Tugas 7 Algoritma Genetika
Praktikum Kecerdasan Buatan

Oleh: Athallah Tsany Satriyaji (H1D023013)

TSPInisialisasiPopulasi.m

```m
function Populasi = TSPInisialisasiPopulasi (UkPop, JumGen)
for ii=1:UkPop,
    [Xval, Ind] = sort(rand(1,JumGen));
    Populasi(ii,:) = Ind;
end
```

Potongan kode MATLAB di atas merupakan fungsi yang digunakan untuk menginisialisasi populasi awal dalam algoritma genetika untuk menyelesaikan permasalahan Travelling Salesman Problem (TSP). Fungsi ini bernama TSPInisialisasiPopulasi dan menerima dua parameter: UkPop, yaitu ukuran populasi (jumlah individu atau solusi yang akan dibuat), dan JumGen, yaitu jumlah gen dalam setiap individu (jumlah kota yang harus dikunjungi dalam TSP). Di dalam fungsi, terdapat perulangan for sebanyak UkPop kali, yang berarti fungsi akan membuat sebanyak UkPop individu. Pada setiap iterasi, fungsi menggunakan rand(1,JumGen) untuk menghasilkan sebuah vektor acak berisi angka antara 0 dan 1 sebanyak JumGen elemen. Kemudian, fungsi sort digunakan untuk mengurutkan angka-angka acak tersebut, dan indeks hasil pengurutan (Ind) diambil sebagai representasi dari urutan kunjungan kota dalam individu tersebut. Karena angka-angka acak diurutkan berdasarkan nilainya, indeks yang diperoleh akan selalu menjadi permutasi unik dari angka 1 hingga JumGen, yang memastikan bahwa setiap kota hanya dikunjungi satu kali. Akhirnya, individu ini disimpan ke dalam matriks Populasi, dengan baris ke-ii berisi satu solusi kandidat. Hasil akhir dari fungsi ini adalah matriks Populasi yang berisi kumpulan solusi awal acak untuk TSP.

TSPEvaluasiIndividu.m

```m
function fitness = TSPEvaluasiIndividu(Kromosom,JumGen,XYkota)
TB = 0
for ii=1:JumGen-1,
    TB = TB + norm(XYkota (Kromosom(ii),:) - XYkota(Kromosom(ii+1),:));
end

% Jalur harus kembali ke kota asal
TB = TB + norm(XYkota(Kromosom(JumGen),:) - XYkota(Kromosom(1),:));
fitness = 1/TB;
```

Fungsi `TSPEvaluasiIndividu` merupakan bagian dari algoritma genetika untuk menyelesaikan masalah Travelling Salesman Problem (TSP), yang bertugas mengevaluasi kualitas atau fitness dari satu individu atau solusi. Fungsi ini menerima tiga parameter: `Kromosom`, yang merupakan array berisi urutan kunjungan kota (sebuah permutasi dari indeks kota), `JumGen`, yaitu jumlah kota yang ada, dan `XYkota`, yaitu matriks koordinat dari setiap kota dengan format dua kolom untuk menyatakan posisi x dan y. Dalam fungsi ini, variabel `TB` (Total Biaya atau jarak total) diinisialisasi ke nol, kemudian dilakukan perulangan dari kota pertama hingga kota kedua terakhir. Pada setiap iterasi, dihitung jarak Euclidean antara kota ke-`ii` dan kota ke-`ii+1` menggunakan fungsi `norm`, dan jarak tersebut dijumlahkan ke dalam `TB`. Setelah itu, karena

dalam TSP jalur harus kembali ke kota asal, maka ditambahkan juga jarak dari kota terakhir kembali ke kota pertama. Akhirnya, fitness dari kromosom tersebut dihitung sebagai kebalikan dari total jarak (`1/TB`), sehingga semakin pendek jarak perjalanan total, semakin besar nilai fitness-nya.

LinearFitnessRanking.m

```
function LFR = LinearFitnessRanking(UkPop,Fitness,MaxF,MinF)

 [SF,IndF] = sort(Fitness);

for rr=1:UkPop,
    LFR(IndF(UkPop-rr+1)) = MaxF - (MaxF - MinF) * ((rr-1) / (UkPop-1));
end
```

Fungsi `LinearFitnessRanking` ini digunakan untuk melakukan proses ranking fitness dalam algoritma genetika, khususnya dengan pendekatan Linear Fitness Ranking. Fungsi ini menerima empat parameter: `UkPop`, yaitu ukuran populasi; `Fitness`, sebuah array berisi nilai fitness dari setiap individu dalam populasi; `MaxF`, yaitu nilai fitness maksimum yang ingin diberikan kepada individu terbaik; dan `MinF`, yaitu nilai fitness minimum yang akan diberikan kepada individu terburuk. Di dalam fungsi, terlebih dahulu dilakukan pengurutan nilai `Fitness` secara ascending menggunakan `sort`, sehingga individu dengan fitness terkecil berada di awal. Fungsi `sort` mengembalikan dua hal: `SF`, yaitu nilai fitness yang telah diurutkan, dan `IndF`, yaitu indeks asli dari individu-individu tersebut sebelum diurutkan. Setelah itu, dilakukan perulangan sebanyak jumlah populasi. Pada setiap iterasi, nilai fitness baru `LFR` (Linear Fitness Ranking) diberikan kepada individu dengan peringkat ke-`rr` (dimulai dari yang terbaik, karena indexing dibalik menggunakan `UkPop - rr + 1`). Nilai fitness baru ini dihitung secara linear berdasarkan posisi peringkat: individu terbaik mendapat `MaxF`, yang terburuk mendapat `MinF`, dan sisanya didistribusikan secara linier di antaranya.

RouletteWheel.m

```
function Pindex = RouletteWheel(UkPop,LinearFitness);

JumFitness = sum(LinearFitness);
KumulatifFitness = 0;
RN = rand;
ii = 1;

while ii <= UkPop
    KumulatifFitness = KumulatifFitness + LinearFitness(ii);
    if(KumulatifFitness/JumFitness) > RN,
        Pindex = ii;
        break;
    end
    ii = ii+1;
end
```

Fungsi RouletteWheel digunakan untuk memilih satu individu dari populasi berdasarkan probabilitas proporsional terhadap nilai fitness-nya. Fungsi ini menghitung total fitness (JumFitness) dari seluruh individu, lalu membangkitkan angka acak RN antara 0 dan 1. Kemudian, fungsi menjumlahkan nilai fitness secara kumulatif hingga proporsi kumulatif tersebut melebihi RN. Individu yang membuat kumulatif melebihi nilai acak tersebut akan dipilih, dan indeksnya dikembalikan sebagai Pindex.

TSPPindahSilang.m

```
function Anak = TSPPindahSilang(Bapak,Ibu,JumGen)

cp1 = 1+fix(rand*(JumGen-1));
cp2 = 1+fix(rand*(JumGen-1));

while cp2==cp1,
    cp2 = 1+fix(rand*(JumGen-1));
end

if cp1<cp2,
    cps=cp1;
    cpd=cp1;
else
    cps=cp2;
    cpd=cp1;
end
Anak(1,cps+1:cpd) = Ibu(cps+1:cpd);
Anak(2,cps+1:cpd) = Bapak(cps+1:cpd);

SisaGenBapak = [];
SisaGenIbu = [];

for ii=1:JumGen,
    if ~ismember(Bapak(ii), Anak(1,:)),
        SisaGenBapak = [SisaGenBapak Bapak(ii)];
    end
    if ~ismember(Ibu(ii), Anak(2,:)),
        SisaGenIbu = [SisaGenIbu Ibu(ii)];
    end
end

Anak(1,cpd+1:JumGen) = SisaGenBapak(1:JumGen-cpd);
Anak(1,1:cps) = SisaGenBapak(1+JumGen-cpd:length(SisaGenBapak));

Anak(2,cpd+1:JumGen) = SisaGenIbu(1:JumGen-cpd);
Anak(2,1:cps) = SisaGenIbu(1+JumGen-cpd:length(SisaGenIbu));
```

Fungsi TSPPindahSilang ini mengimplementasikan operator crossover khusus untuk algoritma genetika pada masalah Travelling Salesman Problem (TSP), dengan tujuan menghasilkan dua anak dari dua induk (Bapak dan Ibu). Pertama, fungsi memilih dua titik potong acak (cp1 dan cp2) yang berbeda sebagai batas segmen crossover. Selanjutnya, bagian segmen antara kedua titik potong ini dari kromosom induk Ibu disalin ke anak pertama, dan segmen yang sama dari kromosom Bapak disalin ke anak kedua. Setelah itu, fungsi mengumpulkan sisa gen yang belum ada pada bagian anak tersebut dengan memeriksa gen yang tidak ada dalam segmen yang sudah disalin, menjaga agar tidak ada duplikasi gen karena setiap kota harus unik dalam rute TSP. Kemudian, sisa gen ini ditempatkan melingkar ke posisi yang tersisa di kromosom anak, sehingga kromosom hasil crossover tetap berupa permutasi valid tanpa mengulang kota. Hasil akhirnya adalah dua anak baru yang menggabungkan informasi dari kedua induk sambil mempertahankan validitas solusi untuk TSP.

TSPMutasi.m

```
function MutKrom = TSPMutasi (Kromosom,JumGen,Pmutasi)
MutKrom = Kromosom;

for ii=1:JumGen,
    if rand < Pmutasi,
        TM2 = 1 + fix(rand*JumGen);
        while TM2==ii,
            TM2 = 1 + fix(rand*JumGen);
        end
        temp = MutKrom(ii);
        MutKrom(ii) = MutKrom(TM2);
        MutKrom(TM2) = temp;
    end
end
```

Fungsi `TSPMutasi` ini melakukan proses mutasi pada sebuah kromosom dalam konteks algoritma genetika untuk Travelling Salesman Problem (TSP). Fungsi menerima kromosom awal, jumlah gen (`JumGen`), dan probabilitas mutasi (`Pmutasi`). Pada setiap gen dalam kromosom, fungsi melakukan pengecekan apakah akan terjadi mutasi berdasarkan nilai acak yang

dibandingkan dengan `Pmutasi`. Jika mutasi terjadi, fungsi memilih posisi gen lain secara acak yang berbeda dari posisi saat ini, kemudian menukar posisi kedua gen tersebut dalam kromosom. Dengan menukar gen ini, fungsi menghasilkan variasi baru dalam solusi tanpa menambah atau menghapus gen, sehingga memastikan bahwa setiap kota tetap unik dan tetap merupakan solusi valid untuk TSP.

AlgenMain.m

```
clc;
clear all;

% Koordinat kota
XYkota = [1 3; 1 7; 3 9; 5 3; 7 1; 9 5; 9 9; 11 1; 15 7; 19 3];

% Parameter algoritma
JumGen = length(XYkota(:,1));
UkPop = 100;
Psilang = 0.8;
Pmutasi = 0.005;
MaxG = 100;

PanjJalHarp = 40;
Fthreshold = 1 / PanjJalHarp;
Bgraf = Fthreshold;

% Inisialisasi grafik
hfig = figure;
hold on
set(hfig, 'Position', [50, 50, 600, 400]);
set(hfig, 'DoubleBuffer', 'on');
axis([1 MaxG 0 Bgraf]);
hbestplot1 = plot(1:MaxG, zeros(1, MaxG));
hbestplot2 = plot(1:MaxG, zeros(1, MaxG));
htext1 = text(0.6 * MaxG, 0.25 * Bgraf, sprintf('Fitness terbaik: %7.6f', 0.0));
htext2 = text(0.6 * MaxG, 0.20 * Bgraf, sprintf('Fitness rata-rata: %7.6f', 0.0));
htext3 = text(0.6 * MaxG, 0.15 * Bgraf, sprintf('Panjang jalur terbaik: %7.3f', 0.0));
htext4 = text(0.6 * MaxG, 0.10 * Bgraf, sprintf('Ukuran Populasi: %3.0f', UkPop));
htext5 = text(0.6 * MaxG, 0.05 * Bgraf, sprintf('Probabilitas Mutasi: %4.3f', Pmutasi));
xlabel('Generasi');
ylabel('Fitness');
hold off;
drawnow;

% Inisialisasi populasi awal
Populasi = TSPInisialisasiPopulasi(UkPop, JumGen);
```

Potongan kode utama di atas adalah bagian awal dari program MATLAB yang mengimplementasikan algoritma genetika untuk menyelesaikan masalah Travelling Salesman Problem (TSP). Di awal, `clc` dan `clear all` digunakan untuk membersihkan command window dan workspace agar lingkungan kerja bersih sebelum program berjalan. Selanjutnya, didefinisikan koordinat kota-kota yang akan dikunjungi dalam matriks `XYkota`, di mana setiap baris berisi pasangan koordinat (x,y) dari sebuah kota. Parameter penting algoritma juga didefinisikan, termasuk jumlah gen (`JumGen`) yang sama dengan jumlah kota, ukuran populasi (`UkPop`) yang menentukan berapa banyak solusi kandidat dibuat, serta probabilitas crossover (`Psilang`) dan mutasi (`Pmutasi`). `MaxG` adalah jumlah generasi maksimum yang akan dijalankan oleh algoritma.

Kemudian, ada pengaturan nilai ambang fitness (`Fthreshold`) berdasarkan panjang jalur harapan (`PanjJalHarp`) yang digunakan sebagai tolok ukur dalam evaluasi solusi. Bagian berikutnya menginisialisasi grafik untuk menampilkan perkembangan fitness selama iterasi algoritma, termasuk pengaturan ukuran dan posisi jendela grafik, axis, dan objek plot serta teks yang akan diperbarui selama proses. Grafik ini berfungsi sebagai visualisasi interaktif untuk memantau fitness terbaik, fitness rata-rata, panjang jalur terbaik, serta parameter algoritma seperti ukuran populasi dan probabilitas mutasi.

Terakhir, kode memanggil fungsi `TSPInisialisasiPopulasi` untuk membuat populasi awal solusi acak sebanyak `UkPop`, masing-masing berupa urutan kunjungan kota sepanjang `JumGen`. Populasi ini menjadi titik awal untuk evolusi solusi menggunakan algoritma genetika.

```
for generasi = 1:MaxG
    MaxF = TSPEvaluasiIndividu(Populasi(1,:), JumGen, XYkota);
    MinF = MaxF;
    IndeksIndividuTerbaik = 1;

    for ii = 1:UkPop
        Fitness(ii) = TSPEvaluasiIndividu(Populasi(ii,:), JumGen, XYkota);
        if Fitness(ii) > MaxF
            MaxF = Fitness(ii);
            IndeksIndividuTerbaik = ii;
            JalurTerbaik = Populasi(ii,:);
        end
        if Fitness(ii) < MinF
            MinF = Fitness(ii);
        end
    end

    FitnessRataRata = mean(Fitness);

    % Update grafik
    plotvector1 = get(hbestplot1, 'YData');
    plotvector1(generasi) = MaxF;
    set(hbestplot1, 'YData', plotvector1);

    plotvector2 = get(hbestplot2, 'YData');
    plotvector2(generasi) = FitnessRataRata;
    set(hbestplot2, 'YData', plotvector2);

    set(htext1, 'String', sprintf('Fitness Terbaik: %7.6f', MaxF));
    set(htext2, 'String', sprintf('Fitness rata-rata: %7.6f', FitnessRataRata));
    set(htext3, 'String', sprintf('Panjang jalur terbaik: %7.3f', 1 / MaxF));
    set(htext4, 'String', sprintf('Ukuran Populasi: %3.0f', UkPop));
    set(htext5, 'String', sprintf('Probabilitas Mutasi: %4.3f', Pmutasi));
    drawnow;

    if MaxF > Fthreshold
        break;
    end
```

Potongan kode looping di atas adalah inti dari proses evolusi dalam algoritma genetika untuk menyelesaikan masalah TSP. Pada setiap iterasi (generasi) dari 1 hingga `MaxG`, program melakukan evaluasi fitness untuk seluruh populasi yang ada. Pertama, nilai fitness individu pertama dijadikan acuan awal sebagai `MaxF` (fitness terbaik) dan `MinF` (fitness terendah), serta indeks individu terbaik diinisialisasi. Selanjutnya, kode melakukan perulangan pada setiap individu dalam populasi, menghitung fitness-nya menggunakan fungsi `TSPEvaluasiIndividu`. Jika ditemukan individu dengan fitness lebih baik dari `MaxF`, nilai tersebut dan indeksnya diperbarui, serta jalur terbaik saat ini disimpan. Sebaliknya, jika fitness lebih rendah dari `MinF`, maka nilai minimum juga diperbarui.

Setelah evaluasi semua individu, program menghitung rata-rata fitness populasi untuk mendapatkan gambaran umum kualitas solusi pada generasi tersebut. Kemudian, bagian berikutnya memperbarui grafik yang telah diinisialisasi sebelumnya, memperlihatkan perkembangan fitness terbaik dan rata-rata fitness sepanjang generasi yang berjalan. Informasi teks pada grafik juga diperbarui untuk menampilkan fitness terbaik, rata-rata fitness, panjang jalur terbaik (dihitung dari kebalikan fitness terbaik), ukuran populasi, dan probabilitas mutasi, sehingga pengguna dapat memantau performa algoritma secara real-time.

Terakhir, ada kondisi penghentian awal, yaitu jika fitness terbaik sudah melewati ambang fitness yang diinginkan (`Fthreshold`), maka proses evolusi dapat dihentikan lebih cepat dari jumlah generasi maksimum yang ditentukan.

```
    % Seleksi dan reproduksi
    TemPopulasi = Populasi;

    if mod(UkPop, 2) == 0
        IterasiMulai = 3;
        TemPopulasi(1,:) = Populasi(IndeksIndividuTerbaik,:);
        TemPopulasi(2,:) = Populasi(IndeksIndividuTerbaik,:);
    else
        IterasiMulai = 2;
        TemPopulasi(1,:) = Populasi(IndeksIndividuTerbaik,:);
    end

    LinearFitness = LinearFitnessRanking(UkPop, Fitness, MaxF, MinF);

    for jj = IterasiMulai:2:UkPop
        IP1 = RouletteWheel(UkPop, LinearFitness);
        IP2 = RouletteWheel(UkPop, LinearFitness);

        if rand < Psilang
            Anak = TSPPindahSilang(Populasi(IP1,:), Populasi(IP2,:), JumGen);
            TemPopulasi(jj,:) = Anak(1,:);
            TemPopulasi(jj+1,:) = Anak(2,:);
        else
            TemPopulasi(jj,:) = Populasi(IP1,:);
            TemPopulasi(jj+1,:) = Populasi(IP2,:);
        end
    end

    % Mutasi
    for kk = IterasiMulai:UkPop
        TemPopulasi(kk,:) = TSPMutasi(TemPopulasi(kk,:), JumGen, Pmutasi);
    end

    % Update populasi
    Populasi = TemPopulasi;
end

% Tampilkan jalur terbaik
JalurTerbaik

% Simpan hasil
save JalurTerbaik.mat JalurTerbaik
```

Potongan kode ini melanjutkan proses evolusi dalam algoritma genetika dengan langkah-langkah seleksi, reproduksi (crossover), mutasi, dan pembaruan populasi untuk setiap generasi. Pertama, dibuat salinan populasi sementara bernama `TemPopulasi` yang akan diisi dengan individu-individu generasi baru. Jika ukuran populasi (`UkPop`) genap, dua individu terbaik (dengan fitness tertinggi) langsung disalin ke dua posisi awal populasi baru sebagai elitisme, memastikan solusi terbaik tetap dipertahankan. Jika jumlah populasi ganjil, hanya satu individu terbaik yang dipertahankan di posisi pertama, dan seleksi serta reproduksi akan dimulai dari indeks kedua.

Selanjutnya, fitness dari seluruh populasi diubah menjadi bentuk linear menggunakan fungsi `LinearFitnessRanking` agar dapat digunakan dalam metode seleksi roulette wheel. Proses seleksi dilakukan secara berpasangan dengan memilih dua individu orang tua berdasarkan probabilitas fitness mereka. Jika probabilitas crossover (`Psilang`) terpenuhi, kedua orang tua akan disilangkan menggunakan fungsi `TSPPindahSilang` untuk menghasilkan dua anak baru yang kemudian dimasukkan ke populasi sementara. Jika tidak, anak tersebut merupakan salinan langsung dari orang tua yang dipilih.

Setelah proses reproduksi selesai untuk seluruh pasangan, populasi sementara menjalani mutasi melalui fungsi `TSPMutasi`, yang mengacak sebagian gen dalam kromosom dengan probabilitas tertentu (`Pmutasi`) untuk menjaga keragaman genetik dan menghindari konvergensi prematur pada solusi lokal.

Akhirnya, populasi asli diperbarui dengan populasi baru yang telah mengalami seleksi, crossover, dan mutasi, menandai akhir satu generasi evolusi. Setelah semua generasi selesai atau kondisi berhenti tercapai, program menampilkan `JalurTerbaik` yang merupakan solusi optimal yang ditemukan dan menyimpannya ke file `JalurTerbaik.mat`.