LAPORAN TUGAS BESAR

Diajukan Untuk Memenuhi Salah Satu Tugas Mata Kuliah Kecerdasan Buatan

Disusun oleh:

1. Nikita Farah Andrea (1305220013)
2. Chacha Alisha Dewintasari (1305220036)
3. Jason Kusuma (1305220095)

Program Studi Sains Data
Fakultas Informatika
Universitas Telkom

2023

KATA PENGANTAR

Assalamu'alaikum Wr. Wb.

Dengan rasa syukur dan hormat, kami sebagai mahasiswa Program Studi Sains Data mengucapkan salam yang tulus. Sebagai bagian dari perkuliahan mata kuliah Kecerdasan Buatan, kami dengan penuh antusiasme ingin mempersembahkan laporan tugas besar kami mengenai Penerapan Algoritma Genetika dalam Menentukan Nilai Optimasi.

Laporan ini disusun sebagai salah satu syarat untuk melengkapi penilaian tugas mata kuliah Kecerdasan Buatan di program studi Sains Data Fakultas Informatika Universitas Telkom.

Pada laporan ini kami akan membahas mengenai teori dasar algoritma genetika, faktor-faktor yang mempengaruhi kinerja algoritma genetika,penerapannya untuk menyelesaikan masalah optimasi non linear, serta metode-metode yang kami gunakan di dalam algoritma genetika . Pada akhir laporan kami harap dapat memberikan solusi yang optimal untuk masalah optimasi yang diberikan.

Kami menyadari bahwa laporan tugas besar ini masih jauh dari kata sempurna, dan begitu banyak kekurangan. Oleh karena itu, kami mengharapkan adanya kritik dan saran yang membangun dari para pembaca, untuk perbaikan laporan di masa yang akan datang.

Akhir kata, kami berharap laporan tugas besar ini dapat memberikan manfaat dan kontribusi yang berarti bagi perkembangan ilmu kecerdasan buatan dan aplikasinya dalam pemecahan masalah dunia nyata.

Wassalamu'alaikum Wr. Wb.

Sabtu, 3 November 2023

Penyusun

BAB I
PENDAHULUAN

1.1 Latar Belakang

Algoritma Genetika merupakan sebuah algoritma yang digunakan untuk menemukan solusi optimal dari suatu permasalahan yang memiliki banyak solusi. Algoritma ini menerapkan prinsip pada proses seleksi alam dan genetika pada makhluk hidup untuk menyelesaikan permasalahan. Algoritma ini pertama kali dikembangkan oleh John Holland dari Amerika Serikat yang kemudian dipublikasikan dalam bukunya yang berjudul "Adaptation in Natural and Artificial System".

Pada algoritma ini terdapat populasi yang terdiri dari beberapa individu yang masing masing mempresentasikan sebuah solusi yang mungkin bagi sebuah permasalahan, individu - individu tersebut akan diseleksi untuk mencapai nilai terbaik dengan kriteria yang ditentukan dalam fungsi fitness, pada algoritma ini terdapat tiga operasi genetik yang digunakan yaitu seleksi, crossover, dan mutasi.

Proses dalam mencari solusi terbaik dalam algoritma genetika dimulai dengan membuat representasi dari solusi-solusi yang mungkin. Representasi ini bisa berupa string biner ataupun representasi lain yang sesuai dengan domain masalah. Dari representasi tersebut, kemudian akan dibentuk populasi individu secara acak, kemudian setiap individu akan dievaluasi dengan fitness function untuk menentukan individu terbaik. Individu dengan nilai fitness terbaik selanjutnya akan dipilih untuk menghasilkan keturunan baru, keturunan baru inilah yang kemudian akan mengalami mutasi dan rekombinasi untuk menghasilkan populasi baru. Proses ini akan terus berulang hingga individu dari populasi mencapai nilai terbaik sesuai dengan nilai fitness yang diinginkan.

Aplikasi algoritma genetika biasanya ditemukan pada penyelesaian masalah-masalah yang membutuhkan solusi kombinatorik seperti masalah optimasi linear dan nonlinear, contohnya seperti penjadwalan, peramalan, dan penentuan jarak terpendek

Dalam tugas besar ini, kami menerapkan aplikasi algoritma genetika untuk menyelesaikan masalah optimasi non linear untuk mendapatkan nilai dari kromosom terbaik dan nilai minimum dari kromosom tersebut berdasarkan fungsi fitness yang telah diberikan, untuk mencapai nilai tersebut kami menggunakan fungsi-fungsi dalam algoritma genetika, yaitu inisialisasi populasi, seleksi orangtua, crossover, dan mutasi. Dengan menggunakan fungsi-fungsi tersebut diharapkan mendapat solusi ideal yang memenuhi batasan yang telah diberikan.

1.2 Rumusan Masalah

Berikut adalah rumusan masalah yang akan dibahas:

1. Ukuran populasi, rancangan kromosom, dan cara decode-nya
2. Metode pemilihan orangtua
3. Metode operasi genetic
4. Probabilitas operasi genetic
5. Metode pergantian generasi
6. Kriteria penghentian evolusi

1.3 Tujuan

Dapat menemukan solusi optimal dalam konteks permasalah yang diberikan menggunakan *Genetic Algorithm* dan dapat:

1. Menentukan populasi dan rancangan kromosom yang optimal
2. Menentukan pemilihan orangtua yang optimal
3. Menentukan operasi genetic dan probabilitasnya
4. Menentukan metode pergantian generasi
5. Menentukan kriteria penghentian evolusi

1.4 Manfaat

Dengan mempelajari metode algoritma genetika diharapkan dapat menambah wawasan mengenai proses algoritma genetika dan memberi pemahaman mengenai proses dalam mencari fitness point.

BAB II
PERANCANGAN

Untuk menemukan solusi dari permasalahan yang kita miliki, yaitu mencari nilai minimum dari fungsi matematis:

$$f(x_1, x_2) = -(sin(x_1)cos(x_2) + \frac{4}{5}exp(1 - \sqrt{x_1^2 + x_2^2}))$$

Dengan batas domain(range) yang diberikan untuk x_1 dan x_2 adalah
$-10 \leq x_1 \leq 10$ dan $-10 \leq x_2 \leq 10$

Ada beberapa hal yang harus kami analisis dan membuat perancangan untuk dapat memecahkan persoalan, yaitu

2.1 Ukuran Populasi

Ukuran populasi adalah parameter yang menentukan jumlah individu dalam setiap generasi algoritma genetika. Ukuran populasi yang lebih besar cenderung meningkatkan kemampuan algoritma untuk mengeksplorasi ruang pencarian. Hal ini disebabkan oleh fakta bahwa populasi yang lebih besar melibatkan lebih banyak individu dalam proses pencarian, yang pada gilirannya dapat menciptakan variasi genetik yang lebih besar dan mengeksplorasi berbagai solusi potensial. Di sisi lain, ukuran populasi yang lebih kecil cenderung lebih fokus pada eksploitasi, yaitu meningkatkan kualitas individu dalam populasi saat ini.

Dari pertimbangan hal yang disebutkan sebelumnya, dalam program yang kami buat, kami menginisialisasi ukuran populasi menjadi 50 karena kami ingin mengeksplorasi berbagai solusi potensial dengan melibatkan banyak individu dalam proses pencarian

2.2 Rancangan Kromosom

Rancangan kromosom yang kami gunakan disini adalah string biner dengan panjang 20 gen, setengah bagian dari kromosom digunakan untuk merepresentasikan nilai x1 dan setengahnya lagi untuk x2 sesuai dengan batas domain yang telah ditentukan.

Alasan kami memilih rancangan kromosom dengan string biner adalah string biner memiliki representasi universal yang dapat digunakan untuk mengkodekan berbagai jenis masalah optimasi, kemudahan operasi genetik seperti crossover dan mutasi, string biner juga mudah diinterpretasikan oleh manusia.

2.3 Dekode Kromosom

Kami melakukan konversi kromosom dari string biner menjadi nilai bilangan real x1 dan x2 sesuai dengan batas domain yang telah ditentukan yaitu antara -10 sampai 10.

2.4 Metode Pemilihan Orang Tua

Kami menggunakan metode turnamen dalam proses pemilihan orang tua. Metode ini dipilih karena dengan menggunakan turnamen, kami dapat mengontrol jumlah peserta dalam turnamen tersebut, sehingga kami dapat mengatur seberapa kompetitif proses pemilihan orang tua. Metode turnamen juga memilih orang tua berdasarkan nilai kecocokan, atau dalam konteks program yang kami buat, nilai maksimum dan minimum dari nilai fitness. Dengan cara ini, kami dapat mengendalikan output orang tua yang ingin dihasilkan.

2.5 Metode Operasi Genetik (crossover dan mutasi)

Kami menggunakan pindah silang (crossover) satu titik untuk menghasilkan anak anak dari pasangan orang tua yang terpilih. Terdapat *probability crossover* untuk mengontrol sejauh mana rekombinasi atau crossover diterapkan. Nilai *probability crossover* yang tinggi cenderung mendukung eksploitasi dan mempercepat konvergensi genetik ke solusi yang lebih cepat, sedangkan nilai *probability crossover* yang rendah cenderung mendukung eksplorasi dan memperlambat konvergensi. Pada program kami, kami menginisialisasi *probability crossover* dengan nilai 0.8 karena kami ingin mempercepat konvergensi dan mempertahankan gen gen baik dari orang tua.

Dilakukannya mutasi setelah *crossover* bertujuan untuk memberikan variasi genetik yang lebih acak ke dalam populasi. *Probability mutation* disini adalah parameter untuk mendefinisikan kemungkinan bahwa setiap gen dalam kromosom akan mengalami mutasi. Pada program ini kami menginisialisasi *probability mutation* dengan nilai 0.01 agar gen dalam kromosom tidak terlalu banyak mengalami mutasi sehingga gen asli dari orang tua masih terjaga.

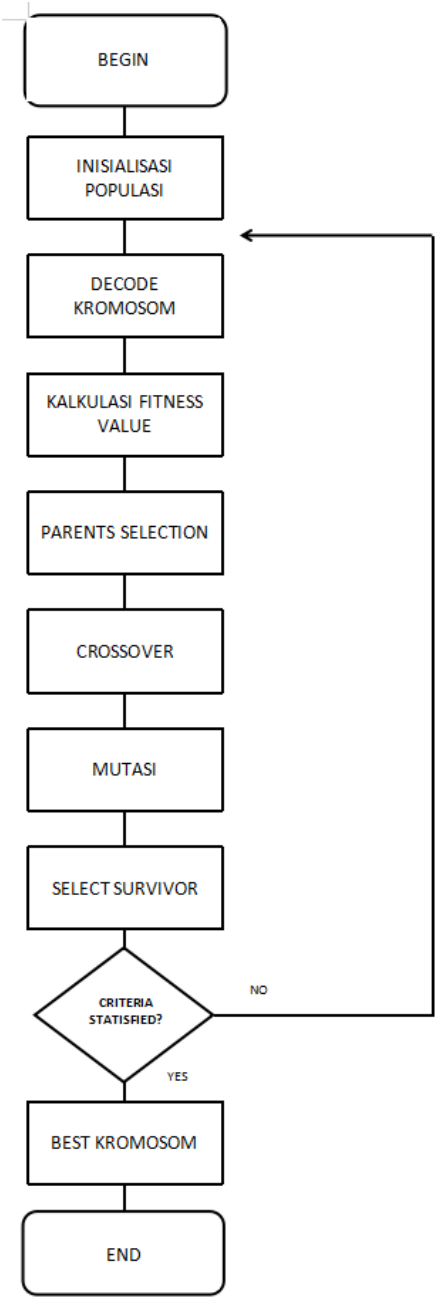
2.6 Pergantian Generasi

Kami menggunakan metode seleksi survivor untuk memilih individu yang akan bertahan dengan membentuk populasi generasi berikutnya. Kami memilih individu dengan nilai fitness tertinggi untuk bertahan dan sisanya akan terbuang dari populasi.

2.7 Penghentian Evolusi

Kriteria pengertian evolusi dalam program kami diatur berdasarkan jumlah generasi maksimum yang telah kita atur menjadi 100 generasi, angka 100 generasi

disini merupakan angka yang besar sehingga diharapkan ruang pencarian bisa semakin luas dan memungkinkan penjelajahan individu yang lebih banyak.
Berikut adalah bentuk dari langkah langkah yang dilakukan dalam *Genetic Algorithm*:



BAB III
IMPLEMENTASI

1. Import library

Kami menggunakan modul “random” dan modul “math” yang merupakan modul

```
[ ]  import random
     import math
```

bagian dari library standar python

3. Mengisi parameter GA (Algoritma Genetika)

```
[ ]  pop_size = 50
     chrom_length = 10
     x1_min, x1_max = -10, 10
     x2_min, x2_max = -10, 10
     max_generations = 100
     pm = 0.01
     pc = 0.8
```

4. Menginisialisasi populasi

```
[ ]  def initialize_population(pop_size, chrom_length):
         population = []
         for _ in range(pop_size):
             chromosome = ''.join(random.choice('01') for _ in range(chrom_length))
             population.append(chromosome)
         return population
```

Kode di atas bertujuan untuk menginisialisasi nilai populasi sesuai dengan ukuran populasi (pop_size) dan panjang kromosom (chrom_length) untuk setiap individu dalam populasi. Isi dari nilai kromosom disini adalah nilai biner. Fungsi ini akan melakukan perulangan sebanyak ‘pop_size’ dan mengembalikan populasi yang berisi kromosom-kromosom yang telah diinisialisasi.

5. Mengkonversi biner ke real value

```
[ ]  def decode_chromosome(chromosome, x1_min, x1_max, x2_min, x2_max):
         chrom_length = len(chromosome)
         x1_range = x1_max - x1_min
         x2_range = x2_max - x2_min
         x1_value = x1_min + int(chromosome[:chrom_length // 2], 2) * (x1_range / (2**(chrom_length // 2) - 1))
         x2_value = x2_min + int(chromosome[chrom_length // 2:], 2) * (x2_range / (2**(chrom_length // 2) - 1))
         return x1_value, x2_value
```

Kode di atas bertujuan untuk merubah representasi kromosom dalam bentuk biner menjadi nilai sesungguhnya (real value) sesuai dengan domain yang telah ditentukan. Fungsi decode_chromosome(chromosome, x1_min, x1_max, x2_min, x2_max) akan mengembalikan dua nilai x1 dan x2 yang telah dikonversikan.

6. Fungsi fitness

```
[ ] def function(x1, x2):
        return -(math.sin(x1) * math.cos(x2) + 4/5 * math.exp(1 - math.sqrt(x1**2 + x2**2)))
```

```
[ ] #minimize fitness function
    def calculate_fitness(x1, x2):
        return 1 / function(x1,x2)
```

Pada kode di atas, function(x1, x2) adalah fungsi matematis yang mengambil dua variabel ‘x1’ dan ‘x2’ dan menghitung nilai dari fungsi tersebut sesuai dengan rumus yang telah diberikan. Sedangkan, calculate_fitness(x1, x2) adalah fungsi yang digunakan untuk menghitung nilai fitness yang diambil nilai minimumnya.

7. Seleksi orangtua

```
[16] def select_parents_tournament(population, fitness_values):
         tournament_size = 5
         parents = []
         for _ in range(2):
             tournament = random.sample(range(len(population)), tournament_size)
             tournament_fitness = [fitness_values[i] for i in tournament]
             winner = tournament[tournament_fitness.index(max(tournament_fitness))]
             parents.append(population[winner])
         return parents
```

Pada kode diatas, akan dilakukan perulangan sebanyak dua kali untuk memilih dua orang tua. Dalam setiap putaran turnamen, sejumlah `tournament_size` indeks individu yang akan berpartisipasi dalam turnamen dipilih secara acak dari populasi. Fungsi `random.sample` digunakan untuk menghasilkan indeks-indeks ini tanpa pengulangan. Nilai fitness dari peserta turnamen dihitung dan disimpan dalam daftar `tournament_fitness`. Peserta turnamen dengan nilai fitness tertinggi ditentukan sebagai pemenang turnamen. Dalam kode ini, `max(tournament_fitness)` digunakan untuk menemukan nilai fitness tertinggi dalam turnamen. Fungsi ini akan mengembalikan daftar parents yang berisi dua orang tua yang telah dipilih menggunakan seleksi metode turnamen.

8. Crossover

```
[ ] # Single-point crossover
    def crossover(parent1, parent2):
        if random.random() < pc:
            crossover_point = random.randint(1, len(parent1) - 1)
            child1 = parent1[:crossover_point] + parent2[crossover_point:]
            child2 = parent2[:crossover_point] + parent1[crossover_point:]
            return child1, child2
        else:
            return parent1, parent2
```

Pada kode di atas, akan dibandingkan terlebih dahulu nilai random yang diambil dari random.random() dengan pc (probabilitas crossover). Apabila nilai random.random() kurang dari nilai pc maka akan dilakukan persilangan antara parent1 dengan parent 2 dan menghasilkan child 1 dan child 2, namun jika nilai

random.random() tidak kurang dari nilai pc maka tidak akan dilakukan persilangan.

9. Mutasi

```
[ ] def mutate(chromosome, mutation_rate):
        mutated_chromosome = list(chromosome)
        for i in range(len(mutated_chromosome)):
            if random.random() < mutation_rate:
                mutated_chromosome[i] = '0' if chromosome[i] == '1' else '1'
        return ''.join(mutated_chromosome)
```

Pada kode di atas, kromosom awal akan disalin ke dalam mutated_chromosome, hal ini dilakukan untuk merubah string kromosom menjadi list karakter sehingga memudahkan dalam memodifikasi karakter-karakternya, lalu akan dilakukan perulangan. Di dalam perulangan akan dibandingkan nilai acak yang diambil dari random.random() dengan probabilitas mutasi (mutation_rate), apabila nilai random.random() kurang dari mutation_rate maka mutasi akan dilakukan yaitu dengan merubah nilai karakternya, jika karakternya adalah '1' maka akan diubah menjadi '0' dan sebaliknya. Setelah semua elemen kromosom diproses, kromosom yang telah dimutasi dikonversi Kembali menjadi string, dan akan dikembalikan sebagai hasil mutasi.

10. Pergantian Generasi (Select survivor)

```
[ ] def select_survivors(population, fitness_values, pop_size):
        combined_population = list(zip(population, fitness_values))
        combined_population.sort(key=lambda x: x[1], reverse=False)
        return [individual for individual, _ in combined_population[:pop_size]]
```

Pada kode di atas, diambil 3 parameter yaitu population (populasi saat ini), fitness_values (nilai fitness dari individu-individu dalam populasi), pop_size(ukuran populasi yang ingin dijaga). Combined_population merupakan penggabungan dua list, yaitu 'population' dan 'fitness_value' menggunakan fungsi zip. Kemudian akan dilakukan sorting berdasarkan nilai fitness, pengurutan akan dilakukan dalam urutan menaik yang berarti individu dengan nilai terbaik akan berada di bagian awal. Fungsi ini akan mengembalikan daftar individu yang akan bertahan dan menjadi bagian dari generasi berikutnya.

11. Main program

```python
population = initialize_population(pop_size, chrom_length)

# Loopin sampai generasi yang ditentukan
for generation in range(max_generations):
    fitness_values = []
    new_population = []

    # Evaluasi fitness populasi
    for chromosome in population:
        x1, x2 = decode_chromosome(chromosome, x1_min, x1_max, x2_min, x2_max)
        fitness = calculate_fitness(x1, x2)
        fitness_values.append(fitness)

    # Pemilihan orangtua dan crossover
    while len(new_population) < pop_size:
        parents = select_parents_tournament(population, fitness_values)
        child1, child2 = crossover(parents[0], parents[1])
        child1 = mutate(child1, pm)
        child2 = mutate(child2, pm)
        new_population.extend([child1, child2])

    # Pergantian generasi
    population = select_survivors(new_population, fitness_values, pop_size)

# Mendapatkan hasil terbaik
best_chromosome = max(population, key=lambda chromosome: calculate_fitness(*decode_chromosome(chromosome, x1_min, x1_max, x2_min, x2_max)))
x1_best, x2_best = decode_chromosome(best_chromosome, x1_min, x1_max, x2_min, x2_max)
max_value = calculate_fitness(x1_best, x2_best)

print(f"Hasil fitness kromosom terbaik: f(x1, x2) = {max_value} pada x1 = {x1_best}, x2 = {x2_best}")
```

```
Hasil fitness kromosom terbaik: f(x1, x2) = 381.9995219366825 pada x1 = -6.129032258064516, x2 = -1.612903225806452
```

Kode di atas merupakan kerangka kerja untuk melakukan optimasi menggunakan algoritma genetika.

Tahap pertama adalah menginisialisasi populasi, lalu akan dilakukan iterasi sebanyak ‘max_generations’, iterasi ini memungkinkan algoritma genetika beroperasi selama sejumlah generasi tersebut. Selama iterasi tersebut akan dilakukan evaluasi fitness, pemilihan orangtua, crossover, mutasi, dan pergantian generasi. Setelah iterasi berakhir akan dilakukan pencarian kromosom terbaik, yaitu dengan menggunakan fungsi max untuk mencari kromosom terbaik dalam populasi saat ini dan membandingkan kromosom-kromosom dalam populasi berdasarkan nilai fitness. Sedangkan fungsi key digunakan untuk menentukan kriteria perbandingan, pada kode ini, kriteria perbandingan adalah nilai fitness dari kromosom, jadi kromosom dengan nilai fitness tertinggi akan menjadi ‘best_chromosome’.

BAB IV
KESIMPULAN

4.1 Analisis Kinerja pada tiap Generasi

Kinerja Algoritma tiap generasi yang kami lakukan biasanya terjadi perkembang nilai fitness seiring berjalannya generasi, tetapi beberapa percobaan yang kami lakukan tidak menghasilkan perkembangan atau hasil di tiap generasi berubah ubah naik turun. Berdasarkan hasil analisis kami kinerja algoritma tiap generasi berjalan sebagai berikut:

1. Generasi Awal
   Pada generasi awal, populasi biasanya terdiri dari individu yang dipilih secara acak. Kinerja awal mungkin buruk, dan solusi yang dihasilkan mungkin jauh dari nilai optimal.
2. Generasi Pertengahan
   Seiring berjalannya waktu, algoritma genetika dapat mengeksplorasi berbagai kombinasi genetik melalui pindah silang dan mutasi. Populasi dapat mulai meng konvergen ke solusi yang lebih baik. Nilai fitness setiap gen dalam populasi mungkin mulai meningkat atau menurun.
3. Generasi Terakhir:
   Generasi terakhir cenderung mencapai tahap konvergensi maksimum. Populasi mendekati solusi yang optimal atau setidaknya solusi yang sangat baik.

Faktor yang mungkin menjadi pengaruh:

1. Ukuran Populasi:
   Ukuran populasi yang lebih besar memiliki potensi untuk melakukan eksplorasi yang lebih baik, tetapi memerlukan lebih banyak perhitungan. Ukuran populasi yang terlalu kecil mungkin tidak cukup untuk mencapai konvergensi yang baik.
2. Probabilitas Mutasi dan Crossover:
   Tingkat probabilitas mutasi dan probabilitas pindah silang (crossover) mempengaruhi seberapa sering variasi genetik diperkenalkan dalam populasi. Pengaturan yang tepat dapat mempercepat atau memperlambat konvergensi.
3. Seleksi Orangtua:
   Metode seleksi orang tua, seperti metode turnamen atau roulette wheel selection, dapat memengaruhi sejauh mana individu-individu yang lebih baik dapat mewariskan informasi genetik mereka.
4. Nilai Maximum generasi:
   Nilai maximum generasi yang terlalu kecil dapat mempengaruhi hasil akhir karena ruang pencarian menjadi lebih kecil dan kurang bisa menjangkau individu yang beragam.
5. Kualitas Inisialisasi Populasi:
   Kualitas populasi awal yang buruk juga alat memperlambat konvergensi karena populasi awal yang akan berpeluang besar menentukan gen generasi selanjutnya

4.2 Kesimpulan yang kami dapatkan

1. Algoritma genetika cenderung konvergen ke solusi yang lebih baik seiring berjalannya generasi. Hal ini dapat dilihat dari peningkatan nilai fitness rata-rata dalam populasi.
2. Pengaturan parameter algoritma genetika, seperti ukuran populasi, probabilitas mutasi, probabilitas pindah silang (crossover), dan metode seleksi orang tua, memiliki pengaruh signifikan terhadap kinerja algoritma. Pengujian dan penyetelan parameter adalah langkah penting dalam mencapai hasil yang optimal.
3. Nilai fitness dalam populasi dapat mengalami fluktuasi. Ini adalah bagian alami dari eksplorasi ruang pencarian dan mencoba menghindari solusi lokal yang buruk.
4. Kriteria penghentian yang ditentukan dalam algoritma genetika mempengaruhi hasil akhir. Jika algoritma berhenti terlalu dini, solusi mungkin belum optimal. Sebaliknya, jika algoritma berjalan terlalu lama, waktu komputasi dapat meningkat.
5. Kualitas populasi awal yang diinisialisasi dapat mempengaruhi kinerja algoritma. Populasi awal yang berkualitas buruk mungkin memerlukan waktu lebih lama untuk mencapai hasil yang baik.

PERAN ANGGOTA KELOMPOK

NO	Nama Anggota Kelompok	Peran
1.	Chacha Alisha Dewintasari	Mengerjakan source code, membuat BAB II, membuat Kesimpulan
2.	Nikita Farah Andrea	Mengerjakan source code, membuat BAB III
3.	Jason Kusuma	Membuat Kata Pengantar, BAB I, dan slide presentasi.