**ALGORITMA GENETIK**

**KASUS SOLUSI PERSAMAAN**

Tentukan solusi untuk masalah optimasi persamaan **a+4b+2c+3d = 30**. Cari nilai a, b, c, dan d yang memenuhi persamaan diatas. Silahkan menggunakan bahasa pemrograman favorit kalian dalam menyelesaikannya. Buatlah akun Github dan buat repository dengan nama “NamaLengkap-NIM”. Kumpulkan solusi dalam bentuk file source code (\*.py, \*.c, atau \*.cpp atau ekstensi lainnya sesuai bahasa pemrograman yang Anda pilih) dan laporan tugas (\*.doc/\*.docx) pada folder “Kecerdasan Buatan 2023” dalam repository masing-masing.

Petunjuk penyelesaian:

1. **Pembentukan chromosome**

Karena yang dicari adalah nilai a, b, c, d maka variabel  a, b, c, d dijadikan sebagai gen-gen pembentuk chromosome. Batasan nilai variabel a adalah bilangan integer 0 sampai 30. Sedangkan batasan nilai variabel b, c, dan d adalah bilangan integer 0 sampai 10.

1. **Inisialisasi**

Proses inisialisasi dilakukan dengan cara memberikan nilai awal gen-gen dengan nilai acak sesuai batasan yang telah ditentukan. Misalkan tentukan jumlah populasi adalah 6.

1. **Evaluasi Chromosome**

Permasalahan yang ingin diselesaikan adalah  nilai variabel a, b, c, dan d yang memenuhi persamaan a+4b+2c+3d = 30, maka fungsi\_objektif yang dapat digunakan untuk mendapatkan solusi adalah fungsi\_objektif (chromosome) = | (a+4b+2c+3d) – 30 |

1. **Seleksi Chromosome**

Proses seleksi dilakukan dengan cara membuat chromosome yang mempunyai fungsi\_objektif kecil mempunyai kemungkinan terpilih yang besar atau mempunyai nilai probabilitas yang tinggi.

Cara seleksi adalah sebagai berikut:

1. Untuk itu dapat digunakan fungsi fitness = (1/(1+fungsi\_objektif)).
2. Hitung probabilitas tiap kromosom yaitu: P[i] = fitness[i]/total\_fitness
3. Hitung nilai kumulatif probabilitas:
4. Bangkitkan bilangan acak R antara 0-1, kemudian pilih chromosome ke-k sebagai induk dengan syarat **C[k-1] < R < C[k].**
5. Kromosom baru terbentuk
6. **Crossover**

Metode yang digunakan salah satunya adalah one-cut point, yaitu memilih secara acak satu posisi dalam chromosome induk kemudian saling menukar gen. Chromosome yang dijadikan induk dipilih secara acak dan jumlah chromosome yang mengalami crossover dipengaruhi oleh parameter crossover\_rate (**ρc**). Pseudo-code untuk proses crossover adalah sebagai berikut:

1. **Mutasi**

Jumlah chromosome yang mengalami mutasi dalam satu populasi ditentukan oleh parameter mutation\_rate. Proses mutasi dilakukan dengan cara mengganti satu gen yang terpilih secara acak dengan suatu nilai baru yang didapat secara acak.

1. Pertama hitung dahulu panjang total gen yang ada dalam satu populasi.
2. Misal ρm tentukan 10% maka diharapkan ada 10% dari total\_gen yang mengalami populasi.

Jawaban

1. **Listing Program**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

// Fungsi objektif untuk mencari nilai fitness

int fungsi\_objektif(int a, int b, int c, int d) {

return abs((a + 4\*b + 2\*c + 3\*d) - 30);

}

// Inisialisasi populasi

std::vector<std::vector<int>>

inisialisasi\_populasi(int jumlah\_populasi) {

std::vector<std::vector<int>> populasi;

srand(time(0)); // Seed the random number generator

for (int i = 0; i < jumlah\_populasi; i++) {

int a = rand() % 31; // Random value between 0 and 30 for variable a

int b = rand() % 11; // Random value between 0 and 10 for variable b

int c = rand() % 11; // Random value between 0 and 10 for variable c

int d = rand() % 11; // Random value between 0 and 10 for variable d

std::vector<int> chromosome = {a, b, c, d};

populasi.push\_back(chromosome);

}

return populasi;

}

// Evaluasi populasi

std::vector<int> evaluasi\_populasi(std::vector<std::vector<int>>& populasi) {

std::vector<int> fitness\_values;

for (const auto& chromosome : populasi) {

int fitness = 1 / (1 + fungsi\_objektif(chromosome[0], chromosome[1], chromosome[2], chromosome[3]));

fitness\_values.push\_back(fitness);

}

return fitness\_values;

}

// Seleksi menggunakan metode roulette wheel

std::vector<std::vector<int>> seleksi(std::vector<std::vector<int>>& populasi, const std::vector<int>& fitness\_values) {

int total\_fitness = 0;

for (int fitness : fitness\_values) {

total\_fitness += fitness;

}

std::vector<double> probabilities;

for (int fitness : fitness\_values) {

double probability = (double)fitness / total\_fitness;

probabilities.push\_back(probability);

}

std::vector<double> cumulative\_probabilities;

double cumulative\_probability = 0;

for (double probability : probabilities) {

cumulative\_probability += probability;

cumulative\_probabilities.push\_back(cumulative\_probability);

}

std::vector<std::vector<int>> selected;

srand(time(0)); // Seed the random number generator

for (int i = 0; i < populasi.size(); i++) {

double r = (double)rand() / RAND\_MAX; // Random value between 0 and 1

for (int j = 0; j < cumulative\_probabilities.size(); j++) {

if ((j == 0 && r < cumulative\_probabilities[j]) || (cumulative\_probabilities[j - 1] < r && r < cumulative\_probabilities[j])) {

selected.push\_back(populasi[j]);

break;

}

}

}

return selected;

}

// Crossover menggunakan one-cut point

std::vector<std::vector<int>> crossover(const std::vector<std::vector<int>>& parents, double crossover\_rate) {

std::vector<std::vector<int>> offspring;

srand(time(0)); // Seed the random number generator

for (int i = 0; i < parents.size(); i += 2) {

if ((double)rand() / RAND\_MAX < crossover\_rate) {

int cut\_point = rand() % parents[i].size(); // Random cut point

std::vector<int> child1, child2;

for (int j = 0; j < cut\_point; j++) {

child1.push\_back(parents[i][j]);

child2.push\_back(parents[i + 1][j]);

}

for (int j = cut\_point; j < parents[i].size(); j++) {

child1.push\_back(parents[i + 1][j]);

child2.push\_back(parents[i][j]);

}

offspring.push\_back(child1);

offspring.push\_back(child2);

} else {

offspring.push\_back(parents[i]);

offspring.push\_back(parents[i + 1]);

}

}

return offspring;

}

// Mutasi dengan mengganti satu gen secara acak

void mutasi(std::vector<std::vector<int>>& populasi, double mutation\_rate) {

srand(time(0)); // Seed the random number generator

for (auto& chromosome : populasi) {

if ((double)rand() / RAND\_MAX < mutation\_rate) {

int index = rand() % chromosome.size(); // Random index

chromosome[index] = rand() % (index == 0 ? 31 : 11); // Random value between 0 and 30 for variable a, or between 0 and 10 for variables b, c, d

}

}

}

// Algoritma genetik

std::vector<std::vector<int>> algoritma\_genetik(int jumlah\_generasi, int jumlah\_populasi, double crossover\_rate, double mutation\_rate) {

std::vector<std::vector<int>> populasi = inisialisasi\_populasi(jumlah\_populasi);

for (int generasi = 0; generasi < jumlah\_generasi; generasi++) {

std::vector<int> fitness\_values = evaluasi\_populasi(populasi);

std::vector<std::vector<int>> selected = seleksi(populasi, fitness\_values);

std::vector<std::vector<int>> offspring = crossover(selected, crossover\_rate);

mutasi(offspring, mutation\_rate);

populasi = offspring;

}

return populasi;

}

int main() {

int jumlah\_generasi = 100;

int jumlah\_populasi = 6;

double crossover\_rate = 0.8;

double mutation\_rate = 0.1;

std::vector<std::vector<int> hasil = algoritma\_genetik(jumlah\_generasi, jumlah\_populasi, crossover\_rate, mutation\_rate);

// Menampilkan hasil

for (const auto & chromosome : hasil) {

int a = chromosome[0];

int b = chromosome[1];

int c = chromosome[2];

int d = chromosome[3];

if (a + 4\*b + 2\*c + 3\*d == 30) {

std::cout << "Solusi ditemukan: a = " << a << ", b = " << b << ", c = " << c << ", d = " << d << std::endl;

}

}

return 0;

}

1. **Penjelasan**

Program di atas adalah implementasi algoritma genetika untuk mencari solusi dari sebuah fungsi objektif. Berikut adalah penjelasan singkat mengenai setiap bagian kode:

1. Fungsi objektif (fungsi\_objektif): Fungsi ini menghitung nilai fitness berdasarkan perbedaan absolut antara hasil fungsi (a + 4\*b + 2\*c + 3\*d) dan nilai target, yaitu 30.
2. Inisialisasi populasi (inisialisasi\_populasi): Fungsi ini menghasilkan populasi awal dengan jumlah individu sebanyak jumlah\_populasi. Setiap individu terdiri dari empat variabel a, b, c, dan d yang diberi nilai acak sesuai dengan batas yang ditentukan.
3. Evaluasi populasi (evaluasi\_populasi): Fungsi ini menghitung nilai fitness untuk setiap individu dalam populasi menggunakan fungsi objektif yang telah didefinisikan sebelumnya.
4. Seleksi (seleksi): Metode seleksi yang digunakan dalam kode ini adalah metode roulette wheel. Fungsi ini menghitung probabilitas munculnya setiap individu dalam populasi berdasarkan nilai fitness-nya. Kemudian, individu dipilih secara acak dengan mempertimbangkan probabilitasnya.
5. Crossover (crossover): Operasi crossover dilakukan dengan menggunakan metode one-cut point. Dalam kode ini, crossover dilakukan antara sepasang individu secara acak dengan probabilitas yang ditentukan (crossover\_rate). Anak-anak baru dihasilkan dengan membagi gen dari kedua orang tua di atas titik pemotongan.
6. Mutasi (mutasi): Operasi mutasi dilakukan pada setiap individu dalam populasi dengan probabilitas tertentu (mutation\_rate). Mutasi ini mengganti nilai acak pada satu gen individu.
7. Algoritma genetika (algoritma\_genetik): Fungsi ini mengimplementasikan algoritma genetika dengan melakukan iterasi sebanyak jumlah\_generasi. Pada setiap iterasi, dilakukan evaluasi populasi, seleksi, crossover, dan mutasi. Populasi hasil dari setiap iterasi digunakan sebagai populasi untuk iterasi berikutnya.
8. Main function (main): Pada fungsi main, parameter-parameter seperti jumlah\_generasi, jumlah\_populasi, crossover\_rate, dan mutation\_rate ditentukan. Kemudian, fungsi algoritma\_genetik dipanggil untuk mendapatkan hasil akhir. Hasil tersebut kemudian ditampilkan, dan solusi yang memenuhi kriteria (a + 4*b + 2*c + 3\*d = 30) akan dicetak.