';

% Melakukan crossover
[offspring1, offspring2] = single_point_crossover_binary(parent1, parent2);

% Menampilkan hasil
disp(['Induk 1: ', parent1]);
disp(['Induk 2: ', parent2]);
disp(['Keturunan 1: ', offspring1]);
disp(['Keturunan 2: ', offspring2]);
```

Hasil Program Crossover

Induk 1: 10101

Induk 2: 11000

Keturunan 1: 10100

Keturunan 2: 11001

Contoh Crossover Real

- Memilih titik crossover secara acak dalam bentuk proporsi (0 hingga 1).
- Tukar bagian nilai real dari kedua induk untuk menghasilkan dua keturunan.

Contoh Program Crossover Real (Python)

```
def single_point_crossover_real(parent1, parent2):
    # Memilih titik crossover acak
    crossover_point = random.uniform(0, 1)
    offspring1 = crossover_point * parent1 + (1 - crossover_point) * parent2
    offspring2 = (1 - crossover_point) * parent1 + crossover_point * parent2
    return offspring1, offspring2

# Contoh induk real
parent1_real = 21.0
parent2_real = 24.0

# Melakukan crossover
offspring1_real, offspring2_real = single_point_crossover_real(parent1_real, parent2_real)

# Menampilkan hasil
print("Induk 1:", parent1_real)
print("Induk 2:", parent2_real)
print("Keturunan 1:", offspring1_real)
print("Keturunan 2:", offspring2_real)
```

Contoh Program Crossover Real (Matlab)

```
function [offspring1, offspring2] = single_point_crossover_real(parent1, parent2)
    % Memilih titik crossover acak
    crossover_point = rand(); % Proporsi
    offspring1 = crossover_point * parent1 + (1 - crossover_point) * parent2;
    offspring2 = (1 - crossover_point) * parent1 + crossover_point * parent2;
end

% Contoh induk real
parent1_real = 21.0;
parent2_real = 24.0;

% Melakukan crossover
[offspring1_real, offspring2_real] = single_point_crossover_real(parent1_real, parent2_real);

% Menampilkan hasil
disp(['Induk 1: ', num2str(parent1_real)]);
disp(['Induk 2: ', num2str(parent2_real)]);
disp(['Keturunan 1: ', num2str(offspring1_real)]);
disp(['Keturunan 2: ', num2str(offspring2_real)]);
```

Hasil Program Crossover Real

Induk 1: 21.0

Induk 2: 24.0

Keturunan 1: 22.5

Keturunan 2: 22.5

Tahap-Tahap Genetic Algorithm:

1. Inisialisasi Populasi
2. Evaluasi Fitness
3. Seleksi
4. Crossover (Rekombinasi)
5. **Mutasi**
6. Penggantian
7. Pengulangan

5. Mutasi

- Mutasi adalah proses acak yang terjadi dalam algoritma genetik untuk meningkatkan keragaman genetik dalam populasi.
- Tujuannya adalah untuk mencegah algoritma terjebak dalam solusi lokal dengan memperkenalkan variasi baru ke dalam individu.
- Mutasi dapat terjadi setelah proses crossover dan biasanya melibatkan pengubahan satu atau lebih gen dalam kromosom individu.

Jenis-Jenis Mutasi

- **Mutasi Biner:** Mengubah bit dari 0 ke 1 atau dari 1 ke 0 dalam kromosom biner.
- **Mutasi Real:** Mengubah nilai dalam kromosom real dengan menambahkan nilai acak atau melakukan sedikit perubahan pada nilai tersebut.

Contoh Mutasi Real

- Misalkan kita memiliki kromosom real:
 - Kromosom Awal: $x = 21.0$
- Kita bisa melakukan mutasi dengan menambahkan nilai acak kecil (misalnya, -0.5 hingga +0.5) ke kromosom.

Contoh Program Mutasi Real (Python)

```
def mutate_real(value, mutation_rate=0.1):
    if random.random() < mutation_rate: # Mutasi berdasarkan probabilitas
        mutation_amount = random.uniform(-0.5, 0.5) # Menambahkan nilai acak
        return value + mutation_amount
    return value # Tidak berubah

# Contoh kromosom real
chromosome_real = 21.0

# Melakukan mutasi
mutated_chromosome_real = mutate_real(chromosome_real)

# Menampilkan hasil
print("Kromosom Awal (Real):", chromosome_real)
print("Kromosom Setelah Mutasi (Real):", mutated_chromosome_real)
```

Contoh Program Mutasi Real (Matlab)

```
function mutated_value = mutate_real(value, mutation_rate)
    if rand() < mutation_rate % Mutasi berdasarkan probabilitas
        mutation_amount = rand() * 1 - 0.5; % Menambahkan nilai acak antara -0.5 dan +0.5
        mutated_value = value + mutation_amount; % Menghitung nilai baru setelah mutasi
    else
        mutated_value = value; % Tidak ada perubahan
    end
end

% Contoh kromosom real
chromosome_real = 21.0;
mutation_rate = 0.1; % Tingkat mutasi

% Melakukan mutasi
mutated_chromosome_real = mutate_real(chromosome_real, mutation_rate);

% Menampilkan hasil
disp(['Kromosom Awal (Real): ', num2str(chromosome_real)]);
disp(['Kromosom Setelah Mutasi (Real): ', num2str(mutated_chromosome_real)]);
```

Hasil Program Mutasi Real

Kromosom Awal (Real): 21.0

Kromosom Setelah Mutasi (Real): 20.7

Tahap-Tahap Genetic Algorithm:

1. Inisialisasi Populasi
2. Evaluasi Fitness
3. Seleksi
4. Crossover (Rekombinasi)
5. Mutasi
6. **Penggantian**
7. Pengulangan

6. Penggantian

- Tahap ini bertujuan untuk memperbarui populasi dengan individu-individu baru yang telah dihasilkan melalui crossover dan mutasi.
- Penggantian adalah bagian penting dari proses evolusi dalam algoritma genetik, karena menentukan individu mana yang akan bertahan di generasi berikutnya.

Metode Penggantian

- **Penggantian Total:** Semua individu dalam populasi yang lama digantikan oleh individu baru yang dihasilkan. Ini cocok untuk pendekatan eksploratif.
- **Penggantian Parsial:** Hanya beberapa individu yang digantikan, sering kali dengan cara memilih individu terbaik dari populasi lama dan mempertahankan mereka.
- **Generasi Campuran:** Menggabungkan individu baru dan lama. Individu baru akan ditambahkan ke populasi yang sudah ada tanpa menghapus individu lama.

Contoh Program Penggantian (python)

```
def replacement(population, new_offspring):
    # Menggabungkan populasi lama dan keturunan baru
    combined_population = population + new_offspring
    # Mengurutkan berdasarkan fitness (nilai fitness lebih rendah lebih baik)
    sorted_population = sorted(combined_population, key=evaluate_fitness)
    # Mengambil individu terbaik dari populasi baru
    return sorted_population[:len(population)] # Mengembalikan ukuran yang sama dengan populasi asli

# Contoh populasi awal
population = [21.0, 24.0]
# Individu baru setelah crossover dan mutasi
new_offspring = [20.7, 22.5]

# Melakukan penggantian
updated_population = replacement(population, new_offspring)

# Menampilkan hasil
print("Populasi Setelah Penggantian:", updated_population)
```

Contoh Program Penggantian (Matlab)

```
function updated_population = replacement(population, new_offspring)
    % Menggabungkan populasi lama dan keturunan baru
    combined_population = [population, new_offspring];
    % Menghitung fitness untuk populasi gabungan
    fitness_values = arrayfun(@evaluate_fitness, combined_population);
    % Mengurutkan berdasarkan fitness
    [~, sorted_indices] = sort(fitness_values);
    % Mengambil individu terbaik dari populasi baru
    updated_population = combined_population(sorted_indices(1:length(population)));
end

% Contoh populasi awal
population = [21.0, 24.0];
% Individu baru setelah crossover dan mutasi
new_offspring = [20.7, 22.5];

% Melakukan penggantian
updated_population = replacement(population, new_offspring);

% Menampilkan hasil
disp('Populasi Setelah Penggantian:');
disp(updated_population);
```

Hasil Program Penggantian

Populasi Setelah Penggantian: [20.7, 21.0]

Tahap-Tahap Genetic Algorithm:

1. Inisialisasi Populasi
2. Evaluasi Fitness
3. Seleksi
4. Crossover (Rekombinasi)
5. Mutasi
6. Penggantian
7. Pengulangan

7. Pengulangan

- Setelah tahap penggantian dalam algoritma genetik, proses berlanjut ke pengulangan.
- Bagian ini penting karena algoritma genetik dirancang untuk beroperasi dalam beberapa generasi untuk secara bertahap menemukan solusi yang lebih baik.
- Proses pengulangan melibatkan kembali langkah **Evaluasi Fitness, Seleksi, Crossover, Mutasi, dan Penggantian** dalam siklus algoritma genetik hingga kriteria penghentian tertentu tercapai.