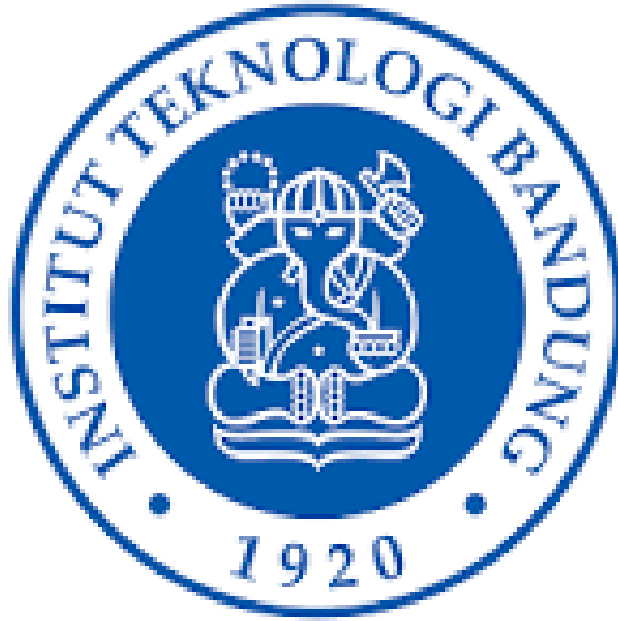


LAPORAN TUGAS BESAR
IF3170 Inteligensi Buatan
Pencarian Solusi Diagonal Magic Cube dengan Local Search



Disusun Oleh:

Kelompok 8

13521117 - Maggie Zeta RS

13522020 - Aurelius Justin PF

13522040 - Dhidit Abdi Aziz

13522090 - Fedrianz Dharma

13522101 - Abdullah Mubarak

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023/2024

Daftar Isi

Daftar Isi	2
BAB 1	
Deskripsi Persoalan	3
BAB 2	
Pembahasan	5
2.1 Pemilihan Fungsi Objektif	5
2.2 Implementasi Algoritma	5
2.3 Hasil Eksperimen dan Analisis	5
BAB 3	
Kesimpulan dan Saran	6
3.1 Kesimpulan	6
3.2 Saran	6
Pembagian Tugas	7
Referensi	8

BAB 1

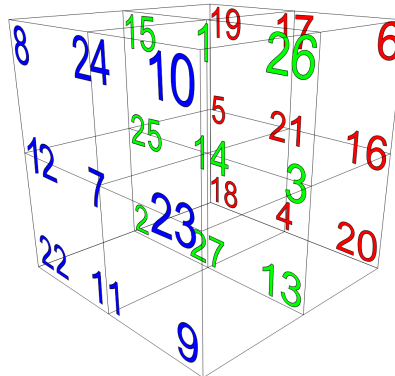
Deskripsi Persoalan

Tugas Besar I pada kuliah IF3170 Inteligensi Buatan bertujuan agar peserta kuliah mendapatkan wawasan tentang bagaimana cara mengimplementasikan algoritma local search untuk mencari solusi suatu permasalahan. Pada tugas ini, peserta kuliah akan ditugaskan untuk mengimplementasikan algoritma local search untuk mencari solusi Diagonal Magic Cube.

Diagonal magic cube merupakan kubus yang tersusun dari angka 1 hingga n^3 tanpa pengulangan dengan n adalah panjang sisi pada kubus tersebut. Angka-angka pada tersusun sedemikian rupa sehingga properti-properti berikut terpenuhi:

- Terdapat satu angka yang merupakan magic number dari kubus tersebut (Magic number tidak harus termasuk dalam rentang 1 hingga n^3 , magic number juga bukan termasuk ke dalam angka yang harus dimasukkan ke dalam kubus)
- Jumlah angka-angka untuk setiap baris sama dengan magic number
- Jumlah angka-angka untuk setiap kolom sama dengan magic number
- Jumlah angka-angka untuk setiap tiang sama dengan magic number
- Jumlah angka-angka untuk seluruh diagonal ruang pada kubus sama dengan magic number
- Jumlah angka-angka untuk seluruh diagonal pada suatu potongan bidang dari kubus sama dengan magic number

Berikut ilustrasi dari potongan bidang yang ada pada suatu kubus berukuran 3:



Terdapat 9 potongan bidang, yaitu:

8	24	10				15	1	26				19	17	6
12	7	23				25	14	3				5	21	16
22	11	9				2	27	13				18	4	20
19	17	6				5	21	16				18	4	20
15	1	26				25	14	3				2	27	13
8	24	10				12	7	23				22	11	9
8	15	19				12	25	5				22	2	18
24	1	17				7	14	21				11	27	4
10	26	6				23	3	16				9	13	20

Diagonal yang dimaksud adalah yang dilingkari warna merah saja

Pada tugas ini, kami kuliah akan menyelesaikan permasalahan Diagonal Magic Cube berukuran $5 \times 5 \times 5$. Initial state dari suatu kubus adalah susunan angka 1 hingga 5^3 secara acak. Kemudian, tiap iterasi pada algoritma local search, langkah yang boleh dilakukan adalah menukar posisi dari 2 angka pada kubus tersebut (2 angka yang ditukar tidak harus bersebelahan). Khusus untuk genetic algorithm, boleh dilakukan penukaran posisi lebih dari 2 angka sekaligus dalam satu iterasi (tetap hanya menukar posisi 2 angka saja juga diperbolehkan).

BAB 2

Pembahasan

2.1 Pemilihan Fungsi Objektif

Diagonal *magic cube* berukuran 5x5x5 memiliki *magic number* 315. Jumlah dari 5 angka pada setiap 25 baris, 25 kolom, 25 tiang, 30 diagonal, dan 4 diagonal ruang harus sama dengan *magic number*, yaitu 315. Dengan begitu, diagonal *magic cube* memiliki *state value (h)* paling optimal sebesar 109. Tiap baris, kolom, tiang, diagonal, atau diagonal ruang yang jumlah 5 angkanya sama dengan *magic number* akan menambahkan *state value (h)* sebesar 1. Oleh karena itu, *objective function* yang digunakan pada *local search* adalah

$$M(n) = \frac{n(n^3+1)}{2}, \text{ nilai magic number}$$

Kami menggunakan satu Objective Function.

1. Objective Function

$f(s)$ = jumlah tiap baris, kolom, tiang, diagonal, atau diagonal ruang yang jumlah 5 angkanya sama dengan magic number $M(5) = 315$. Nilai maksimum dari $f(s)$ adalah 109.

Fungsi ini dipilih karena **kesederhanaan dan kejelasannya**. Dengan menghitung jumlah baris, kolom, tiang, atau diagonal yang memiliki jumlah tepat 315, fungsi ini memberi gambaran langsung tentang seberapa baik *state* magic cube. Jika semakin banyak fitur yang jumlahnya mencapai magic number, maka konfigurasi magic cube semakin mendekati solusi optimal. Fungsi ini melakukan pendekatan evaluasi yang lebih **diskrit**, di mana fokus utamanya adalah mencapai atau tidak mencapai magic number.

2.2 Implementasi Algoritma

1. Kelas Util

Fungsi:

- `randomize_initial_state`: Menginisialisasi state awal secara acak dengan vektor 1 dimensi

```
static vector<int> randomize_initial_state(int n
= 5, int random_state = 42) {
    std::mt19937 rng(random_state);
    vector<int> state(n * n * n);
    iota(state.begin(), state.end(), 1);
```

```

        std::shuffle(state.begin(), state.end(),
rng);
        return state;
    }

```

- **objective_function:** Menghitung nilai dari suatu state

```

static int Objective_Function(vector<int>&
state, int n = 5) {
    int magic_number = (n * (n * n * n + 1))
/ 2;
    int fulfilled_properties = 0;

    auto get = [&](int z, int y, int x) {
        return state[z * n * n + y * n + x];
    };

    // Check Columns
    for (int z = 0; z < n; ++z) {
        for (int x = 0; x < n; ++x) {
            int col_sum = 0;
            for (int y = 0; y < n; ++y) {
                col_sum += get(z, y, x);
            }
            if (col_sum == magic_number)
++fulfilled_properties;
        }
    }

    // Check Rows
    for (int z = 0; z < n; ++z) {
        for (int y = 0; y < n; ++y) {
            int row_sum = 0;
            for (int x = 0; x < n; ++x) {
                row_sum += get(z, y, x);
            }
            if (row_sum == magic_number)
++fulfilled_properties;
        }
    }

    // Check Layers
    for (int x = 0; x < n; ++x) {
        for (int y = 0; y < n; ++y) {
            int layer_sum = 0;

```

```

        for (int z = 0; z < n; ++z) {
            layer_sum += get(z, y, x);
        }
        if (layer_sum == magic_number)
++fulfilled_properties;
    }

    // Check Space Diagonals
    int space_diagonals[4] = {0};
    for (int i = 0; i < n; ++i) {
        space_diagonals[0] += get(i, i, i);
        space_diagonals[1] += get(i, i, n -
i - 1);
        space_diagonals[2] += get(i, n - i -
1, i);
        space_diagonals[3] += get(i, n - i -
1, n - i - 1);
    }
    for (int i = 0; i < 4; ++i) {
        if (space_diagonals[i] ==
magic_number) ++fulfilled_properties;
    }

    for (int i = 0; i < n; ++i) {
        // XY plane
        int xy_diag1 = 0, xy_diag2 = 0;
        for (int j = 0; j < n; ++j) {
            xy_diag1 += get(j, j, i);
            xy_diag2 += get(j, n - j - 1,
i);
        }
        if (xy_diag1 == magic_number)
++fulfilled_properties;
        if (xy_diag2 == magic_number)
++fulfilled_properties;

        // XZ plane
        int xz_diag1 = 0, xz_diag2 = 0;
        for (int j = 0; j < n; ++j) {
            xz_diag1 += get(j, i, j);
            xz_diag2 += get(n - j - 1, i,
j);
        }
        if (xz_diag1 == magic_number)
++fulfilled_properties;
    }

```

```

        if (xz_diag2 == magic_number)
++fulfilled_properties;

        // YZ plane
        int yz_diag1 = 0, yz_diag2 = 0;
        for (int j = 0; j < n; ++j) {
            yz_diag1 += get(i, j, j);
            yz_diag2 += get(i, n - j - 1,
j);
        }

        if (yz_diag1 == magic_number)
++fulfilled_properties;
        if (yz_diag2 == magic_number)
++fulfilled_properties;
    }

    return fulfilled_properties;
}

```

2. Kelas Problem

Atribut:

- current_state: state saat ini
- coordinates: menyimpan koordinat-koordinat yang mungkin pada kubus
- all_pairs: menyimpan permutasi dari koordinat-koordinat
- n: ukuran dari kubus

Fungsi:

- Constructor: Menginisialisasi objek problem

```

problem(int n = 5, int random_seed = 0) : n(n) {
    // Initialize with random state
    this->current_state =
Util::randomize_initial_state(n, random_seed);

    // Generate all possible coordinates in
the cube
    for (int x = 0; x < n; ++x) {
        for (int y = 0; y < n; ++y) {
            for (int z = 0; z < n; ++z) {

this->coordinates.emplace_back(x, y, z);
            }
        }
    }

    // Generate pairs of coordinates

```



```

        for (size_t i = 0; i <
this->coordinates.size(); ++i) {
            for (size_t j = i + 1; j <
this->coordinates.size(); ++j) {

this->all_pairs.emplace_back(this->coordinates[i
], this->coordinates[j]);
            }
        }
    }
}

```

- **index:** Mendapatkan indeks

```

int index(int z, int y, int x) const {
    return z * n * n + y * n + x;
}

```

- **action:** Menukar posisi angka pada koordinat 1 dengan koordinat 2

```

vector<int> action(tuple<int, int, int> coor1,
tuple<int, int, int> coor2) {
    vector<int> new_state =
this->current_state;

    int temp =
current_state[index(get<0>(coor1),
get<1>(coor1), get<2>(coor1))];
    new_state[index(get<0>(coor1),
get<1>(coor1), get<2>(coor1))] =
current_state[index(get<0>(coor2),
get<1>(coor2), get<2>(coor2))];
    new_state[index(get<0>(coor2),
get<1>(coor2), get<2>(coor2))] = temp;

    return new_state;
}

```

- **get_neighbor:** Mendapatkan *neighbor state* dari *current state*

```

Node get_neighbor() {
    const int num_threads =
thread::hardware_concurrency();
    int max_val = 0;
    vector<int> best_neighbor;
}

```

```

        auto chunk_size = all_pairs.size() /
num_threads;
        vector<future<pair<int, vector<int>>>>
futures;

        for (int i = 0; i < num_threads; ++i) {
            auto start = all_pairs.begin() + i *
chunk_size;
            auto end = (i == num_threads - 1) ?
all_pairs.end() : start + chunk_size;

            // Launch asynchronous task for each
chunk

futures.emplace_back(async(launch::async, [this,
start, end]() {
                int local_max_val = 0;
                vector<int> local_best_neighbor;

                for (auto it = start; it != end;
++it) {
                    auto next_state =
action(it->first, it->second);
                    int value =
Util::BObjective_Function(next_state, n);

                    if (value > local_max_val) {
                        local_max_val = value;
                        local_best_neighbor =
next_state;
                    }
                }
                return make_pair(local_max_val,
local_best_neighbor);
            }));
        }

        for (auto& fut : futures) {
            auto result = fut.get();

            if (result.first > max_val) {
                max_val = result.first;
                best_neighbor = result.second;
            }
        }

```

```

current_state = best_neighbor;
return Node(best_neighbor);
}

```

- `get_neighbor_random`: Mendapatkan *neighbor state* secara *random*

```

Node get_neighbor_random() {
    const int current_val =
Util::BObjective_Function(current_state);
    double better = 0;
    double same = 0;
    double worse = 0;
    for (const auto& pair : all_pairs) {
        // Launch asynchronous task for each
pair in all_pairs
        auto next_state = action(pair.first,
pair.second);
        int value =
Util::BObjective_Function(next_state, n);
        if (value > current_val) {
            better++;
        } else if (value == current_val) {
            same++;
        } else {
            worse++;
        }
    }
    cout << "worse: " << worse/all_pairs.size()
<< ", same: " << same/all_pairs.size() << ",
better: " << better/all_pairs.size() << endl;

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> distrib(0, n -
1);

    tuple<int, int, int> point1, point2;

    do {
        point1 = make_tuple(distrib(gen),
distrib(gen), distrib(gen));
        point2 = make_tuple(distrib(gen),
distrib(gen), distrib(gen));
    } while (point1 == point2);

    auto next_state = action(point1, point2);
}

```

```

        for (int i = 0; i < next_state.size(); i++)
        {
            if (next_state[i] != current_state[i])
            {
                cout << next_state[i] << " ";
                cout << current_state[i] << " ";
                cout << i << endl;
                cout << Util::BObjective_Function(current_state) << " ";
                cout << Util::BObjective_Function(next_state) << endl;
            }
        }

        return Node(next_state);
    }

```

3. Kelas Node

Atribut:

- state: state dari node ini
- value: nilai dari state saat ini
- n: magic number

Fungsi

- Constructor: Menginisialisasi objek Node

```

Node(vector<int> state, int n=5) : state(state),
n(n) {
    this->value =
    Util::BObjective_Function(state, n);
}

```

4. LocalSearch

Fungsi

- SteepestHillClimb: Menerapkan algoritma steepest ascent hill-climbing untuk mencari solusi.

```

Node steepestHillClimb(problem p) {
    Node current = Node(p.current_state);
    while (true) {
        Node neighbor = p.get_neighbor();
        if (neighbor.value <= current.value) {
            cout << current.value << endl;
            return current;
        }
    }
}

```

```

    }
    current = neighbor;
  }
}

```

- SidewayHillClimb: Menerapkan algoritma hill-climbing with sideways move.

```

Node SidewaysHillClimb(problem p) {
    Node current = Node(p.current_state);
    int i = 0;
    while (true) {
        i++;
        Node neighbor = p.get_neighbor();
        if (neighbor.value < current.value) {
            return current;
        }
        current = neighbor;
    }
}

```

- RandomRestartHillClimb: Menerapkan algoritma random restart hill-climbing.

```

vector<int> RandomRestartHillClimbing() {
    mutex output_mutex;
    const int num_threads =
thread::hardware_concurrency();
    int best_value = 0;
    vector<int> best_state;

    for (int batch_start = 0; best_value < 60;
batch_start += num_threads) {
        vector<future<Node>> futures;

        for (int i = 0; i < num_threads; ++i) {
            int random_seed = batch_start + i;
            problem p = problem(5, random_seed);

            futures.emplace_back(async(launch::async,
steepestHillClimb, p));
        }

        for (auto& fut : futures) {
            auto result = fut.get();
            int value = result.value;

```

```

        if (value > best_value) {
            lock_guard<mutex>
lock(output_mutex);
            best_value = value;
            best_state = result.state;
        }
    }

    lock_guard<mutex> lock(output_mutex);
    cout << "Best value after " << batch_start
+ num_threads
        << " random restarts: " << best_value
<< endl;
    }

    cout << "Final best value: " << best_value
<< endl;
    return best_state;
}

```

- StochasticHillClimb: Menerapkan algoritma stochastic hill-climbing

```

vector<int> StochasticHillClimb(problem p, int
iter){
    Node current = Node(p.current_state);
    for (int i = 0; i < iter; ++i) {
        Node neighbor = p.get_neighbor_random();
        if (neighbor.value > current.value) {
            current = neighbor;
        }
    }
    return current.state;
}

```

5. Kelas Scheduler

Atribut:

- tipe: tipe dari scheduler
- T0: suhu awal
- alpha: hyperparameter untuk tipe eksponensial
- beta: hyperparameter untuk tipe logaritmik
- k: hyperparameter untuk tipe linear

Fungsi:

- Constructor: Membuat objek dari kelas Scheduler

```
Scheduler(string tipe = "linear", double T0 =
100, double alpha = 0.99, double beta = 0.1,
double k = 1){
    this->tipe = tipe;
    this->T0 = T0;
    this->alpha = alpha;
    this->beta = beta;
    this->k = k;
}
```

- Calculate: Menghitung suhu berdasarkan waktu yang telah berjalan

```
double calculate(int time){
    if (tipe == "linear"){
        return max(T0 - k * time, 0.0);
    } else if (tipe == "exp"){
        return T0 * pow(alpha, time);
    } else { // log
        return T0 / (1 + beta * log(time
+ 1));
    }
}
```

6. Simulated Annealing

Fungsi:

- choose_next: mengecek apakah berpindah ke state berikutnya atau tidak berdasarkan perhitungan suhu, delta, dan threshold

```
bool choose_next(double delta, double T, bool
static2 = true, double thresh = 0.5){
    double proba = exp(delta/T);
    random_device rd;
    mt19937 rng(rd());
    uniform_real_distribution<float>
distFloat(0.0f, 1.0f);
    if (!static2){
        thresh = distFloat(rng);
    }
    cout << (proba > thresh) << endl;
    return proba > thresh;
}
```

- simulatedAnnealing: Menerapkan algoritma simulated annealing

```

vector<int>      simulatedAnnealing(problem      p,
Scheduler scheduler, double thresh = 0.5){
    int cont = 0;
    Node current = Node(p.current_state);
    int time = 1;
    while (true){
        double T = scheduler.calculate(time);
        if (T == 0){
            return current.state;
        }
        Node next = p.get_neighbor_random();
        int delta = next.value - current.value;
        if (delta == 0) {
            cont++;
        } else {
            cont = 0;
        }
        if (delta > 0){
            current = next;
        } else {
            if      (choose_next(delta,      T,      true,
thresh)){
                current = next;
            }
        }
        // cout << "t: " << time << endl;
        // cout << "value: " << current.value <<
endl;
        if (time % 10000 == 0) {
            cout << "t: " << time << endl;
            cout << "value: " << current.value <<
endl;
        }
        time++;
    }
}

```

7. Kelas GeneticAlgo

Atribut:

- pop_size
- n
- generation
- Mutation_rate
- crossover_type

Fungsi:

- Constructor: menginisialisasi objek GenetikAlgo

```
GeneticAlgo(int pop_size = 100, int n = 5, int
generations = 1000, double mutation_rate = 0.1,
string crossover_type = "ox")
    :          pop_size(pop_size),          n(n),
      generations(generations),
      mutation_rate(mutation_rate),
      crossover_type(crossover_type) {}
```

- Init_individual

```
vector<int> init_individual() {
    vector<int> individual(n * n * n);
    iota(individual.begin(), individual.end(),
1);
    shuffle(individual.begin(),
            individual.end(), random_device());
    return individual;
}
```

- Mutate

```
vector<int> mutate(vector<int> individual) {
    int idx1 = rand() % individual.size();
    int idx2 = rand() % individual.size();
    swap(individual[idx1], individual[idx2]);
    return individual;
}
```

- Order_crossover

```
pair<vector<int>, vector<int>>
order_crossover(const vector<int>& parent1,
const vector<int>& parent2) {
    int total_elements = n * n * n;
    vector<int> child1(total_elements, 0),
child2(total_elements, 0);

    int point1 = rand() % total_elements,
point2 = rand() % total_elements;
    if (point1 > point2) swap(point1, point2);

    copy(parent1.begin() + point1,
parent1.begin() + point2, child1.begin() +
point1);
```

```

        copy(parent2.begin() + point1,
parent2.begin() + point2, child2.begin() +
point1);

    for (int i = 0; i < total_elements; ++i) {
        if (child1[i] == 0) {
            for (int val : parent2) {
                if (find(child1.begin(),
child1.end(), val) == child1.end()) {
                    child1[i] = val;
                    break;
                }
            }
        }
        if (child2[i] == 0) {
            for (int val : parent1) {
                if (find(child2.begin(),
child2.end(), val) == child2.end()) {
                    child2[i] = val;
                    break;
                }
            }
        }
    }

    return {child1, child2};
}

```

- Tournament_selection

```

vector<int> tournament_selection(const
vector<vector<int>>& population, const
vector<double>& fitnesses, int k = 3) {
    auto max_fitness_it =
max_element(fitnesses.begin(), fitnesses.end());
    int max_index = distance(fitnesses.begin(),
max_fitness_it);
    return population[max_index];
}

```

- Genetic_algo

```

vector<int> genetic_algo() {
    vector<vector<int>> population;
    for (int i = 0; i < pop_size; ++i) {

```

```

population.push_back(init_individual());
    }

    vector<double> fitnesses;
    for (int generation = 0; generation <
generations; ++generation) {
        // Parallel fitness computation
        vector<future<double>>
fitness_futures;
        for (auto& individual : population) {

fitness_futures.push_back(async(launch::async,
[&individual,      this]()      {      return
Util::Objective_Function(individual, n); }));
        }

        fitnesses.clear();
        for (auto& f : fitness_futures) {
            fitnesses.push_back(f.get());
        }

        if      (*max_element(fitnesses.begin(),
fitnesses.end()) == 105) {
            cout << "Solution found at
generation " << generation << endl;
            return
population[max_element(fitnesses.begin(),
fitnesses.end()) - fitnesses.begin()];
        }

        vector<vector<int>> new_population;
        for (int i = 0; i < pop_size / 2; ++i)
        {
            // Select parents
            vector<int>      parent1      =
tournament_selection(population, fitnesses);
            vector<int>      parent2      =
tournament_selection(population, fitnesses);

            auto      [child1,      child2]      =
order_crossover(parent1, parent2);

            if ((double)rand() / RAND_MAX <
mutation_rate) child1 = mutate(child1);
            if ((double)rand() / RAND_MAX <
mutation_rate) child2 = mutate(child2);

```

```

        new_population.push_back(child1);
        new_population.push_back(child2);
    }

    population = new_population;

    if (generation % 100 == 0) {
        cout << "Generation " <<
generation << ", Max fitness: " <<
*max_element(fitnesses.begin(), fitnesses.end())
<< endl;
    }
}

    auto best_it =
max_element(fitnesses.begin(), fitnesses.end());
    return population[best_it -
fitnesses.begin()];
}

```

2.3 Hasil Eksperimen dan Analisis

Pertanyaan analisis (nanti dihapus):

- Seberapa dekat tiap-tiap algoritma bisa mendekati global optima dan mengapa hasilnya demikian?
- Bagaimana perbandingan hasil pencarian tiap-tiap algoritma dengan algoritma local search yang lain?
- Bagaimana perbandingan durasi proses pencarian tiap algoritma relatif terhadap algoritma lainnya?
- Seberapa konsisten hasil akhir yang didapatkan dari tiap-tiap eksperimen yang dilakukan?
- Bagaimana pengaruh banyak iterasi dan jumlah populasi terhadap hasil akhir pencarian pada Genetic Algorithm?

2.3.1 Steepest Ascent Hill-Climbing

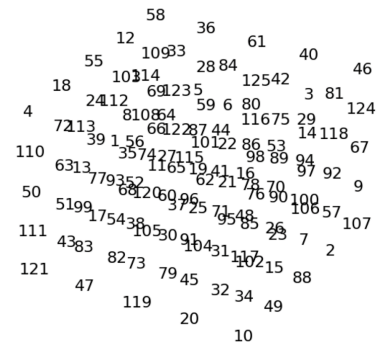
a. Hasil

Nilai objective function akhir: 39, 39, 39

Iterasi: 34, 34, 34

Durasi: 0.28298 detik, 0.395216 detik, 0.396672 detik

3D Magic Cube Visualization



Algorithm: HC

Execution Time: 0.154603

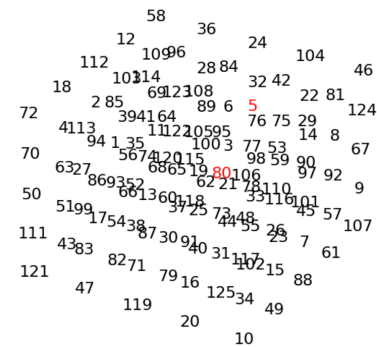
Iteration: 0

State Value: 2

Load File

Show Value Plot

3D Magic Cube Visualization



Algorithm: HC

Execution Time: 0.154603

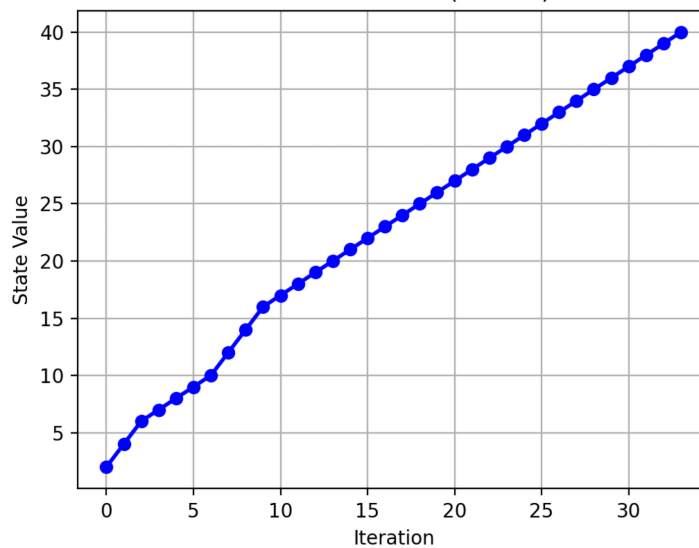
Iteration: 33

State Value: 40

Load File

Show Value Plot

Iteration vs. state (2D Plot)



Initial state	Final State	Screenshot Proses
11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	68 11 69 109 58 62 100 89 28 36 33 98 76 32 24 45 97 14 22 104 107 9 67 124 46 66 56 39 103 12 37 65 122 123 96 44 21 3 6 84 23 116 59 75 42 61 57 92 8 81 17 86 94 2 112 87 13 74 41 114 40 25 19 105 108 102 55 78 77 5 88 7 101 90 29 43 51 63 4 18 82 54 93 1 85 79 30 60 120 64 125 31 73 80 95 49 15 26 110 53 121 111 50 70 72 47 83 99 27 113 119 71 38 52 35 20 16 91 118 115 10 34 117 48 106	<pre> iterasi: 0 value: 2 11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 4 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 3 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 47 83 99 13 113 0 3 0 4 4 4 value: 2 iterasi: 2 2 1 0 2 2 3 value: 4 iterasi: 3 0 0 1 283 4 0 value: 6 iterasi: 4 1 0 0 283 4 0 value: 7 iterasi: 5 0 1 1 2 4 2 value: 8 iterasi: 6 0 1 2 1 3 2 value: 9 iterasi: 7 value: 39 iterasi: 34 0 0 4 0 3 4 Elapsed time: 0.28298 seconds </pre>

	<pre> 68 11 69 109 58 62 100 89 28 36 33 98 76 32 24 45 97 14 22 104 107 9 67 124 46 66 56 39 103 12 37 65 122 123 96 44 21 3 6 84 23 116 59 75 42 61 57 92 8 81 17 86 94 2 112 87 13 74 41 114 40 25 19 105 108 102 55 78 77 5 88 7 101 90 29 43 51 63 4 18 82 54 93 1 85 79 30 60 120 64 125 31 73 80 95 49 15 26 110 53 121 111 50 70 72 47 83 99 27 113 119 71 38 52 35 20 16 91 118 115 10 34 117 48 106 </pre>	<pre> iterasi: 0 value: 2 11 66 69 109 58 62 4 55 105 120 74 10 3 99 13 113 119 73 0 3 0 4 4 4 value: 2 iterasi: 2 2 1 0 2 2 3 value: 4 iterasi: 3 0 0 1 283 4 0 value: 6 iterasi: 4 1 0 0 283 4 0 value: 7 iterasi: 5 0 1 1 2 4 2 value: 8 value: 37 iterasi: 32 3 4 3 4 0 3 value: 38 iterasi: 33 2 3 4 3 3 3 value: 39 iterasi: 34 0 0 4 0 3 4 Elapsed time: 0.395216 seconds </pre>
--	--	--

	68 11 69 109 58 62 100 89 28 36 33 98 76 32 24 45 97 14 22 104 107 9 67 124 46 66 56 39 103 12 37 65 122 123 96 44 21 3 6 84 23 116 59 75 42 61 57 92 8 81 17 86 94 2 112 87 13 74 41 114 40 25 19 105 108 102 55 78 77 5 88 7 101 90 29 43 51 63 4 18 82 54 93 1 85 79 30 60 120 64 125 31 73 80 95 49 15 26 110 53 121 111 50 70 72 47 83 99 27 113 119 71 38 52 35 20 16 91 118 115 10 34 117 48 106	<pre> iterasi: 0 value: 2 11 66 69 109 58 62 101 59 2 4 55 105 120 74 108 114 104 3 99 13 113 119 73 38 52 56 0 3 0 4 4 4 value: 2 iterasi: 2 2 1 0 2 2 3 value: 4 iterasi: 3 0 0 1 283 4 0 value: 6 iterasi: 4 1 0 0 283 4 0 value: 7 iterasi: 5 iterasi: 33 2 3 4 3 3 3 value: 39 iterasi: 34 0 0 4 0 3 4 Elapsed time: 0.396672 seconds </pre>
--	---	--

b. Analisis

Steepest Ascent Hill-Climb memiliki kendala dalam mencapai global optimum karena tidak memiliki cara untuk keluar dari local optimum. Meskipun begitu, algoritma ini memiliki kecepatan yang paling cepat.

2.3.2 Hill-Climbing with Sideways Move

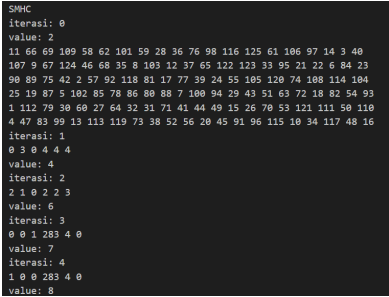
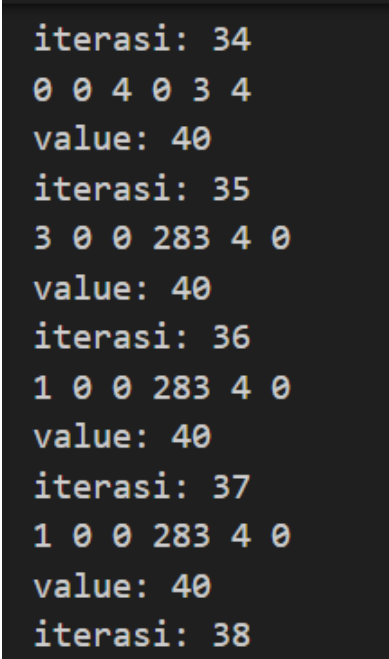
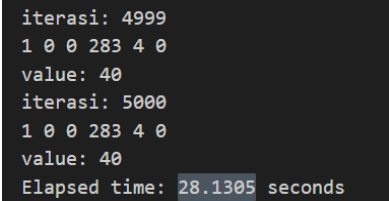
a. Hasil

1. Percobaan 1

Nilai objective function akhir: 40

Max Iterasi: 5000

Durasi: 28.6961, 28.1305, 27.8379 detik

Initial state	Final State	Screenshot Proses
11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	66 11 69 109 104 62 100 89 28 36 33 98 76 32 24 45 97 14 22 58 107 9 67 124 46 43 56 39 103 12 37 65 122 123 96 44 21 3 6 84 23 116 59 75 42 61 57 92 8 81 17 86 94 2 112 87 13 74 41 114 40 25 19 105 108 102 55 78 77 5 88 7 101 90 29 68 51 63 4 18 82 54 93 1 85 79 30 60 120 64 125 31 73 80 95 49 15 26 110 53 121 111 50 70 72 47 83 99 27 113 119 71 38 52 35 20 16 91 118 115 10 34 117 48 106	 <pre> SMC iterasi: 0 value: 2 11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16 iterasi: 1 0 3 0 4 4 4 value: 4 iterasi: 2 2 1 0 2 2 3 value: 6 iterasi: 3 0 0 1 283 4 0 value: 7 iterasi: 4 1 0 0 283 4 0 value: 8 </pre>  <pre> iterasi: 34 0 0 4 0 3 4 value: 40 iterasi: 35 3 0 0 283 4 0 value: 40 iterasi: 36 1 0 0 283 4 0 value: 40 iterasi: 37 1 0 0 283 4 0 value: 40 iterasi: 38 </pre>  <pre> iterasi: 4999 1 0 0 283 4 0 value: 40 iterasi: 5000 1 0 0 283 4 0 value: 40 Elapsed time: 28.1305 seconds </pre>

		<pre> iterasi: 4996 1 0 0 283 4 0 value: 40 iterasi: 4997 1 0 0 283 4 0 value: 40 iterasi: 4998 1 0 0 283 4 0 value: 40 iterasi: 4999 1 0 0 283 4 0 value: 40 iterasi: 5000 1 0 0 283 4 0 value: 40 Elapsed time: 27.8379 seconds </pre>
--	--	--

b. Analisis

Dari beberapa percobaan, didapat bahwa algoritma Hill-Climbing with Sideways Move hanya bisa mencapai score sekitar 40. Hal ini terjadi karena algoritma ini tidak bisa keluar dari local optima. Dibandingkan local search lain, algoritma ini memiliki waktu eksekusi yang cukup cepat, namun bukan yang paling cepat.

Hasil akhir yang didapat pada tiap iterasi tidak terlalu bervariasi, yaitu pada kisaran 27-30 detik.

2.3.3 Random Restart Hill Climbing

a. Hasil

1. Percobaan 1

Nilai objective function akhir: 34

Max Restart: 50

Jumlah iterasi per restart: 30-40an

Durasi: 10.8199, 8.95702, 9.2617 detik

Initial state	Final State	Screenshoot Proses
---------------	-------------	--------------------

11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	39 119 40 47 70 4 94 69 41 107 30 11 64 63 89 108 17 92 55 43 114 74 50 109 6 124 34 110 84 20 122 113 31 75 29 3 95 112 12 93 10 117 102 80 48 56 26 118 37 125 49 57 38 72 99 45 24 35 88 123 86 67 8 121 33 22 46 18 13 1 25 28 58 21 87 120 100 68 105 111 65 19 76 61 90 9 115 52 62 77 2 96 32 82 103 54 106 53 44 42 23 5 59 7 15 79 85 16 60 97 91 27 83 98 116 78 101 71 51 14 66 81 36 104 73	Jumlah iterasi per restart: 34 Jumlah iterasi per restart: 34 Jumlah iterasi per restart: 30 Jumlah iterasi per restart: 30 Jumlah iterasi per restart: 29 Jumlah iterasi per restart: 30 Jumlah iterasi per restart: 32 Jumlah iterasi per restart: 34 Jumlah iterasi per restart: 32 Jumlah iterasi per restart: 32 Jumlah iterasi per restart: 33 Jumlah iterasi per restart: 29 Jumlah iterasi per restart: 34 Jumlah iterasi per restart: 32 Jumlah iterasi per restart: 36 Jumlah iterasi per restart: 27 Jumlah iterasi per restart: 35 Jumlah iterasi per restart: 36 Jumlah iterasi per restart: 38 Jumlah iterasi per restart: 30 Jumlah iterasi per restart: 35 Jumlah iterasi per restart: 36 Jumlah iterasi per restart: 36 Jumlah iterasi per restart: 40 Jumlah iterasi per restart: 35 Jumlah iterasi per restart: 34 Jumlah restart: 50 Execution Time: 10.8199
---	---	---

b. Analisis

Algoritma Random Restart Hill Climbing melakukan pengulangan Hill Climbing setiap kali mengalami *stuck* dengan mengubah initial statenya. Hasil dari algoritma ini cukup baik, namun karena memiliki batas max restart, algoritma ini dapat mengalami penurunan jika initial state pada restart berikutnya lebih buruk dan itu merupakan restart yang terakhir.

2.3.2 Simulated Annealing

a. Hasil

1. Percobaan 1

Nilai objective function akhir: 2

Frekuensi stuck: 870

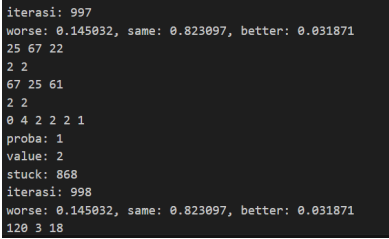
Tipe: linear

T0 = 1000

k = 1

thresh = 0.5

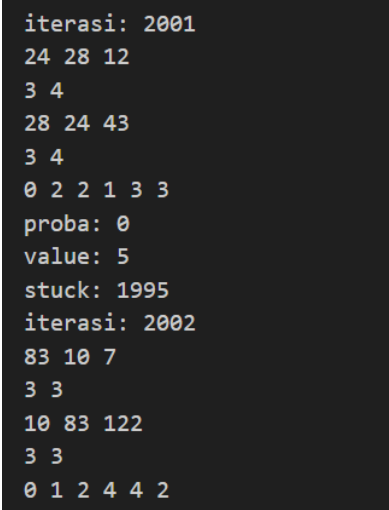
Durasi: 38.2901 detik

Initial state	Final State	Screenshot Proses
11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 120 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 3 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	 <pre> iterasi: 997 worse: 0.145032, same: 0.823097, better: 0.031871 25 67 22 2 2 67 25 61 2 2 0 4 2 2 2 1 proba: 1 value: 2 stuck: 868 iterasi: 998 worse: 0.145032, same: 0.823097, better: 0.031871 120 3 18 </pre>

2. Percobaan 2

Nilai objective function akhir: 5

Frekuensi stuck: 3333**Tipe: exponent****T0 = 100****alpha = 0.8****thresh = 0.5****Durasi: 89.8217 detik**

Initial state	Final State	Screenshoot Proses
72 76 57 22 88 118 101 10 20 103 58 86 28 113 123 35 117 107 12 21 66 106 96 121 29 55 91 9 47 8 31 74 105 109 94 39 70 73 15 64 69 108 53 24 125 11 23 19 52 5 85 17 71 1 116 42 110 119 49 60 59 115 111 38 104 34 37 114 99 36 7 56 75 51 122 41 32 33 44 81 102 4 6 65 124 62 87 13 25 92 90 40 30 48 80 78 84 46 26 89 2 68 100 120 61 45 3 95 97 79 112 18 27 50 54 93 16 67 77 14 63 43 83 82 98	72 76 57 22 88 118 101 10 20 103 58 86 28 113 123 35 117 107 12 44 66 106 96 121 29 55 91 9 47 8 31 74 105 109 94 39 70 73 15 64 69 108 53 24 125 11 23 19 52 5 85 17 71 1 116 42 110 119 49 60 59 115 111 38 104 34 37 114 99 36 7 56 75 51 122 41 32 33 21 81 102 4 6 65 124 62 87 13 25 92 90 40 30 48 80 78 84 46 26 89 2 68 100 120 61 45 3 95 97 79 112 18 27 50 54 93 16 67 77 14 63 43 83 82 98	 <pre> iterasi: 2001 24 28 12 3 4 28 24 43 3 4 0 2 2 1 3 3 proba: 0 value: 5 stuck: 1995 iterasi: 2002 83 10 7 3 3 10 83 122 3 3 0 1 2 4 4 2 </pre>

3. Percobaan 3

Nilai objective function akhir: 2

Frekuensi stuck: 567**Tipe: linear****T0 = 200****k = 0.3****thresh = 0.5****Durasi: 32.8991 detik**

Initial state	Final State	Screenshoot Proses
11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 31 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 29 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	<pre> iterasi: 665 26 109 3 2 2 109 26 97 2 2 0 0 3 3 4 2 proba: 1 value: 2 stuck: 566 iterasi: 666 31 29 74 2 2 29 31 91 2 2 2 4 4 3 3 1 proba: 1 value: 2 stuck: 567 iterasi: 667 Elapsed time: 32.8991 seconds </pre>

b. Analisis

Dari beberapa percobaan, didapat bahwa algoritma Simulated Annealing hanya bisa mencapai score kurang dari 10 walaupun sudah mencapai lebih dari 1 juta iterasi. Hal ini terjadi karena kemungkinan successor yang lebih baik dari state saat ini berkurang seiring state value yang meningkat.

Dibandingkan local search lain, algoritma ini memiliki waktu eksekusi yang cepat karena tidak perlu melakukan pemilihan successor terbaik, hanya random.

Pengaruh schedule terhadap simulated annealing.

Jenis Schedule yang digunakan untuk melakukan *mapping* antara waktu dengan suhu akan mempengaruhi jumlah iterasi dan nilai probabilitas untuk melakukan perpindahan state apabila *successor state* lebih buruk dari *current state*. Semakin lambat suhu turun, maka pergerakan dari state akan lebih bebas untuk naik ataupun turun. Sebaliknya, semakin cepat suhu turun, maka pergerakan state untuk berpindah akan lebih terbatas sesuai dengan nilai probabilitas dan threshold yang ditentukan.

2.3.3 Genetic Algorithm

a. Hasil Eksperimen

1. Percobaan 1

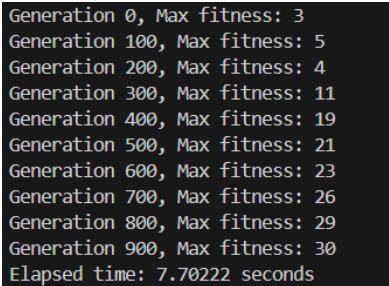
Nilai objective function akhir: 30, 29, 31

Jumlah populasi: 100

Banyak iterasi: 1000

Mutation rate: 0.3

Durasi: 7.70222 detik, 9.18359 detik, 9.91201 detik

Initial state	Final State	Screenshot Proses
11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 120 74 108 114	67 22 87 88 51 20 120 37 61 124 70 79 11 80 123 99 30 95 9 82 59 64 85 84 108 13 115 53 42 92 24 74 107 111 46 91 86 21 73 106 112 16 6 2 48 75 83 29 102 26 122 1 97 17 56 27 63 49 60 36	 <pre> Generation 0, Max fitness: 3 Generation 100, Max fitness: 5 Generation 200, Max fitness: 4 Generation 300, Max fitness: 11 Generation 400, Max fitness: 19 Generation 500, Max fitness: 21 Generation 600, Max fitness: 23 Generation 700, Max fitness: 26 Generation 800, Max fitness: 29 Generation 900, Max fitness: 30 Elapsed time: 7.70222 seconds </pre>

104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	4 34 47 66 104 43 52 54 65 101 119 44 35 15 18 25 8 114 50 40 68 71 103 45 28 62 3 10 58 96 31 100 7 121 117 93 19 77 109 72 57 105 23 118 12 39 41 116 38 81 69 113 55 76 14 94 125 32 78 110 90 33 89 5 98	
	53 81 106 4 71 25 88 23 28 48 45 58 77 52 83 11 65 49 24 33 74 103 26 39 73 15 78 121 9 92 108 123 115 35 79 50 44 37 42 36 85 104 2 105 19 57 17 116 124 1 80 109 68 38 66 120 27 110 51 72 13 98 21 67 75 119 76 60 69 96 101 5 56 90 6 31 63 107 54 114 3 112 55 99 46 122 113 87 61 30 22 20 97 94 82 70 111 84 7 43 40 64 47 59 125 10 12 100 102 18 117 95 34 93 91 62 14 118 32 89 86 41 16 29 8	Generation 0, Max fitness: 3 Generation 100, Max fitness: 7 Generation 200, Max fitness: 18 Generation 300, Max fitness: 24 Generation 400, Max fitness: 26 Generation 500, Max fitness: 26 Generation 600, Max fitness: 28 Generation 700, Max fitness: 28 Generation 800, Max fitness: 29 Generation 900, Max fitness: 29 Elapsed time: 9.18359 seconds

	13 21 87 121 84 51 109 67 44 33 75 38 125 17 40 83 11 106 29 117 93 78 26 104 111 39 99 42 114 59 27 74 79 103 96 10 64 118 47 115 43 76 98 61 22 36 2 54 112 23 123 110 25 14 105 116 88 4 50 20 53 68 60 82 52 6 37 56 92 48 63 12 57 77 119 34 1 15 73 72 35 85 70 69 28 55 8 113 120 19 45 94 9 102 65 71 5 108 124 7 91 16 62 24 101 90 86 95 3 41 122 107 32 49 89 100 97 46 31 18 30 58 80 81 66	<pre> Generation 0, Max fitness: 4 Generation 100, Max fitness: 7 Generation 200, Max fitness: 14 Generation 300, Max fitness: 18 Generation 400, Max fitness: 21 Generation 500, Max fitness: 25 Generation 600, Max fitness: 28 Generation 700, Max fitness: 30 Generation 800, Max fitness: 30 Generation 900, Max fitness: 31 Elapsed time: 9.91201 seconds </pre>
--	---	--

2. Percobaan 2

Nilai objective function akhir: 33, 38, 35**Jumlah populasi:** 100**Banyak iterasi:** 2000**Durasi:** 7.29607 detik

Initial state	Final State	Screenshot Proses
---------------	-------------	-------------------

11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	40 70 36 72 97 102 77 86 33 17 1 93 104 53 64 103 14 81 71 46 67 94 7 3 23 57 121 80 18 39 43 125 85 20 42 51 100 119 21 24 83 10 76 35 111 107 117 13 15 63 114 5 60 109 27 66 68 6 74 49 87 29 45 59 95 32 48 52 56 26 16 2 124 55 118 30 34 19 91 88 22 78 37 115 9 90 75 89 11 50 108 69 105 106 8 65 98 79 61 12 73 116 120 25 122 96 123 101 84 38 28 41 110 54 82 113 31 112 47 44 62 4 92 58 99	Generation 0, Max fitness: 3 Generation 100, Max fitness: 4 Generation 200, Max fitness: 4 Generation 300, Max fitness: 5 Generation 400, Max fitness: 19 Generation 500, Max fitness: 23 Generation 600, Max fitness: 24 Generation 700, Max fitness: 26 Generation 800, Max fitness: 26 Generation 900, Max fitness: 27 Generation 1000, Max fitness: 28 Generation 1100, Max fitness: 28 Generation 1200, Max fitness: 29 Generation 1300, Max fitness: 29 Generation 1400, Max fitness: 30 Generation 1500, Max fitness: 31 Generation 1600, Max fitness: 32 Generation 1700, Max fitness: 33 Generation 1800, Max fitness: 33 Generation 1900, Max fitness: 33
---	---	--

	17 40 48 97 113 34 103 27 26 62 49 14 120 63 69 77 41 81 101 109 19 117 98 28 29 91 51 50 30 93 89 74 15 33 102 44 99 57 1 114 45 80 23 66 13 46 43 118 96 12 22 125 84 5 79 92 55 67 90 11 56 9 106 53 47 75 94 61 21 85 87 100 70 6 111 112 2 116 78 7 122 18 20 58 123 32 24 76 104 54 83 38 86 25 36 3 39 119 59 95 73 121 68 110 52 105 65 4 108 82 115 71 10 88 31 35 42 64 37 72 8 16 60 107 124	<pre> Generation 0, Max fitness: 2 Generation 100, Max fitness: 5 Generation 200, Max fitness: 5 Generation 300, Max fitness: 5 Generation 400, Max fitness: 9 Generation 500, Max fitness: 15 Generation 600, Max fitness: 22 Generation 700, Max fitness: 25 Generation 800, Max fitness: 27 Generation 900, Max fitness: 28 Generation 1000, Max fitness: 30 Generation 1100, Max fitness: 31 Generation 1200, Max fitness: 34 Generation 1300, Max fitness: 35 Generation 1400, Max fitness: 36 Generation 1500, Max fitness: 36 Generation 1600, Max fitness: 37 Generation 1700, Max fitness: 37 Generation 1800, Max fitness: 37 Generation 1900, Max fitness: 38 Elapsed time: 17.5194 seconds </pre>
--	---	---

	124 12 107 8 64 101 99 5 72 38 98 78 22 80 27 45 70 53 59 97 20 77 33 96 89 24 87 26 19 23 35 4 109 106 62 58 31 68 74 84 95 15 21 3 111 103 39 66 123 36 25 30 69 28 51 14 112 88 65 82 63 47 41 50 114 125 16 1 56 117 49 57 94 34 81 75 73 46 115 6 55 92 37 54 52 17 116 85 7 122 32 11 121 60 91 93 100 120 40 44 90 113 67 83 13 110 9 76 18 102 79 43 71 104 29 86 108 119 61 10 105 42 2 48 118	<pre> Generation 0, Max fitness: 3 Generation 100, Max fitness: 4 Generation 200, Max fitness: 4 Generation 300, Max fitness: 4 Generation 400, Max fitness: 17 Generation 500, Max fitness: 22 Generation 600, Max fitness: 27 Generation 700, Max fitness: 29 Generation 800, Max fitness: 32 Generation 900, Max fitness: 32 Generation 1000, Max fitness: 32 Generation 1100, Max fitness: 32 Generation 1200, Max fitness: 34 Generation 1300, Max fitness: 34 Generation 1400, Max fitness: 34 Generation 1500, Max fitness: 35 Generation 1600, Max fitness: 35 Generation 1700, Max fitness: 35 Generation 1800, Max fitness: 35 Generation 1900, Max fitness: 35 Elapsed time: 18.164 seconds </pre>
--	---	---

3. Percobaan 3

Nilai objective function akhir: 39, 42, 41**Jumlah populasi:** 100**Banyak iterasi:** 10000**Durasi:** 78.9606 detik, 106.44 detik, 105.12 detik

Initial state	Final State	Screenshot Proses
---------------	-------------	-------------------

11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	81 30 93 113 32 122 65 50 5 79 4 80 35 107 89 40 119 94 10 76 124 21 43 97 62 78 69 14 125 53 100 123 55 88 8 60 85 26 64 115 114 37 104 31 82 12 1 72 7 57 23 87 44 84 77 47 63 56 96 71 16 11 108 42 9 86 34 38 48 109 3 120 74 45 73 58 39 101 25 92 13 59 15 18 46 118 110 41 24 22 2 102 52 121 95 36 54 20 99 106 75 90 19 103 28 33 6 91 70 111 117 29 112 51 98 68 83 27 105 61 116 49 66 67 17	Generation 0, Max fitness: 4 Generation 100, Max fitness: 9 Generation 200, Max fitness: 19 Generation 300, Max fitness: 25 Generation 400, Max fitness: 27 Generation 500, Max fitness: 28 Generation 600, Max fitness: 28 Generation 700, Max fitness: 29 Generation 800, Max fitness: 31 Generation 900, Max fitness: 32 Generation 1000, Max fitness: 32 Generation 1100, Max fitness: 33 Generation 1200, Max fitness: 34 Generation 1300, Max fitness: 35 Generation 1400, Max fitness: 36 Generation 1500, Max fitness: 36 Generation 1600, Max fitness: 36 Generation 1700, Max fitness: 36 Generation 1800, Max fitness: 37 Generation 1900, Max fitness: 37 Generation 2000, Max fitness: 37 Generation 2100, Max fitness: 37 Generation 2200, Max fitness: 37 Generation 2300, Max fitness: 37 Generation 2400, Max fitness: 37 Generation 2500, Max fitness: 37 Generation 2600, Max fitness: 37 Generation 2700, Max fitness: 37 Generation 2800, Max fitness: 37 Generation 2900, Max fitness: 37 Generation 3000, Max fitness: 37 Generation 3100, Max fitness: 37 Generation 3200, Max fitness: 37 Generation 3300, Max fitness: 37 Generation 3400, Max fitness: 37 Generation 3500, Max fitness: 37 Generation 3600, Max fitness: 37 Generation 3700, Max fitness: 37 Generation 3800, Max fitness: 37 Generation 3900, Max fitness: 37 Generation 4000, Max fitness: 37 Generation 4100, Max fitness: 37 Generation 9800, Max fitness: 39 Generation 9900, Max fitness: 39 Elapsed time: 78.9606 seconds
---	---	---

	<p>48 72 22 59 26 32 73 28 84 98 30 14 77 114 27 100 23 80 9 103 105 60 42 43 65</p> <p>113 34 16 101 51 33 90 75 1 116 37 79 38 67 8 17 2 55 20 36 115 110 7 13 104</p> <p>47 4 39 117 108 122 85 19 89 24 6 111 10 86 102 5 54 94 124 3 82 61 121 53 78</p> <p>66 57 74 50 68 40 46 76 112 41 123 92 120 97 118 106 11 125 71 15 95 52 49 107 12</p> <p>18 96 93 45 63 88 21 25 29 62 119 109 70 69 99 87 58 44 91 35 64 31 83 81 56</p>	<p>Generation 0, Max fitness: 5 Generation 100, Max fitness: 8 Generation 200, Max fitness: 18 Generation 300, Max fitness: 22 Generation 400, Max fitness: 27 Generation 500, Max fitness: 28 Generation 600, Max fitness: 29 Generation 700, Max fitness: 29 Generation 800, Max fitness: 31 Generation 900, Max fitness: 34 Generation 1000, Max fitness: 34 Generation 1100, Max fitness: 35 Generation 1200, Max fitness: 36 Generation 1300, Max fitness: 37 Generation 1400, Max fitness: 37 Generation 1500, Max fitness: 37 Generation 1600, Max fitness: 38 Generation 1700, Max fitness: 38 Generation 1800, Max fitness: 38 Generation 1900, Max fitness: 38 Generation 2000, Max fitness: 38 Generation 2100, Max fitness: 38 Generation 2200, Max fitness: 38 Generation 2300, Max fitness: 38 Generation 2400, Max fitness: 38 Generation 2500, Max fitness: 38 Generation 2600, Max fitness: 38 Generation 2700, Max fitness: 38</p> <p>Generation 9000, Max fitness: 42 Generation 9100, Max fitness: 42 Generation 9200, Max fitness: 42 Generation 9300, Max fitness: 42 Generation 9400, Max fitness: 42 Generation 9500, Max fitness: 42 Generation 9600, Max fitness: 42 Generation 9700, Max fitness: 42 Generation 9800, Max fitness: 42 Generation 9900, Max fitness: 42 Elapsed time: 106.44 seconds</p>
--	--	---

	52 32 50 94 87 99 85 20 29 82 107 19 125 55 9 56 47 118 5 89 113 41 70 92 48 101 18 49 74 73 112 26 62 93 22 51 81 34 111 38 13 39 104 14 115 78 77 66 27 67 86 24 43 60 102 30 96 7 44 15 65 37 121 33 124 122 36 84 69 4 12 21 23 58 100 110 8 98 11 88 57 83 116 35 79 90 119 1 45 91 109 97 6 75 28 72 53 80 64 46 103 3 106 16 95 17 25 117 114 42 2 59 120 71 63 31 105 10 108 61 40 123 76 68 54	<pre> Generation 0, Max fitness: 3 Generation 100, Max fitness: 4 Generation 200, Max fitness: 5 Generation 300, Max fitness: 5 Generation 400, Max fitness: 16 Generation 500, Max fitness: 24 Generation 600, Max fitness: 25 Generation 700, Max fitness: 27 Generation 800, Max fitness: 31 Generation 900, Max fitness: 31 Generation 1000, Max fitness: 32 Generation 1100, Max fitness: 32 Generation 1200, Max fitness: 34 Generation 1300, Max fitness: 34 Generation 1400, Max fitness: 34 Generation 1500, Max fitness: 35 Generation 1600, Max fitness: 35 Generation 1700, Max fitness: 35 Generation 1800, Max fitness: 35 Generation 1900, Max fitness: 35 Generation 2000, Max fitness: 35 Generation 2100, Max fitness: 36 Generation 2200, Max fitness: 36 Generation 2300, Max fitness: 37 Generation 2400, Max fitness: 38 Generation 2500, Max fitness: 38 Generation 2600, Max fitness: 38 Generation 2700, Max fitness: 38 Generation 9600, Max fitness: 41 Generation 9700, Max fitness: 41 Generation 9800, Max fitness: 41 Generation 9900, Max fitness: 41 Elapsed time: 105.12 seconds </pre>
--	---	---

4. Percobaan 4

Nilai objective function akhir: 28, 28, 29**Jumlah populasi:** 50**Banyak iterasi:** 1000**Durasi:** 3.63564 detik, 3.73187 detik, 3.70777 detik

Initial state	Final State	Screenshot Proses
---------------	-------------	-------------------

11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	5 6 119 84 101 70 92 106 110 122 1 93 62 100 40 86 49 25 77 78 113 10 3 97 104 124 32 23 116 20 44 87 55 74 11 54 42 90 121 8 34 72 123 41 45 59 82 50 117 61 38 108 21 115 33 53 91 96 17 79 30 46 94 2 103 9 7 68 4 37 15 14 36 52 16 48 60 71 118 18 29 76 67 12 39 88 75 47 57 99 51 85 58 56 27 83 26 63 107 114 109 111 28 43 24 120 98 95 102 64 81 112 22 35 65 13 31 80 66 125 89 19 105 69 73	Generation 0, Max fitness: 3 Generation 100, Max fitness: 11 Generation 200, Max fitness: 16 Generation 300, Max fitness: 17 Generation 400, Max fitness: 22 Generation 500, Max fitness: 24 Generation 600, Max fitness: 27 Generation 700, Max fitness: 28 Generation 800, Max fitness: 28 Generation 900, Max fitness: 28 Elapsed time: 3.63564 seconds
	29 96 20 86 98 117 101 121 61 123 7 110 17 99 82 67 81 112 47 50 58 105 125 22 62 97 88 27 51 49 74 37 21 80 65 38 77 8 4 36 23 87 44 114 106 83 26 34 113 59 25 92 90 66 42 40 63 33 76 102 64 104 85 122 68	Generation 0, Max fitness: 3 Generation 100, Max fitness: 11 Generation 200, Max fitness: 19 Generation 300, Max fitness: 24 Generation 400, Max fitness: 24 Generation 500, Max fitness: 25 Generation 600, Max fitness: 26 Generation 700, Max fitness: 26 Generation 800, Max fitness: 28 Generation 900, Max fitness: 28 Elapsed time: 3.73187 seconds

	<p>16 54 89 84 72 108 2 11 32 43</p> <p>100 14 45 39 124 48 111 31 13 10 30 116 115 35 19 103 46 5 107 12 56 60 9 119 53</p> <p>94 24 79 70 73 69 91 3 109 18 15 28 71 120 55 41 52 118 1 75 6 78 93 95 57</p>	
	<p>59 28 119 24 69 37 85 20 52 121 111 7 83 113 1 57 66 84 93 15 45 110 9 33 109</p> <p>53 95 61 63 44 97 10 6 102 104 107 11 8 35 39 22 14 103 94 101 124 19 54 21 27</p> <p>77 43 80 65 41 25 92 112 78 49 67 51 48 68 71 108 91 23 42 98 38 55 123 62 56</p> <p>4 58 50 115 88 70 46 73 96 30 122 125 90 31 89 117 32 26 40 5 2 13 76 105 17</p> <p>18 74 16 120 87 86 82 72 29 60 106 47 36 12 114 3 99 79 34 100 64 118 116 81 75</p>	<pre> Generation 0, Max fitness: 3 Generation 100, Max fitness: 10 Generation 200, Max fitness: 13 Generation 300, Max fitness: 15 Generation 400, Max fitness: 17 Generation 500, Max fitness: 21 Generation 600, Max fitness: 25 Generation 700, Max fitness: 27 Generation 800, Max fitness: 29 Generation 900, Max fitness: 29 Elapsed time: 3.70777 seconds </pre>

5. Percobaan 5

Nilai objective function akhir: 6, 7, 9**Jumlah populasi:** 150**Banyak iterasi:** 1000**Durasi:** 13.6513 detik, 11.8476 detik, 13.491 detik

Initial state	Final State	Screenshoot Proses
11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42 2 57 92 118 81 17 77 39 24 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	7 108 80 5 11 113 2 78 98 50 118 39 23 106 84 76 72 54 83 120 111 47 56 109 101 73 25 60 64 119 93 79 97 104 102 35 85 40 121 61 12 96 63 107 92 112 86 115 103 44 117 66 3 124 17 31 82 81 8 45 87 90 100 22 16 34 38 10 91 13 114 41 125 70 71 55 89 15 51 14 27 4 30 18 6 110 36 9 94 24 48 105 19 59 42 37 62 28 67 77 88 43 26 116 57 32 75 123 52 33 53 46 20 21 99 65 122 69 58 1 49 29 95 68 74	<pre> Generation 0, Max fitness: 3 Generation 100, Max fitness: 4 Generation 200, Max fitness: 6 Generation 300, Max fitness: 4 Generation 400, Max fitness: 4 Generation 500, Max fitness: 5 Generation 600, Max fitness: 5 Generation 700, Max fitness: 6 Generation 800, Max fitness: 4 Generation 900, Max fitness: 6 Elapsed time: 13.6513 seconds </pre>

	<p>41 67 74 8 77 76 44 16 101 35 30 104 118 22 4 19 93 99 59 97 37 42 55 17 1</p> <p>111 123 26 91 102 80 47 116 63 9 15 49 95 83 73 62 31 110 88 24 13 45 85 68 20</p> <p>119 92 61 108 106 89 48 115 38 18 86 100 11 40 33 3 82 58 94 109 46 5 60 51 124</p> <p>54 52 125 122 23 87 28 121 6 25 75 65 90 56 96 21 71 69 34 120 12 78 81 113 103</p> <p>7 64 43 98 50 72 114 70 36 39 84 105 57 66 27 107 14 79 32 112 29 10 2 117 53</p>	<pre> Generation 0, Max fitness: 3 Generation 100, Max fitness: 3 Generation 200, Max fitness: 6 Generation 300, Max fitness: 7 Generation 400, Max fitness: 6 Generation 500, Max fitness: 6 Generation 600, Max fitness: 6 Generation 700, Max fitness: 8 Generation 800, Max fitness: 5 Generation 900, Max fitness: 4 Elapsed time: 11.8476 seconds </pre>
	<p>85 66 77 84 4 16 81 34 70 33 107 71 58 47 73 125 74 55 64 112 93 104 23 1 94</p> <p>49 111 32 2 96 14 56 109 41 76 68 5 57 28 18 117 13 99 82 83 35 7 19 27 43</p> <p>106 15 25 75 78 31 30 26 59 101 69 45 118 110 100 46 108 11 79 67</p>	<pre> Generation 0, Max fitness: 3 Generation 100, Max fitness: 5 Generation 200, Max fitness: 5 Generation 300, Max fitness: 6 Generation 400, Max fitness: 7 Generation 500, Max fitness: 6 Generation 600, Max fitness: 6 Generation 700, Max fitness: 9 Generation 800, Max fitness: 5 Generation 900, Max fitness: 4 Elapsed time: 13.491 seconds </pre>

	60 36 105 21 62 17 80 90 42 24 51 119 53 123 72 61 6 44 87 8 29 39 88 102 10 63 38 52 121 115 50 116 20 65 95 114 113 54 9 48 92 22 103 40 86 124 91 12 97 122 120 98 37 89 3	
--	---	--

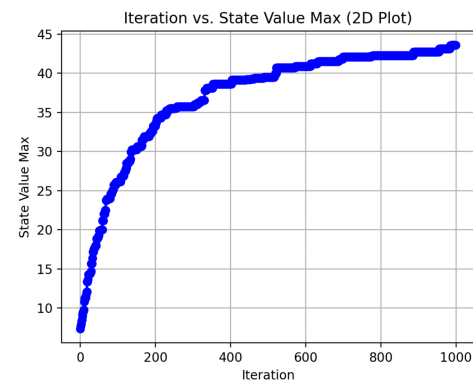
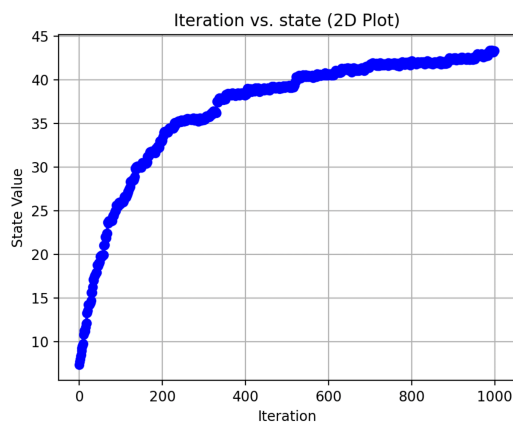
6. Percobaan 6

Nilai objective function akhir: 16, 14, 22

Jumlah populasi: 25

Banyak iterasi: 1000

Durasi: 2.20158 detik, 2.50711 detik, 2.46556 detik



Initial state	Final State	Screenshoot Proses
11 66 69 109 58 62 101 59 28 36 76 98 116 125 61 106 97 14 3 40 107 9 67 124 46 68 35 8 103 12 37 65 122 123 33 95 21 22 6 84 23 90 89 75 42	81 65 91 34 44 103 96 16 50 4 45 86 73 75 51 83 106 111 22 90 114 53 104 107 35 58 121 14 28 94 54 10 76 63 102 48 77 40 92 119 31 38 46 56 55	<pre> Generation 0, Max fitness: 2 Generation 100, Max fitness: 7 Generation 200, Max fitness: 6 Generation 300, Max fitness: 7 Generation 400, Max fitness: 7 Generation 500, Max fitness: 10 Generation 600, Max fitness: 13 Generation 700, Max fitness: 10 Generation 800, Max fitness: 16 Generation 900, Max fitness: 16 Elapsed time: 2.20158 seconds </pre>

2 57 92 118 81 17 77 39 24 55 105 120 74 108 114 104 25 19 87 5 102 85 78 86 80 88 7 100 94 29 43 51 63 72 18 82 54 93 1 112 79 30 60 27 64 32 31 71 41 44 49 15 26 70 53 121 111 50 110 4 47 83 99 13 113 119 73 38 52 56 20 45 91 96 115 10 34 117 48 16	124 60 113 32 82 57 8 112 19 9 116 118 37 69 43 6 79 70 39 85 20 68 29 108 120 99 42 18 64 62 67 21 125 115 49 27 110 95 3 80 78 41 13 17 93 117 72 23 24 1 2 71 59 101 61 52 36 100 74 109 15 5 30 7 47 84 97 123 12 11 98 88 25 105 26 66 89 87 33 122	
	16 66 22 51 96 85 76 15 50 89 33 78 12 64 17 122 54 7 23 2 18 4 46 62 49 114 28 45 107 71 84 3 69 19 63 43 106 41 30 44 40 87 103 109 104 110 34 75 112 117 83 125 94 101 31 65 48 37 72 124 13 95 9 74 77 98 80 108 60 5 79 97 39 100 61 47 90 55 86 1 81 27 52 67 88 11 24 29 42 118 116 92 32 91 105 70 82 8 25 121 58 14 53 6 99 56 113 119 21 120	Generation 0, Max fitness: 3 Generation 100, Max fitness: 10 Generation 200, Max fitness: 12 Generation 300, Max fitness: 16 Generation 400, Max fitness: 11 Generation 500, Max fitness: 12 Generation 600, Max fitness: 14 Generation 700, Max fitness: 16 Generation 800, Max fitness: 16 Generation 900, Max fitness: 14 Elapsed time: 2.50711 seconds

	123 20 36 102 38 68 111 59 35 115 10 57 93 26 73	
	19 42 54 117 83 66 78 121 88 60 26 17 102 28 100 47 49 34 61 87 113 101 4 62 55 93 3 9 32 107 1 25 73 96 120 76 85 123 11 124 56 77 67 44 15 89 65 43 13 105 94 104 59 39 82 68 86 30 74 70 103 40 114 8 31 99 71 48 112 109 12 37 64 51 23 63 36 29 92 95 58 52 115 111 98 108 33 38 90 22 106 69 14 116 10 53 21 119 81 91 50 45 72 35 97 122 5 6 24 84 57 16 80 18 20 7 125 46 110 27 75 2 79 118 41	<pre> Generation 0, Max fitness: 2 Generation 100, Max fitness: 13 Generation 200, Max fitness: 14 Generation 300, Max fitness: 17 Generation 400, Max fitness: 17 Generation 500, Max fitness: 19 Generation 600, Max fitness: 20 Generation 700, Max fitness: 21 Generation 800, Max fitness: 22 Generation 900, Max fitness: 22 Elapsed time: 2.46556 seconds </pre>

b. Analisis

Dari beberapa percobaan, didapat bahwa algoritma Genetic Algorithm mencapai score sekitar 30-40, tetapi memiliki kemungkinan mencapai global optimum yang lebih tinggi karena mempunyai beberapa cara untuk keluar dari local optimum. Dibandingkan local search lain, yang cenderung lebih cepat tetapi seringkali mudah terjebak pada local optimum, algoritma ini memiliki waktu eksekusi yang lebih lambat tetapi dengan kemampuan keluar dari local optimum dengan crossover, mutasi.

Penambahan populasi menambah keberagaman sehingga algoritma bisa mengeksplor lebih banyak bagian sehingga mengurangi kemungkinan terjebak di local

optimum. Namun, dari percobaan, dapat dilihat bahwa semakin tinggi populasi, kecepatan kenaikan menjadi semakin rendah.

Semakin tinggi jumlah iterasi, semakin baik kualitas solusi yang ditemukan karena GA memiliki lebih banyak kesempatan untuk memperbaiki solusi melalui proses seleksi, crossover, dan mutasi berulang kali. Namun, setelah titik tertentu, penambahan iterasi memberikan hasil yang semakin sedikit (diminishing returns). Algoritma mungkin telah mendekati solusi optimal atau terjebak di sekitar solusi terbaik yang ada. Pada titik ini, meskipun iterasi terus berlanjut, peningkatan kualitas solusi menjadi sangat lambat atau tidak ada.

BAB 3

Kesimpulan dan Saran

3.1 Kesimpulan

Dalam tugas ini, beberapa algoritma local search diterapkan untuk mencari solusi optimal bagi masalah Diagonal Magic Cube, termasuk Steepest Ascent Hill Climbing, Simulated Annealing, dan Genetic Algorithm. Setiap algoritma ini menunjukkan kekuatan serta kelemahannya masing-masing berdasarkan hasil eksperimen.

Steepest Ascent Hill Climbing memperlihatkan keunggulan dalam kecepatan eksekusi, karena algoritma ini secara konsisten memilih successor terbaik pada setiap langkahnya. Namun, algoritma ini cenderung terjebak pada local optima, yang membuatnya sulit untuk mencapai solusi optimal secara keseluruhan. Hasil eksperimen menunjukkan bahwa algoritma ini berhenti di nilai sekitar 30-40, menandakan bahwa proses sering terhenti ketika tidak ada peningkatan lebih lanjut di area yang sedang dicari. Kecepatan algoritma ini memang tinggi, tetapi keterbatasannya terletak pada kurangnya fleksibilitas untuk menjelajahi seluruh ruang pencarian secara optimal.

Sementara itu, Simulated Annealing memiliki kelebihan dalam menerima solusi yang kurang optimal pada awal pencarian, sehingga lebih efektif dalam menghindari local optima dibandingkan algoritma lainnya. Namun, seiring berjalannya waktu dan menurunnya suhu (temperature), algoritma ini menjadi lebih rentan terhadap jebakan local optima. Eksperimen menunjukkan bahwa Simulated Annealing umumnya mencapai nilai akhir yang lebih rendah dibandingkan Steepest Ascent Hill Climbing. Meski begitu, algoritma ini tetap cepat karena tidak memerlukan pemilihan successor terbaik dan cukup mengambil solusi berikutnya secara acak.

Genetic Algorithm menawarkan kemampuan eksplorasi ruang solusi yang lebih luas berkat proses seleksi, crossover, dan mutasi, sehingga mampu mencapai solusi yang lebih optimal pada iterasi yang lebih panjang. Namun, eksperimen menunjukkan bahwa algoritma ini cenderung berhenti pada nilai sekitar 10, yang mengindikasikan pentingnya konfigurasi parameter yang tepat untuk hasil yang lebih baik. Fleksibilitas Genetic Algorithm memang menjadi kelebihan dalam mendekati solusi optimal, tetapi diperlukan pengaturan parameter seperti ukuran populasi dan jumlah iterasi yang cermat agar potensi pencariannya benar-benar maksimal.

3.2 Saran

Berdasarkan hasil eksperimen, beberapa algoritma misalnya Steepest Ascent Hill Climbing dan Simulated Annealing menunjukkan pola performa yang berbeda dengan teori di kelas. Steepest Ascent sering dinilai sebagai algoritma terburuk sebab probabilitas terjebak pada local optimanya sangat tinggi. Akan tetapi, eksperimen menunjukkan bahwa algoritma tersebut berhasil memberikan hasil yang lebih baik bahkan apabila dibandingkan dengan Simulated Annealing. Disamping itu, ketiga algoritma yang kami gunakan tidak ada satupun yang berhasil

memperoleh hasil akhir yang mendekati nilai Magic Number. Maka dari itu, disarankan untuk melakukan peningkatan melalui eksplorasi dengan algoritma lain misalnya Stochastic Hill Climbing. Di samping itu, pembuatan parameter keberhasilan yang lebih konkret sangat disarankan agar dalam membandingkan masing-masing algoritma dapat dilakukan dengan objektif. Sebab, tentu akan ada trade-off antara kualitas solusi dengan waktu komputasi, terlebih lagi pada kasus Magic Cube yang termasuk ke dalam kasus berukuran besar.

Pembagian Tugas

NIM	Tugas
13521117	Laporan
13520020	Visualisasi, algoritma
13522040	Laporan
13522090	Laporan, algoritma
13522101	Laporan, algoritma

Referensi

https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/194228-Beyond-Classical-Search/1693804818592_IF3170_Materi03_Seg01_BeyondClassicalSearch_LocalSearch.pdf
https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/194228-Beyond-Classical-Search/1693804849395_IF3170_Materi3_Seg03_BeyondClassicalSearch_HillClimbing.pdf
https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/194228-Beyond-Classical-Search/1693804872404_IF3170_Materi03_Seg04_BeyondClassicalSearch_SimulatedAnnealing.pdf
https://cdn-edunex.itb.ac.id/storages/files/1727405202098_IF3170_Materi03_Seg05_BeyondClassicalSearch.pdf