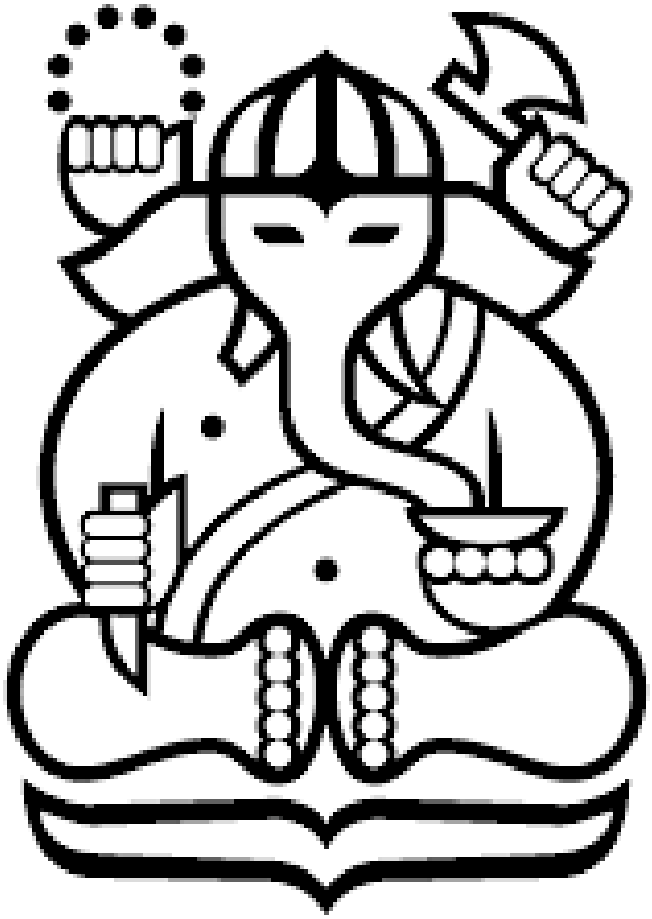
Laporan Tugas Besar 1
IF3170 Inteligensi Artifisial

Pencarian Solusi Diagonal Magic Cube dengan Local Search



Disusun oleh:

Maria Flora Renata Siringoringo	13522010
M Athaullah Daffa Kusuma Mulia	13522044
Dzaky Satrio Nugroho	13522059
Rafiki Prawira Harianto	13522065

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

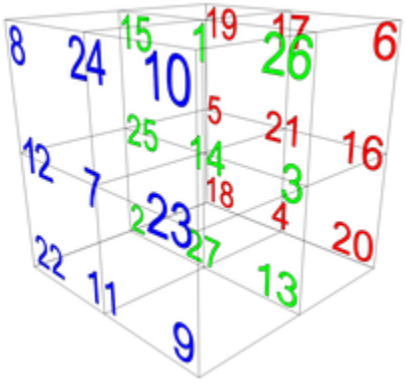
Daftar Isi

Bab 1: Deskripsi Persoalaan

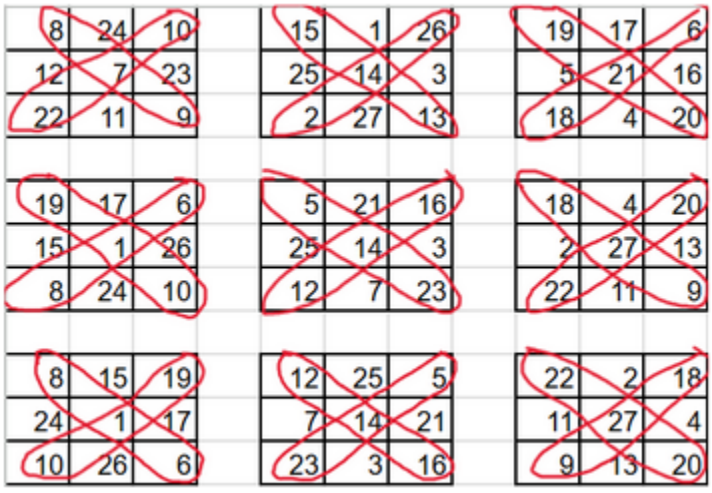
Diagonal magic cube merupakan kubus yang tersusun dari angka 1 hingga n^3 tanpa pengulangan dengan n adalah panjang sisi pada kubus tersebut. Angka-angka pada tersusun sedemikian rupa sehingga properti-properti berikut terpenuhi:

- 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:



Diagonal yang dimaksud adalah yang dilingkari warna merah saja

Pada tugas ini, akan diselesaikan permasalahan Diagonal Magic Cube berukuran 5x5x5. Initial state dari suatu kubus adalah susunan angka 1 hingga 125 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. *Objective Function*

Objective function yang dipilih adalah banyak rusuk dan diagonal *magic cube* yang memiliki jumlah nilai yang sama paling banyak.

2.2. Implementasi algoritma local search

Implementasi algoritma ini didasari pada kelas magicCube.

magic-cube.hpp

```
class magicCube {
    public:
        vector<int> state;
        int value;

        magicCube(){
            for (int i = 1; i <= 125; i++) this->state.push_back(i);
            // Random seed
            srand(time(nullptr));
            random_shuffle(state.begin(), state.end());
            this->value = this->countValue();
        }
```

```
magicCube(vector<int> state) {
    this->state = state;
    this->value = this->countValue();
}

vector<int> makeNeighborState(int num1, int num2) {
    vector<int> neighborState = this->state;
    int temp = neighborState[num1];
    neighborState[num1] = neighborState[num2];
    neighborState[num2] = temp;
    return neighborState;
}

vector<int> makeRandomNeighborState() {
    int num1 = rand() % 124;
    int num2 = rand() % 124;
    while (num1 == num2) num2 = rand() % 124;
    return (this->makeNeighborState(num1, num2));
}

void switchStates(vector<int> newState) {
    this->state.swap(newState);
    this->value = this->countValue();
}
```

```
// Fungsi untuk menghitung value dari state sekarang
int countValue(){
    ...
}

// Fungsi untuk print state ke terminal
void printCube(){
    ...
}
```

Kelas hillClimb untuk algoritma Hill Climb.

```
hillclimb.hpp
```

```
class hillClimb {
    public:
        magicCube magiccube;

        hillClimb(){
            this->magiccube = magicCube();
        }

        // Mengembalikan neighbor, yaitu successor dengan value terbesar
        magicCube highestValueNeighbor(magicCube current) {
            int maxvalue = 0, curvalue;
            magicCube neighbor = current,temp;

            for (int i = 0; i < 125; i++) {
                for (int j = i + 1; j < 125; j++) {
                    vector<int> successor = current.state;
                    swap(successor[i], successor[j]);
                    temp.state = successor;
                    curvalue = temp.countValue();
                    temp.value = curvalue;
                    if(temp.countValue() > maxvalue){
                        maxvalue = curvalue;
                        neighbor = temp;
                    }
                }
            }

            return neighbor;
        }
```

2.2.1. Hill Climb : Steepest Ascent

```
void steepestAscent(){
    // Generate random cube
    random_shuffle(this->magiccube.state.begin(),
this->magiccube.state.end());
    this->magiccube.value = this->magiccube.countValue();

    // Print state awal
    cout << "State Awal :\n";
    this->magiccube.printCube();

    // Start timer
    auto start = high_resolution_clock::now();
```

```
    // Fungsi Steepest Ascent
    int iterasi = 0;
    vector<int> plot;
    bool peak = false;
    plot.push_back(this->magiccube.value);
    while(!peak){
        magicCube neighbor = highestValueNeighbor(this->magiccube);
        if(neighbor.value <= this->magiccube.value){
            peak = true;
        } else {
            this->magiccube = neighbor;
        }
        iterasi++;
        plot.push_back(this->magiccube.value);
    }

    // Stop timer
    auto end = high_resolution_clock::now();

    // Hitung waktu
    auto duration = duration_cast<milliseconds>(end - start);

    // Print state akhir
    cout << "State Akhir :\n";
    this->magiccube.printCube();

    // Print info
    cout << "Nilai Akhir Objective Function : " <<
this->magiccube.value << endl;
    cout << "Durasi : " << duration.count() / 1000.0 << " seconds"
<< endl;
    cout << "Banyak Iterasi : " << iterasi << endl;

    cout << plot[0];
    for(int i=0;i<=iterasi;i++){
        cout << "," << plot[i];
    } cout << endl;

    // Delay
    system("pause");
}
```

2.2.2. Hill Climb : with Sideways Move

```
void sideWays(int maximumSidewaysMove){
    // Generate random cube
    random_shuffle(this->magiccube.state.begin(),
this->magiccube.state.end());
```

```
    this->magiccube.value = this->magiccube.countValue();

    // Print state awal
    cout << "State Awal :\n";
    this->magiccube.printCube();

    // Start timer
    auto start = high_resolution_clock::now();

    // Fungsi Steepest Ascent
    int iterasi = 0, counterSideways;
    vector<int> plot;
    bool peak = false;
    plot.push_back(this->magiccube.value);
    while(!peak){
        magicCube neighbor = highestValueNeighbor(this->magiccube);
        if(neighbor.value < this->magiccube.value){
            peak = true;
        } else {
            if(neighbor.value == this->magiccube.value){
                counterSideways++;
            } else {
                counterSideways = 0;
            }
            if(counterSideways == maximumSidewaysMove){
                peak = true;
            } else {
                this->magiccube = neighbor;
            }
        }
        iterasi++;
        plot.push_back(this->magiccube.value);
    }

    // Stop timer
    auto end = high_resolution_clock::now();

    // Hitung waktu
    auto duration = duration_cast<milliseconds>(end - start);

    // Print state akhir
    cout << "State Akhir :\n";
    this->magiccube.printCube();

    // Print info
    cout << "Nilai Akhir Objective Function : " <<
this->magiccube.value << endl;
    cout << "Durasi : " << duration.count() / 1000.0 << " seconds"
<< endl;
    cout << "Banyak Iterasi : " << iterasi << endl;
```

```
    cout << plot[0];
    for(int i=0;i<=iterasi;i++){
        cout << "," << plot[i];
    } cout << endl;

    // Delay
    system("pause");
}
```

2.2.3. Hill Climb : Random Restart

```
        void randomRestart(int maximumRestart){
            int maxvalue = 0,restart;
            vector<int> iterasi;
            vector<vector<int>> plot;

            // Print state awal
            cout << "State Awal :\n";
            this->magiccube.printCube();

            // Start timer
            auto start = high_resolution_clock::now();

            for(restart=0;restart<maximumRestart;restart++){
                // Generate random cube
                random_shuffle(this->magiccube.state.begin(),
this->magiccube.state.end());
                this->magiccube.value =
this->magiccube.countValue();
                // Fungsi Steepest Ascent
                int iterasii = 0;
                vector<int> plottemp;
                plottemp.clear();
                bool peak = false;
                plottemp.push_back(this->magiccube.value);
                while(!peak){
                    magicCube neighbor =
highestValueNeighbor(this->magiccube);
                    if(neighbor.value <= this->magiccube.value){
                        peak = true;
                    } else {
                        this->magiccube = neighbor;
                    }
                    iterasii++;
                    plottemp.push_back(this->magiccube.value);
                }
                plot.push_back(plottemp);
                iterasi.push_back(iterasii);
```

```
                cout << "iterasi ke-" << restart+1 << endl;
                cout << plottemp[0];
                for(int i=1;i<=iterasii;i++){
                    cout << "," << plottemp[i];
                } cout << endl;

                if(maxvalue < this->magiccube.value){
                    maxvalue = this->magiccube.value;
                }
            }

            // Stop timer
            auto end = high_resolution_clock::now();

            // Hitung waktu
            auto duration = duration_cast<milliseconds>(end -
start);

            // Print state akhir
            cout << "State Akhir :\n";
            this->magiccube.printCube();

            // Print info
            cout << "Nilai Akhir Objective Function : " <<
this->magiccube.value << endl;
            cout << "Durasi : " << duration.count() / 1000.0 << "
seconds" << endl;
            cout << "Iterasi per restart :\n";
            for(int i=0;i<restart;i++){
                cout << "restart-" << i+1 << " : " << iterasi[i] <<
endl;
            }

            for(int i=0;i<restart;i++){
                cout << "restart-" << i+1 << endl;
                int j=0;
                for(int plotting : plot[i]){
                    cout << j << "," << plotting << endl;
                    j++;
                }
            }

            cout << "Maximum value : " << maxvalue << endl;

            // Delay
            system("pause");
        }
```

2.2.4. Hill Climb : Stochastic

```
void stochastic(int maximumMove){
    // Generate random cube
    random_shuffle(this->magiccube.state.begin(),
this->magiccube.state.end());
    this->magiccube.value = this->magiccube.countValue();

    // Print state awal
    cout << "State Awal :\n";
    this->magiccube.printCube();

    // Start timer
    auto start = high_resolution_clock::now();

    // Fungsi Stochastic
    int iterasi = 0;
    vector<int> plot;
    bool peak = false;
    plot.push_back(this->magiccube.value);
    while(maximumMove--){
        magicCube neighbor = this->magiccube;

neighbor.switchStates(this->magiccube.makeRandomNeighborState());
        if(neighbor.value > this->magiccube.value){
            this->magiccube = neighbor;
        }
        iterasi++;
        plot.push_back(this->magiccube.value);
    }

    // Stop timer
    auto end = high_resolution_clock::now();

    // Hitung waktu
    auto duration = duration_cast<milliseconds>(end - start);

    // Print state akhir
    cout << "State Akhir :\n";
    this->magiccube.printCube();

    // Print info
    cout << "Nilai Akhir Objective Function : " <<
this->magiccube.value << endl;
    cout << "Durasi : " << duration.count() / 1000.0 << " seconds"
<< endl;
    cout << "Banyak Iterasi : " << iterasi << endl;

    cout << plot[0];
    for(int i=0;i<=iterasi;i++){
        cout << "," << plot[i];
```

```
    } cout << endl;

    // Delay
    system("pause");
}
```

2.2.5. Simulated Annealing

Kelas simulated annealing pada simulated-annealing.hpp, dengan fungsi schedule berupa $T = 0.6^{t-1}$

```
class SimulatedAnnealing {
private:
    magicCube cube;
    double temperature;
    vector<int> plot;
    vector<double> plotexp;
    int freqStuck;

    double schedule(int t) {
        return pow(0.6,t-1);
    }
```

Algoritma Utama:

```
public:
    SimulatedAnnealing() : temperature(1.0), freqStuck(0) {}

    void run(int maxIterations) {
        plot.push_back(cube.value);
        for (int t = 1; t <= maxIterations; ++t) {
            temperature = schedule(t);
            if (temperature <= 0) break;

            magicCube successor = cube;
            successor.switchStates(cube.makeRandomNeighborState());
            int deltaE = successor.value - cube.value;

            if (deltaE > 0) {
                cube.switchStates(successor.state);
            } else {
                freqStuck++;
                double acceptanceProb;
                if (temperature <= 0) acceptanceProb = 0;
                else{
                    acceptanceProb = exp(deltaE / temperature);
                    if (acceptanceProb > 1) acceptanceProb = 1;
                }

                plotexp.push_back(acceptanceProb);
                if (((double)rand() / (double)RAND_MAX) < acceptanceProb) {
                    cube.switchStates(successor.state);
                }
            }
            plot.push_back(cube.value);
        }
    }
```

Kakas untuk algoritma utama:

```
void start(){
    // Clear terminal
    system("cls");

    // Print state awal
    cout << "State Awal :\n";
    cube.printCube();

    // Start timer
    auto start = high_resolution_clock::now();

    // Fungsi Simulated-Annealing
    this->run(1000);

    // Stop timer
    auto end = high_resolution_clock::now();

    // Hitung waktu
    auto duration = duration_cast<milliseconds>(end - start);

    // Print state akhir
    cout << "State Akhir :\n";
    this->cube.printCube();

    // Print info
    cout << "Nilai Akhir Objective Function : " << this->cube.value << endl;
    cout << "Durasi : " << duration.count() / 1000.0 << " seconds" << endl;
    cout << "Frekuensi Stuck : " << freqStuck << endl;
    cout << "Plot Objective Function : " << endl; printVectorInt(plot);
    cout << "Plot exp(deltaE/T) : " << endl; printVectorDouble(plotexp);
}
```

Isi main pada simmulated-annealing.cpp

```
#include "simmulated-annealing.hpp"

int main(){
    SimulatedAnnealing SA;

    system("cls");
    cout << "Algoritma Simmulated Annealing\n";
    SA.start();
    exit(0);
}
```

2.2.6. Genetic Algorithm

genetic-algorithm.hpp

```
#ifndef GENETICALGORITHM_HPP
#define GENETICALGORITHM_HPP

#include <bits/stdc++.h>
#include "magic-cube.hpp"
using namespace std;

class geneticAlgorithm {
    private:
        vector<magicCube> population;
        int iterations;
        int popcount;

    public:
        geneticAlgorithm(int i, int p);
        void initPopulation(magicCube origin, int count);
        int selection();
        vector<vector<int>> crossover(magicCube parent1, magicCube
parent2);
        void mutate(magicCube* child);
        void go(int e);
        int bestIndividual();
        double meanValues();
};

#endif
```

genetic-algorithm.cpp

```
#include "genetic-algorithm.hpp"
#include <fstream>
using namespace std;

void printVec(vector<int> vec, int ex) {
    string filename = "max";
    filename.append(to_string(ex));
    ofstream maxFile(filename);

    // Write to the file
    for (int i=0; i<vec.size(); i++) {
        maxFile << vec[i] << ",";
    }
    maxFile << vec[124] << endl;

    // Close the file
    maxFile.close();
}
```

```
void printVecD(vector<double> vec, int ex) {
    string filename = "mean";
    filename.append(to_string(ex));
    ofstream meanFile(filename);
    // Write to the file
    for (int i=0; i<vec.size(); i++) {
        meanFile << vec[i] << ",";
    }
    meanFile << vec[124] << endl;

    // Close the file
    meanFile.close();
}

geneticAlgorithm::geneticAlgorithm(int i, int p) {
    magicCube origin = magicCube();
    this->iterations = i;
    this->popcount = p;
    this->initPopulation(origin, p);
    srand(time(NULL)); //setting seed
    cout << "Nilai Awal Objective Function: " << origin.value <<
endl;
    origin.printCube();
}

void geneticAlgorithm::initPopulation(magicCube origin, int count)
{
    vector<magicCube> popul;
    for (int i=0; i<count; i++)
popul.push_back(magicCube(origin.makeRandomNeighborState()));
    this->population = popul;
}

int geneticAlgorithm::selection() {
    double selector = static_cast <double> (rand()) / static_cast
<double> (RAND_MAX);
    double curval = 0;
    double sumvalues = 0;

    for (int i=0; i<popcount; i++) {
        sumvalues += population[i].value;
    }

    for (int i=0; i<popcount; i++) {
        curval += population[i].value/sumvalues;
        if (selector < curval) {
            return i;
        }
    }
    return popcount-1;
```

```
}

vector<vector<int>> geneticAlgorithm::crossover(magicCube parent1,
magicCube parent2) {
    int point = rand() % 124;

    vector<vector<int>> res;
    vector<int> child1;
    vector<int> child2;

    child1.insert(child1.begin(), parent1.state.begin(),
parent1.state.begin()+point);
    child1.insert(child1.end(), parent2.state.begin()+point,
parent2.state.end());
    res.push_back(child1);

    child2.insert(child2.begin(), parent2.state.begin(),
parent2.state.begin()+point);
    child2.insert(child2.end(), parent1.state.begin()+point,
parent1.state.end());
    res.push_back(child2);
    return res;
}

void geneticAlgorithm::mutate(magicCube* child) {
    double chance = static_cast <double> (rand()) / static_cast
<double> (RAND_MAX);
    if (chance > 0.95)
child->switchStates(child->makeRandomNeighborState());
}

void geneticAlgorithm::go(int exp) {
    // Start timer
    auto start = chrono::high_resolution_clock::now();

    //vector untuk menyimpan value per iterasi
    vector<int> maxVals;
    vector<double> meanVals;

    for (int i=0; i<this->iterations; i++) {
        int selection1 = this->selection();
        int selection2 = this->selection();
        while (selection2==selection1) selection2 =
this->selection();

        magicCube parent1 = this->population[selection1];
        magicCube parent2 = this->population[selection2];

        vector<vector<int>> cross = this->crossover(parent1,
parent2);
        magicCube child1 = magicCube(cross[0]);
```

```
        magicCube child2 = magicCube(cross[1]);

        this->mutate(&child1);
        this->mutate(&child2);

        this->population.push_back(child1);
        this->population.push_back(child2);
        this->popcount += 2;

maxVals.push_back(this->population[this->bestIndividual()].value);
        meanVals.push_back(this->meanValues());
    }
    // Stop timer
    auto end = chrono::high_resolution_clock::now();

    // Hitung waktu
    auto duration = chrono::duration_cast<chrono::milliseconds>(end
- start);
    int best_idx = this->bestIndividual();
    cout << "Nilai Akhir Objective Function : " <<
this->population[best_idx].value << endl;
    this->population[best_idx].printCube();
    cout << "Durasi : " << duration.count() / 1000.0 << " seconds"
<< endl;
    cout << "Banyak Iterasi : " << this->iterations << endl;
    printVec(maxVals, exp);
    printVecD(meanVals, exp);

}

int geneticAlgorithm::bestIndividual() {
    int max = 0;
    int idx = 0;
    for (int i=1; i<this->popcount; i++) {
        if (this->population[i].value > max) {
            idx = i;
            max = this->population[i].value;
        }
    }
    return idx;
}

double geneticAlgorithm::meanValues() {
    double sum = 0;
    for (int i=1; i<this->popcount; i++)
sum+=this->population[i].value;
    return sum/popcount;
}

int main() {
    int iterations, size, experiment;
```

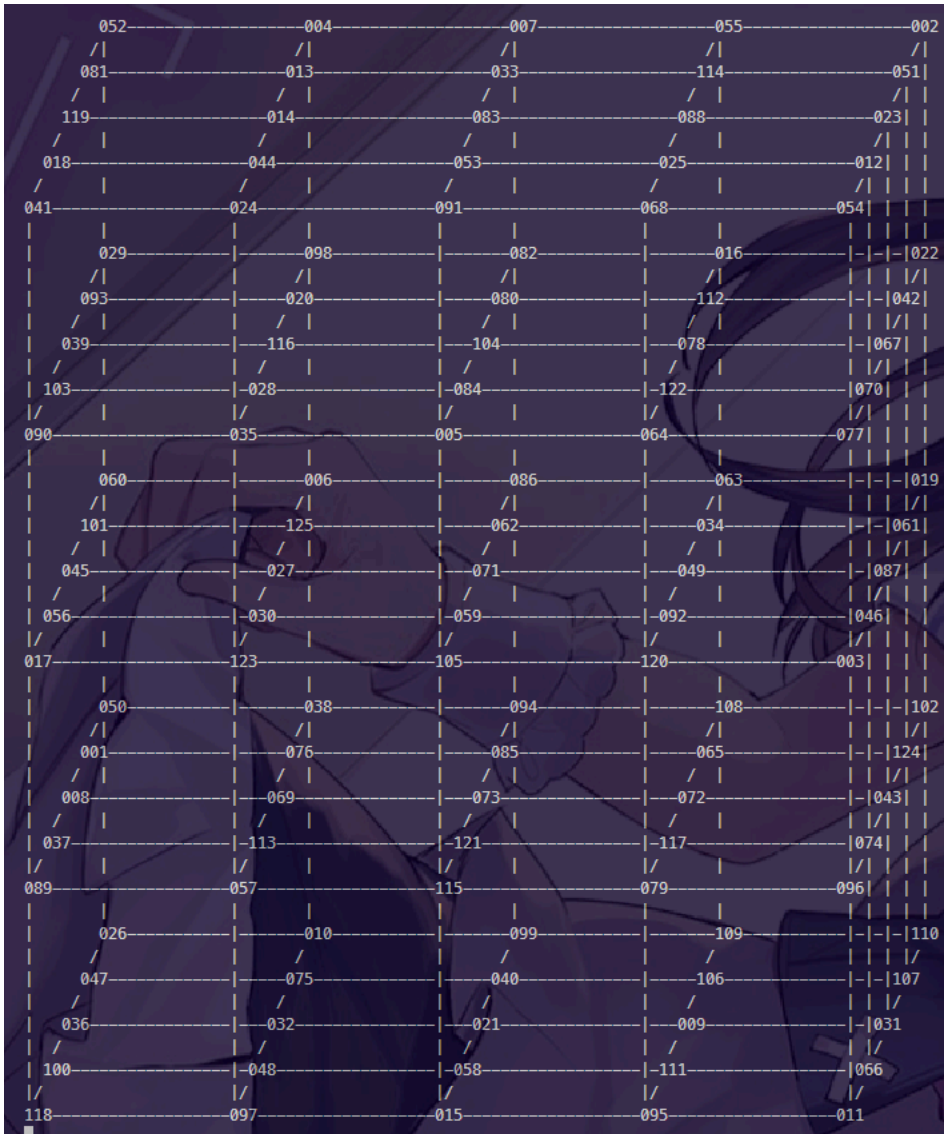
```
    cout << "Masukkan angka eksperimen: ";
    cin >> experiment;
    cout << "Masukkan jumlah iterasi: ";
    cin >> iterations;
    cout << "Masukkan jumlah populasi awal: ";
    cin >> size;
    geneticAlgorithm genalg = geneticAlgorithm(iterations, size);
    genalg.go(experiment);
}
```

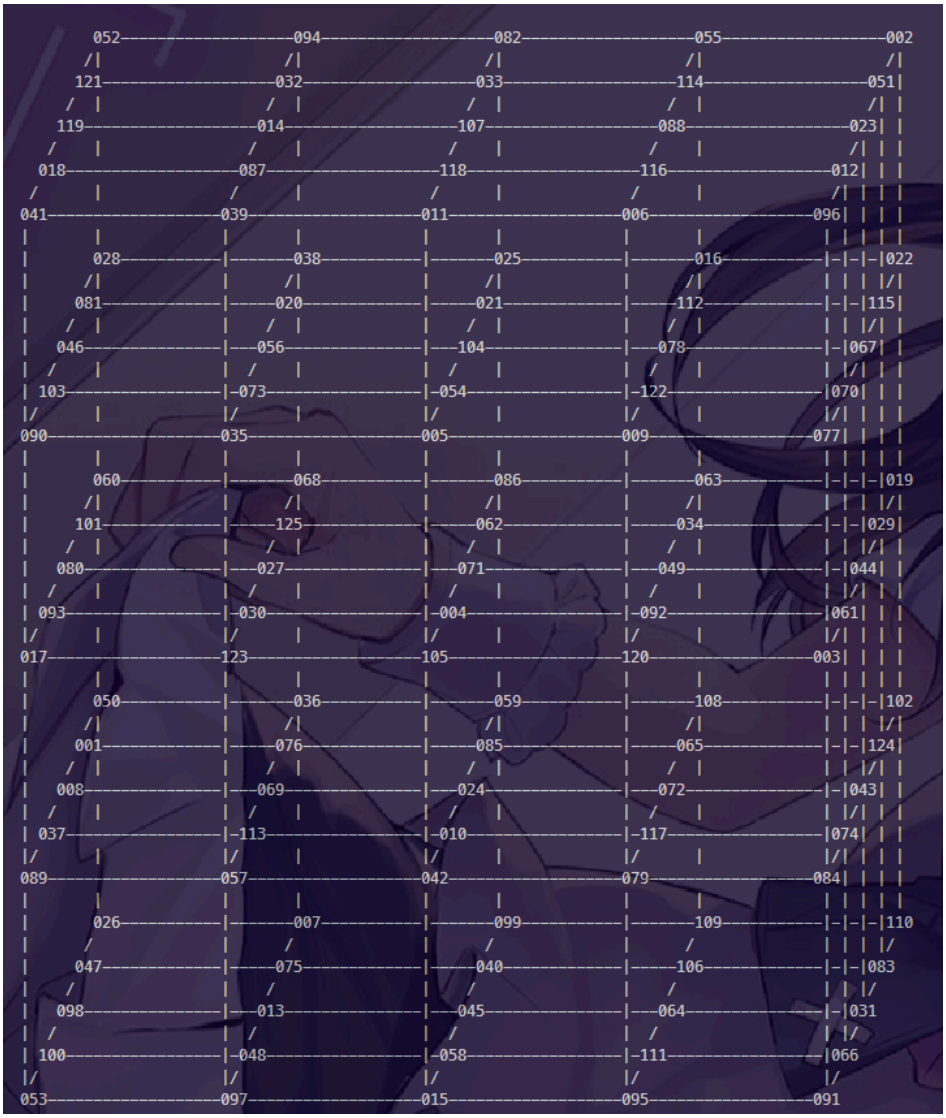
2.3. Hasil Eksperimen dan Analisis

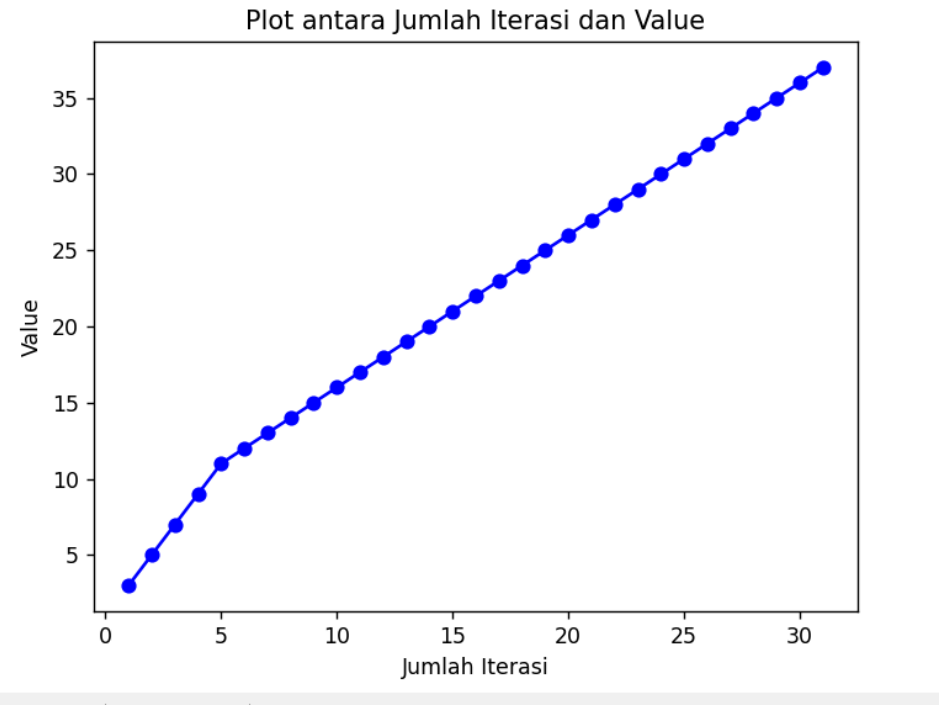
2.3.1. Eksperimen

- Hill Climb : Steepest Ascent

  Percobaan 1

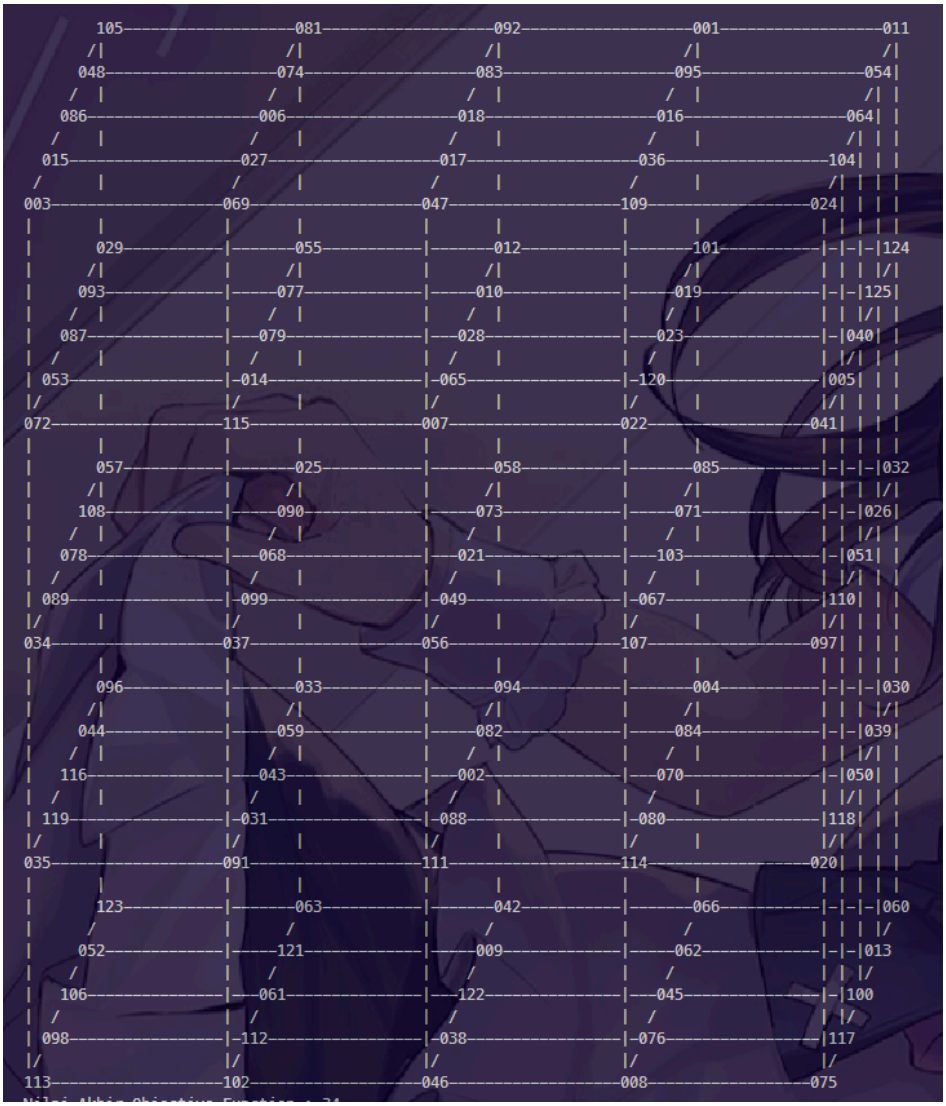
State Awal	

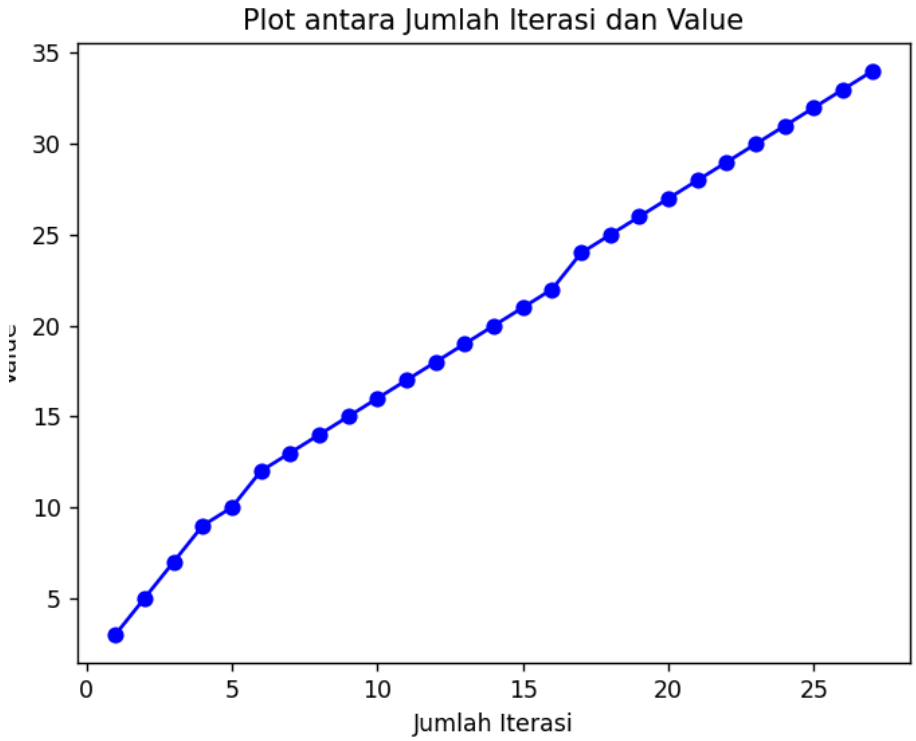
State Akhir	
Nilai Akhir Objective Function	37

Plot	
Durasi	11.658 detik
Banyak Iterasi	28

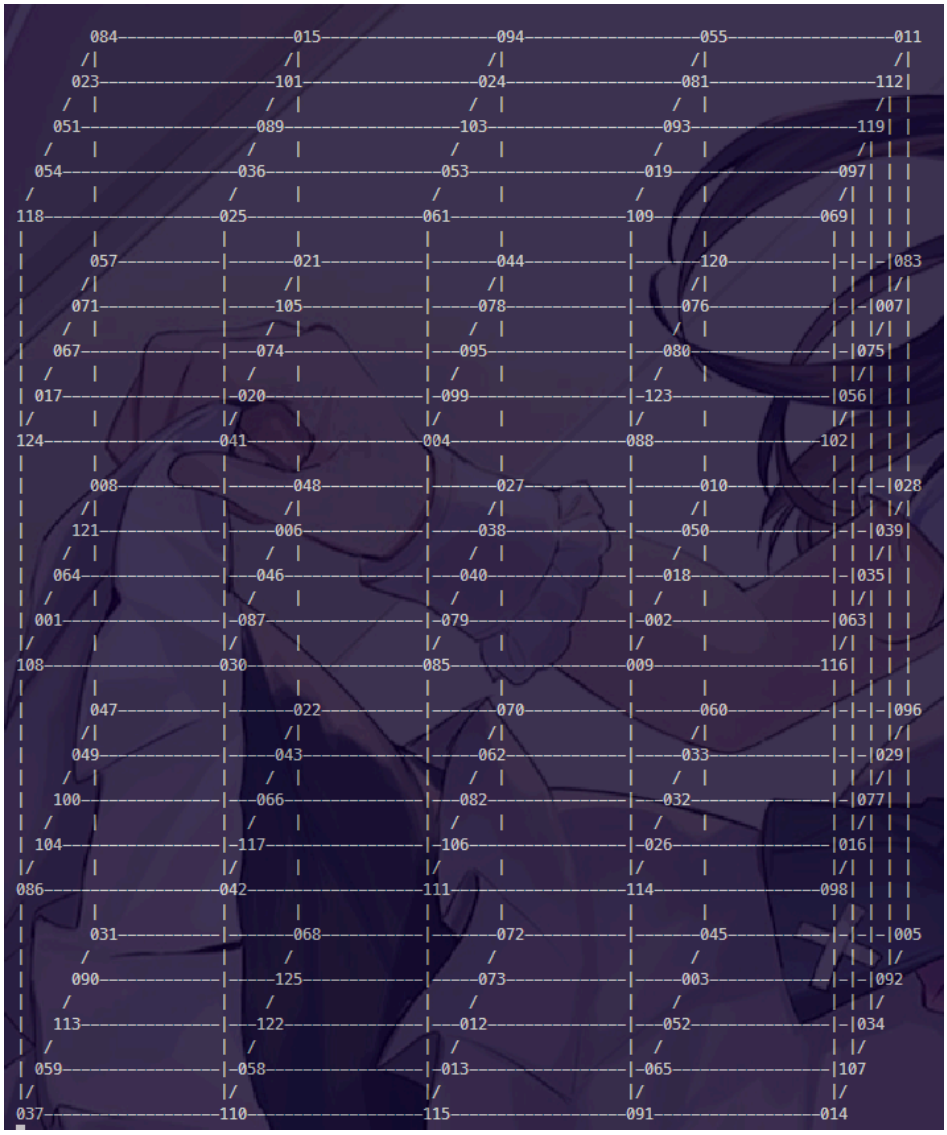
Percobaan 2

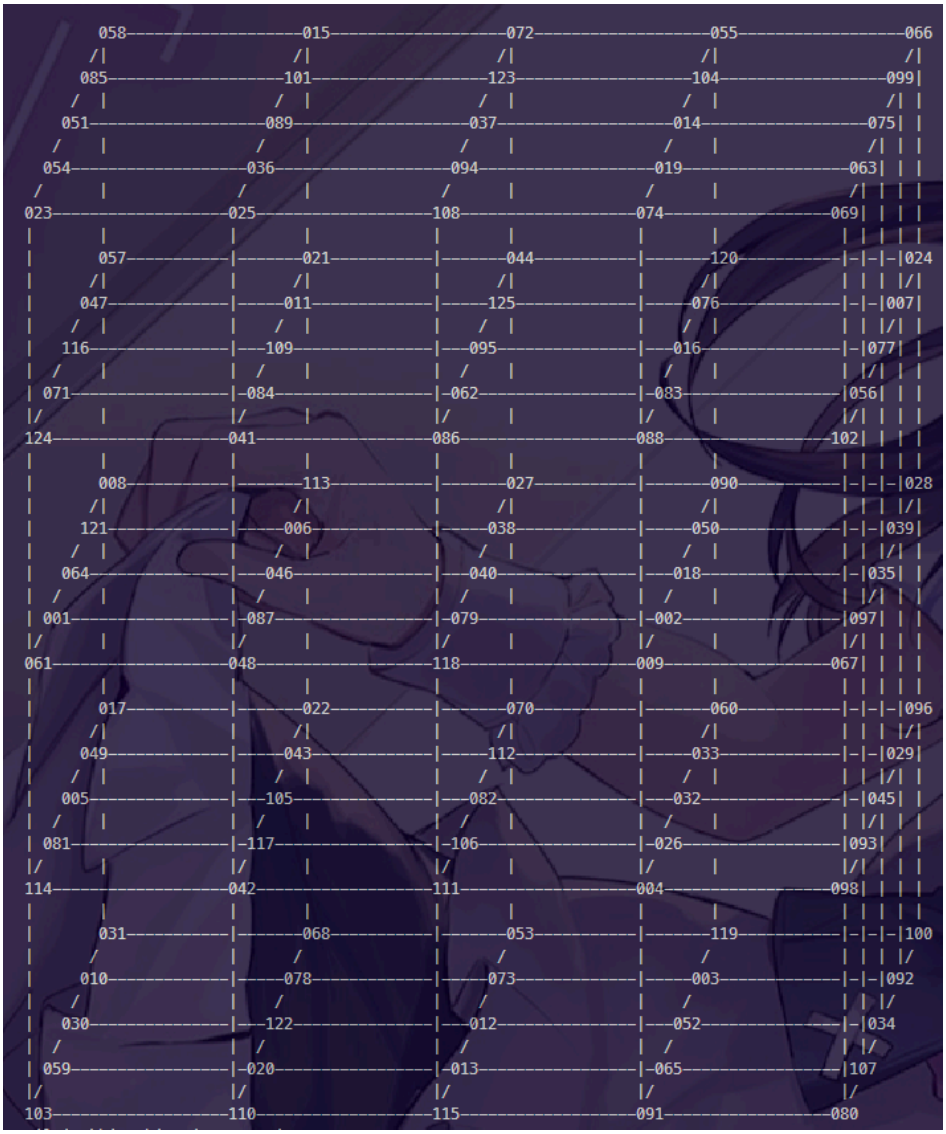
State Awal	

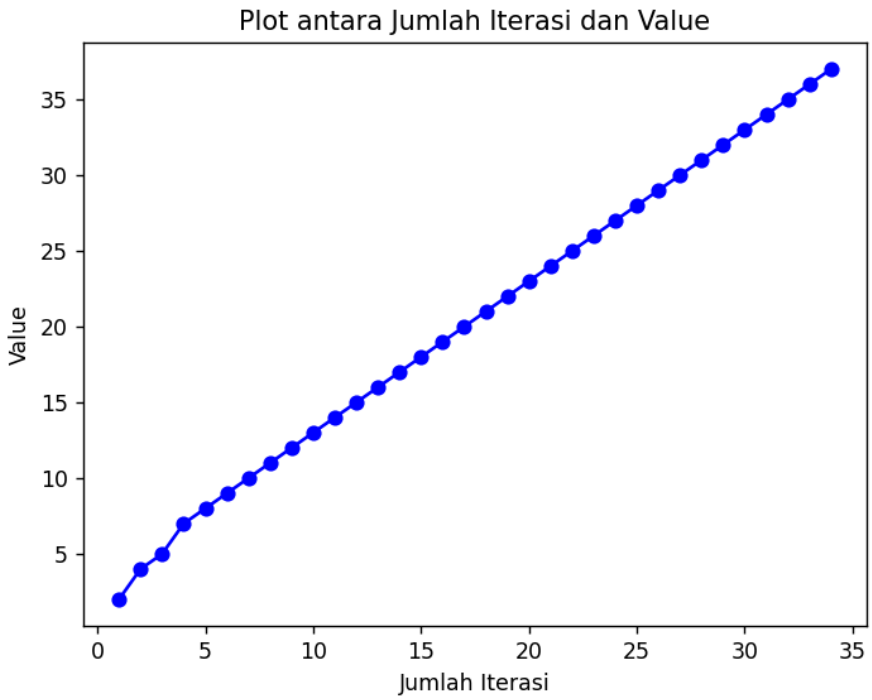
State Akhir	
Nilai Akhir Objective Function	34

Plot	
Durasi	9.968
Banyak Iterasi	27

Percobaan 3

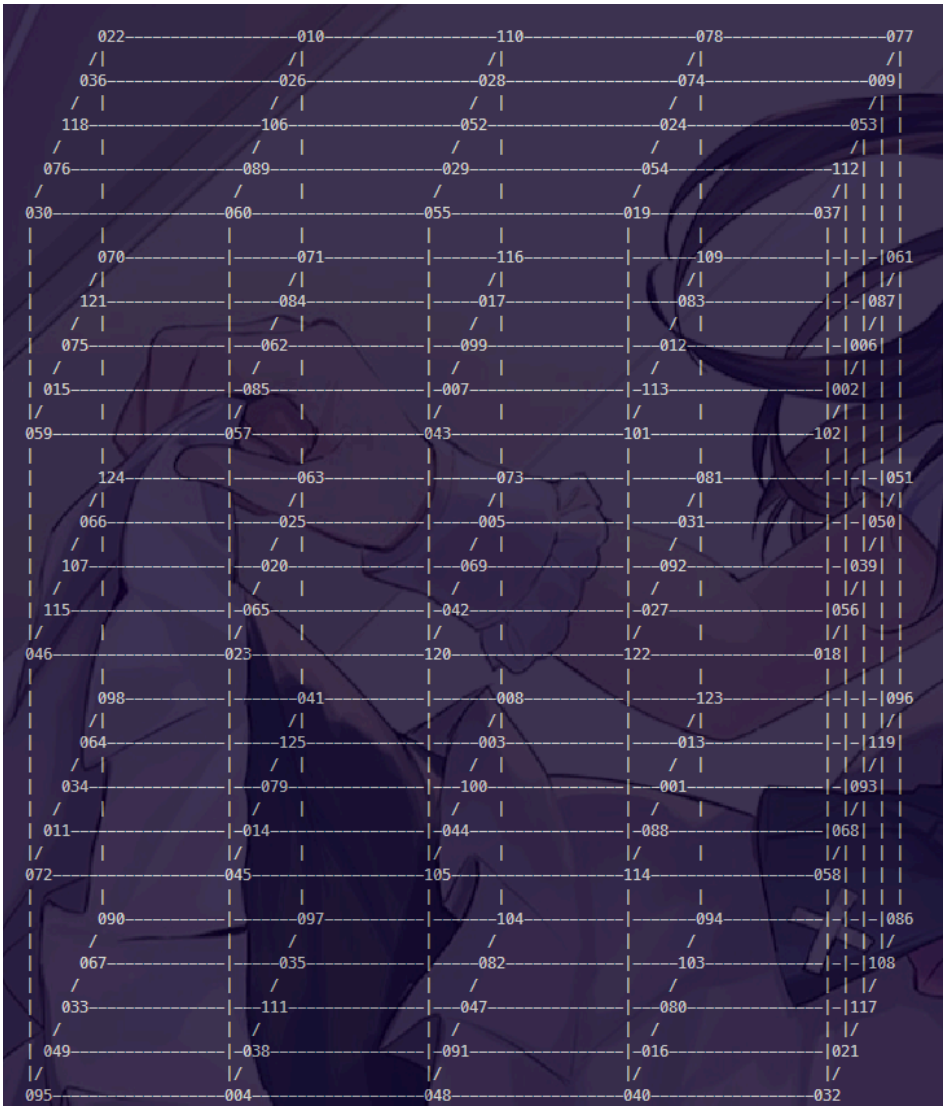
State Awal	

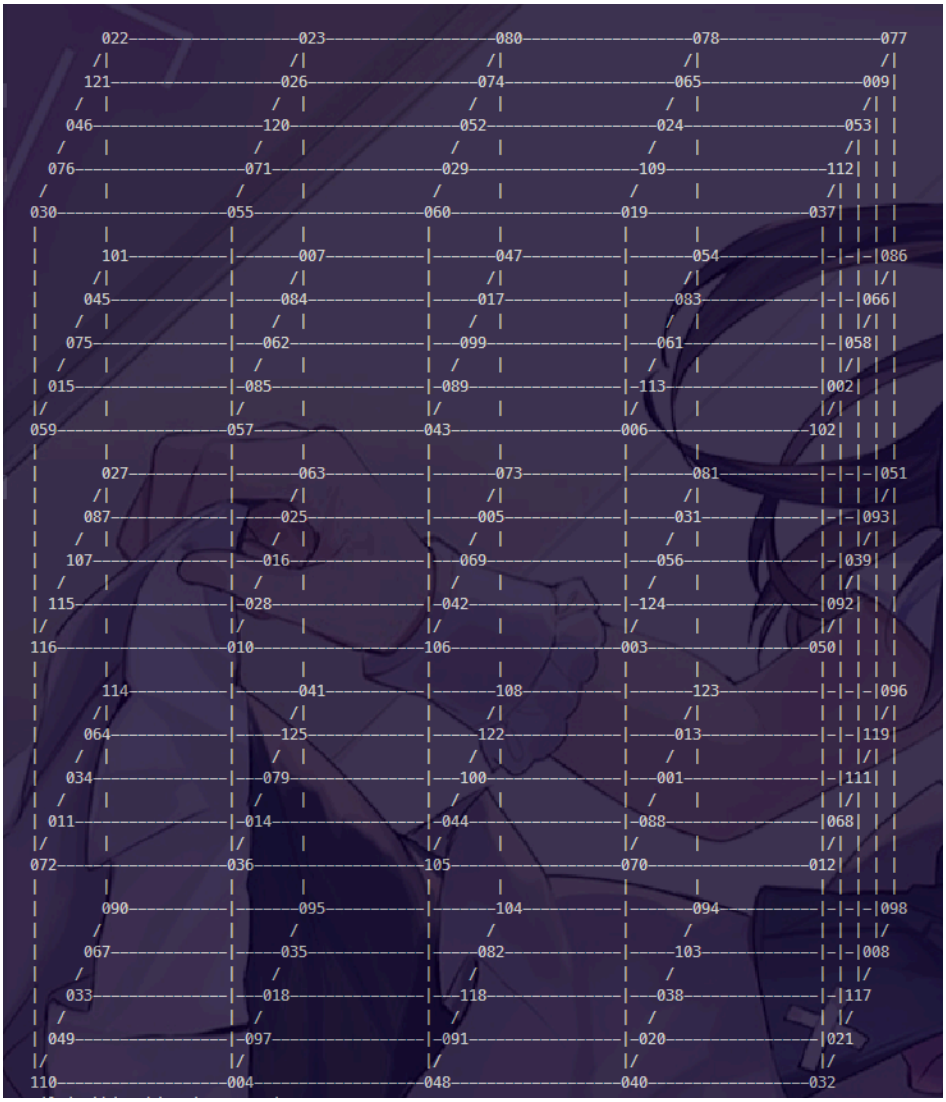
State Akhir	
Nilai Akhir Objective Function	37

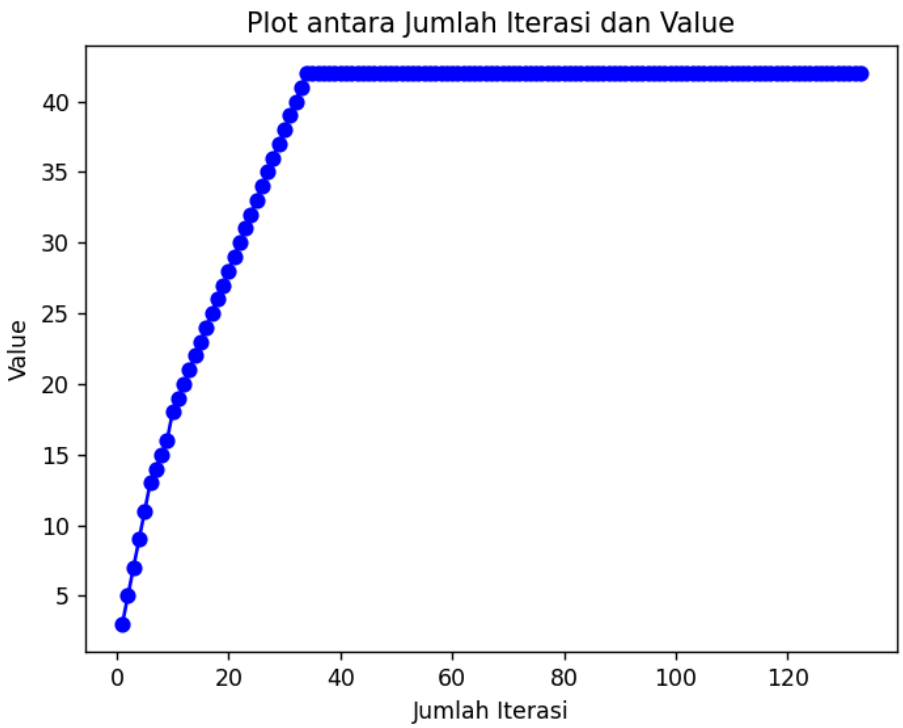
Plot	
Durasi	13.486 detik
Banyak Iterasi	34

- Hill Climb : with Sideways Move

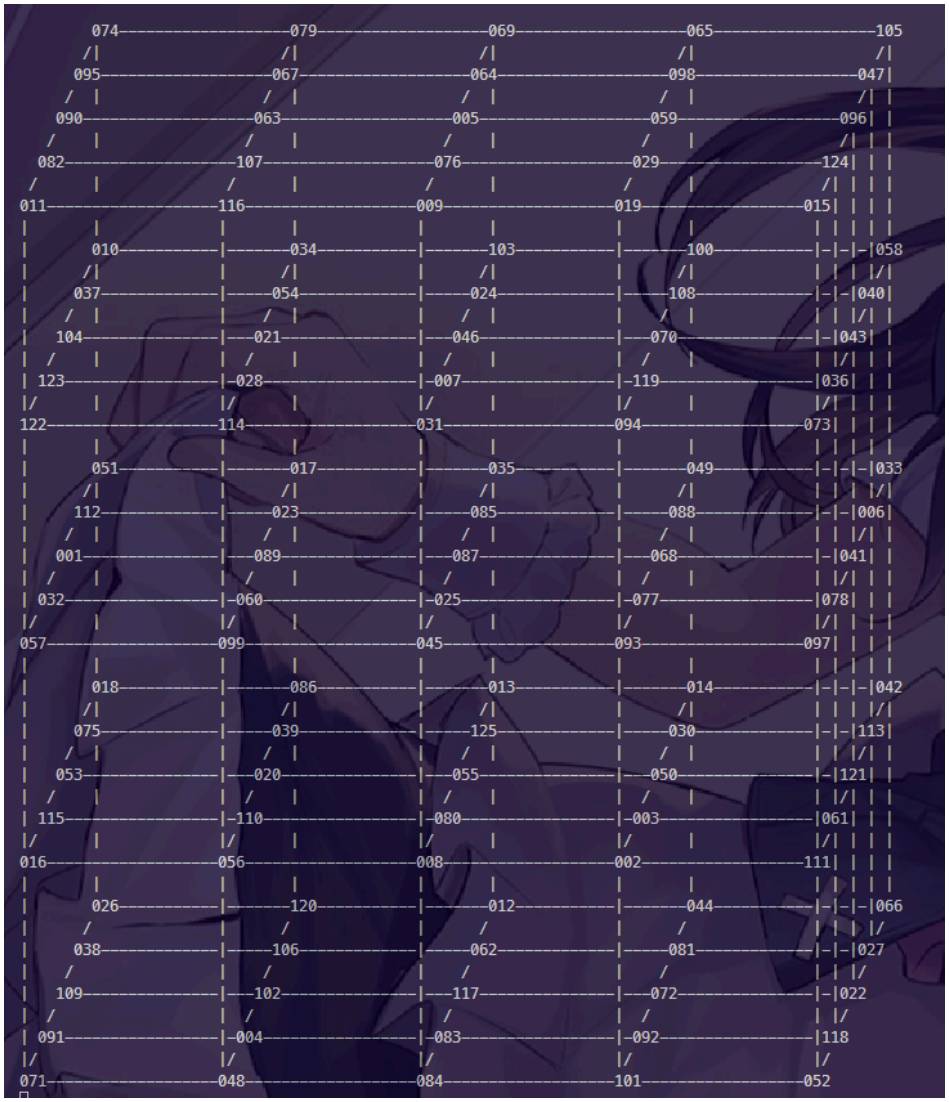
  Percobaan 1

State Awal	

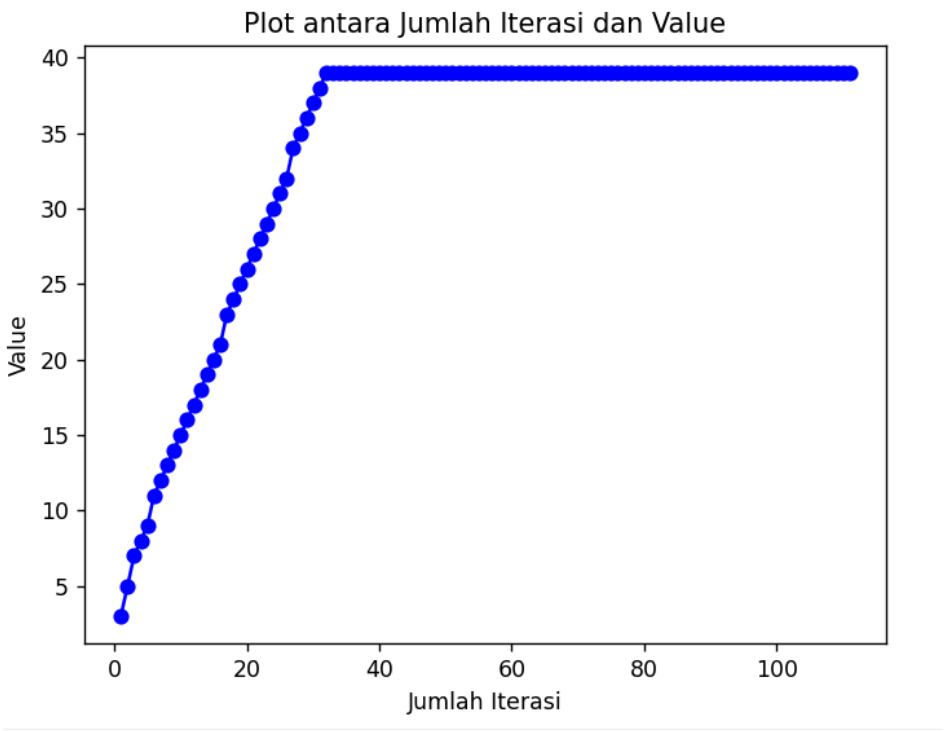
State Akhir	
Nilai Akhir Objective Function	42

Plot	
Durasi	44.629 detik
Banyak Iterasi	133
Parameter Maksimum	100

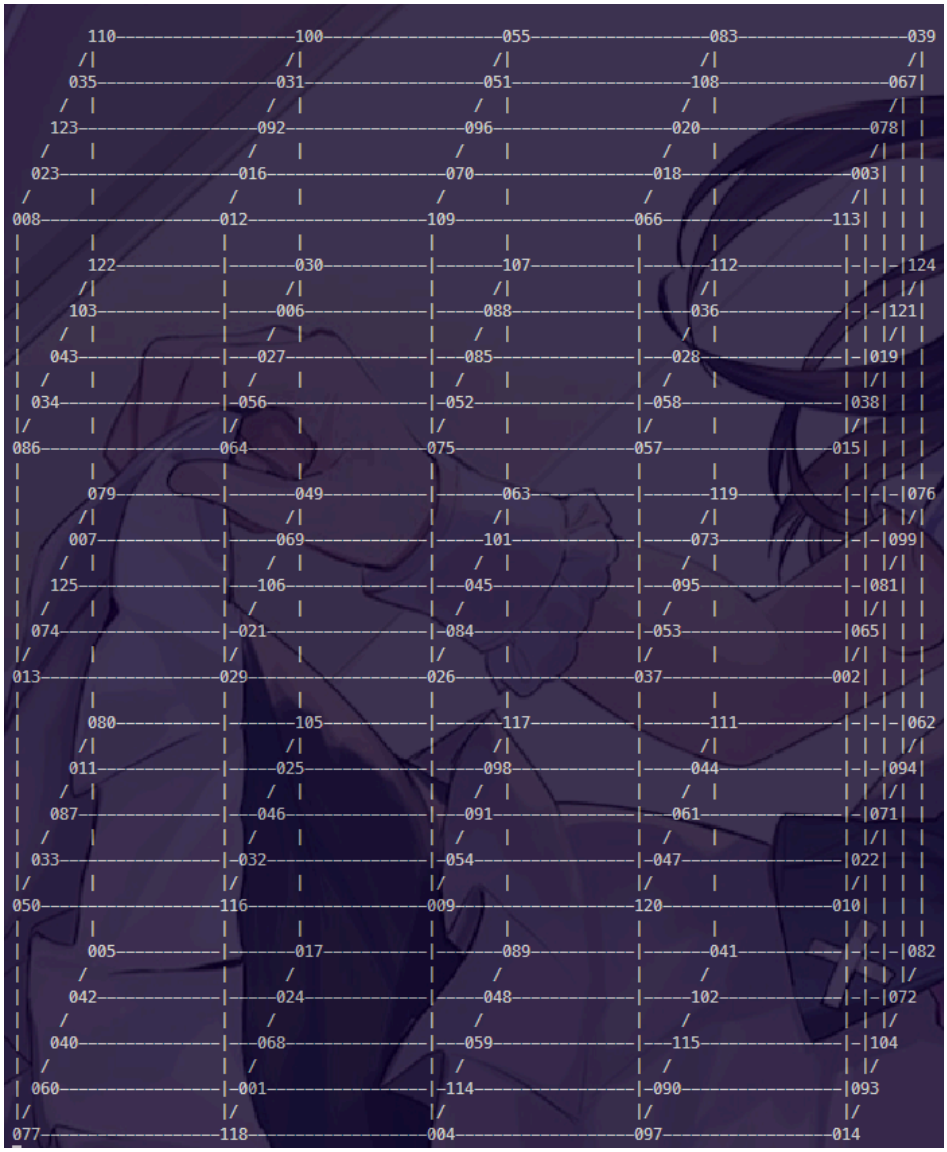
Percobaan 2

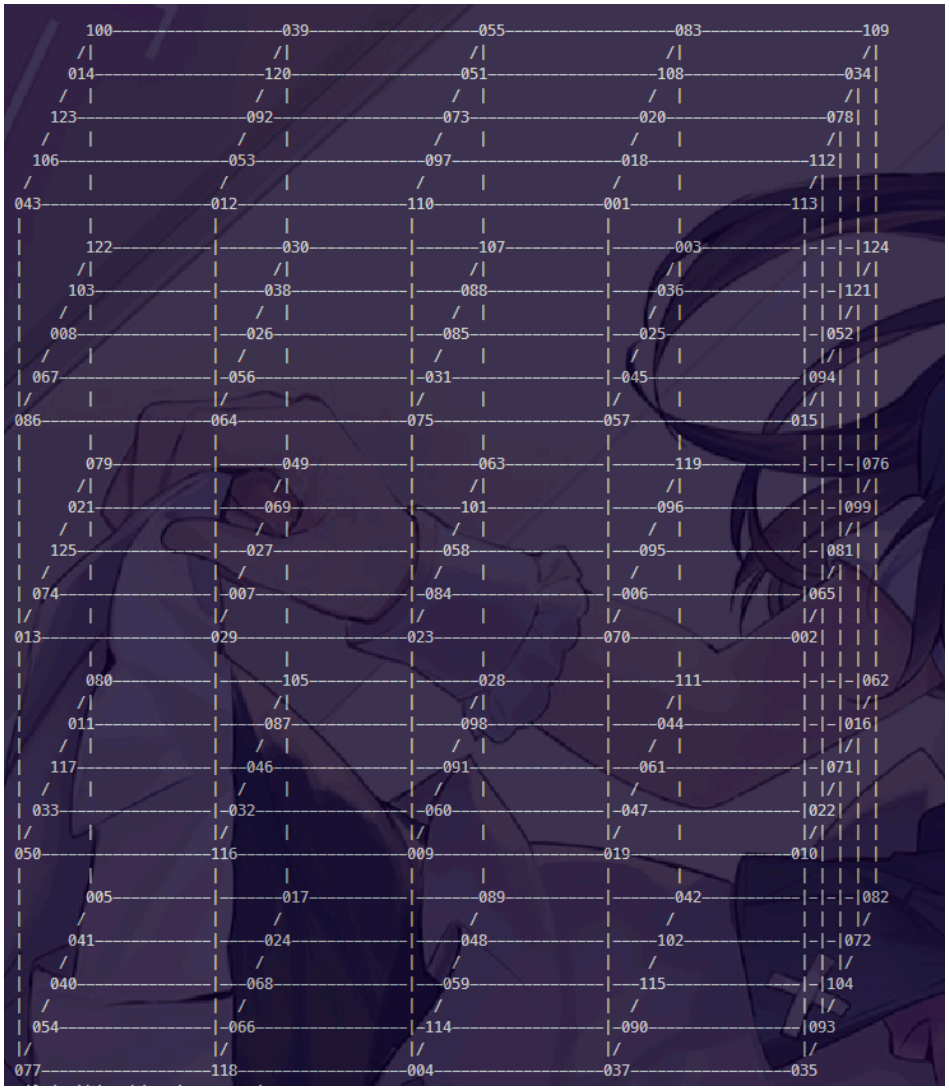
State Awal	

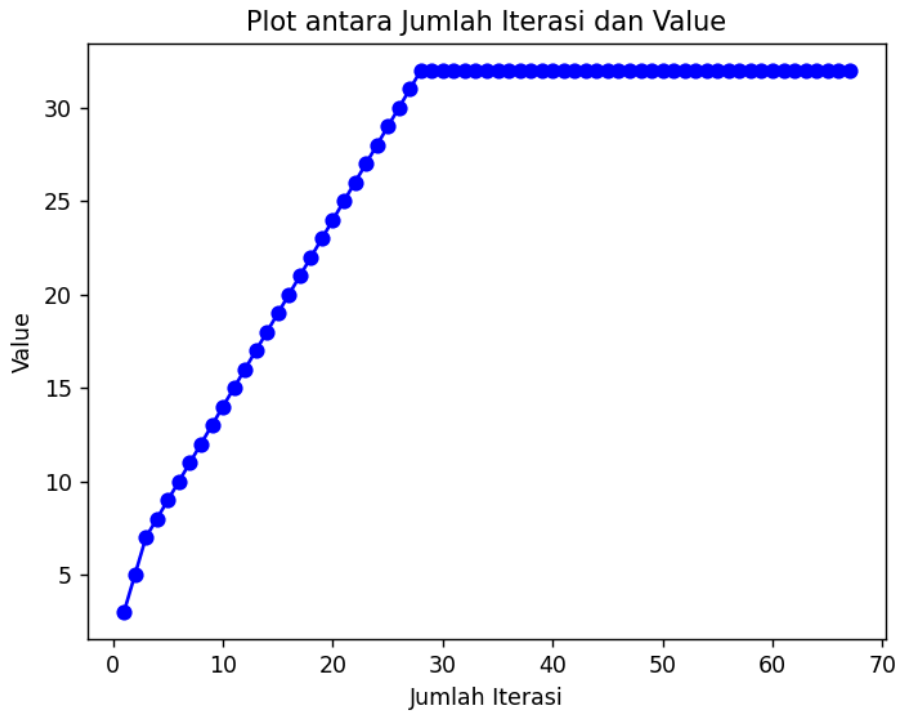
State Akhir	
Nilai Akhir Objective Function	39

Plot	
Durasi	37.846 detik
Banyak Iterasi	111
Parameter Maksimum	80

Percobaan 3

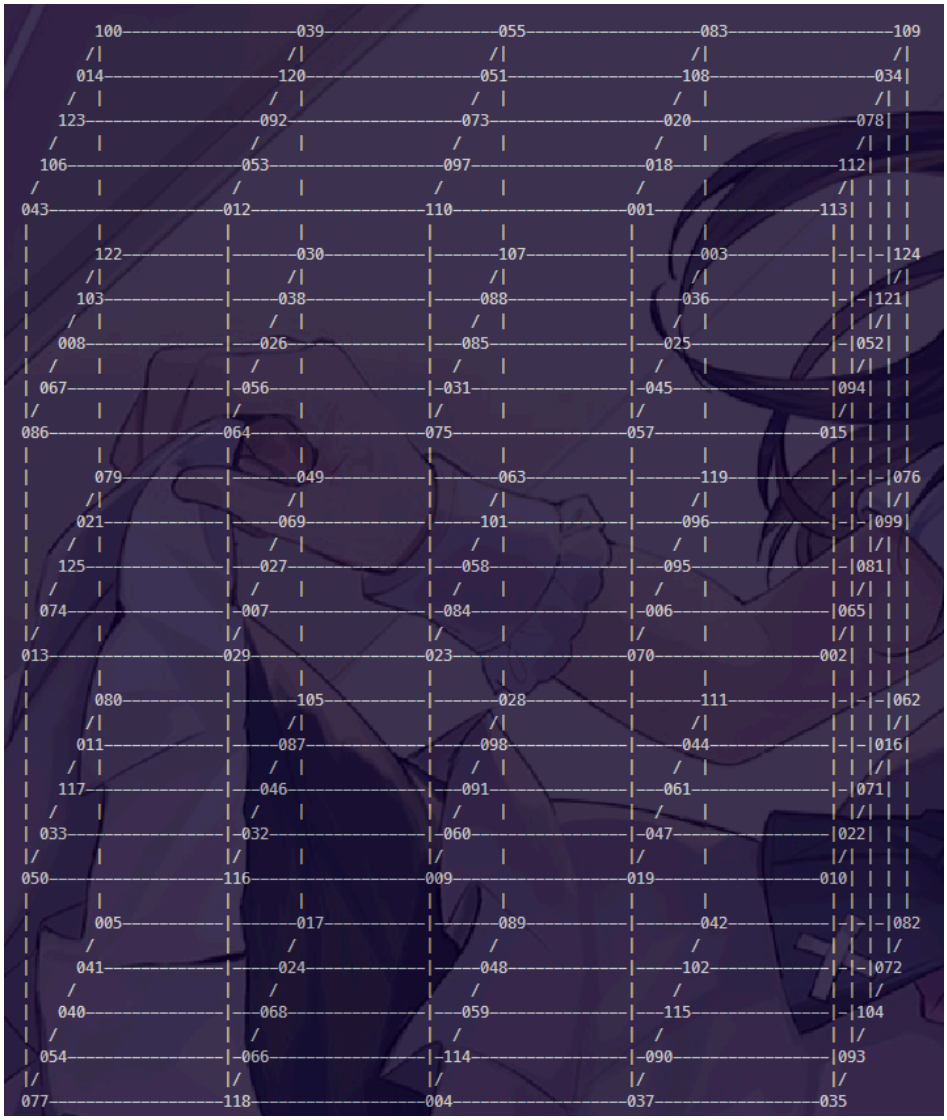
State Awal	

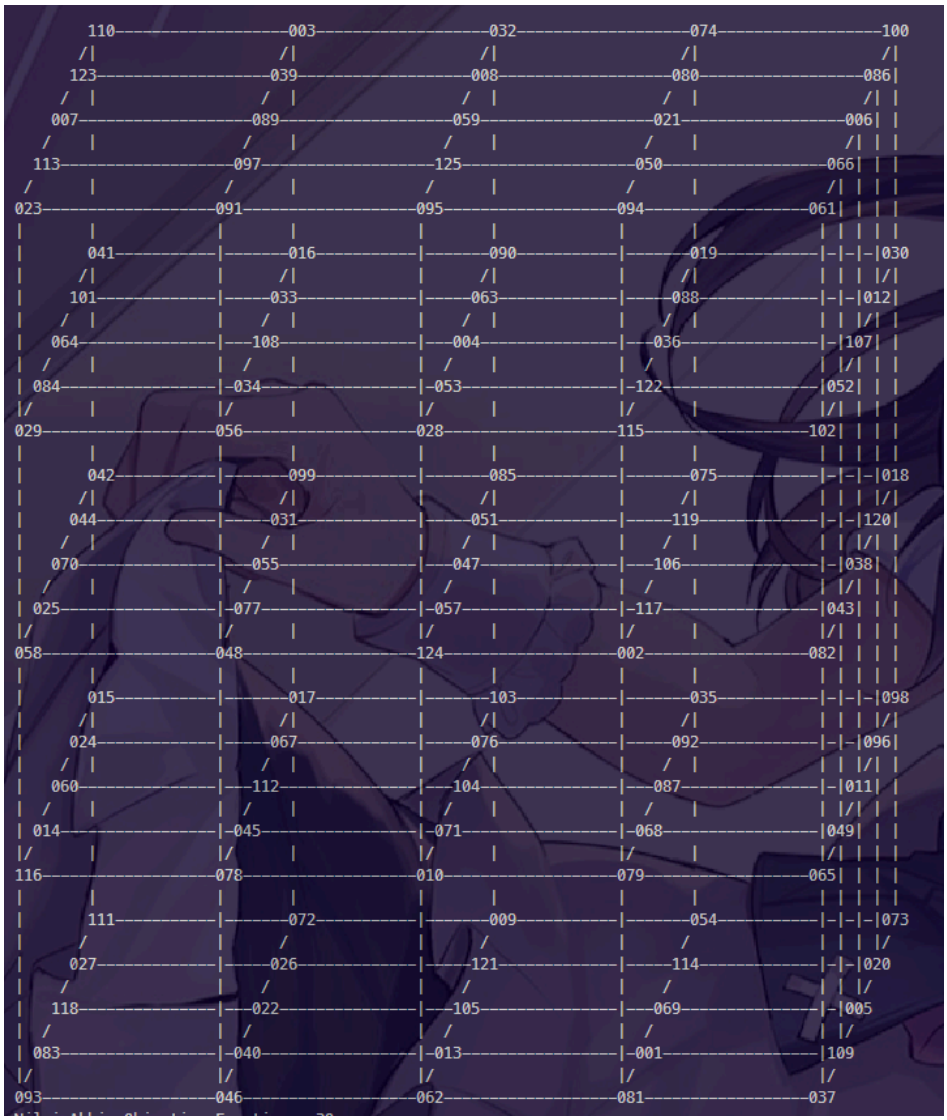
State Akhir	
Nilai Akhir Objective Function	32

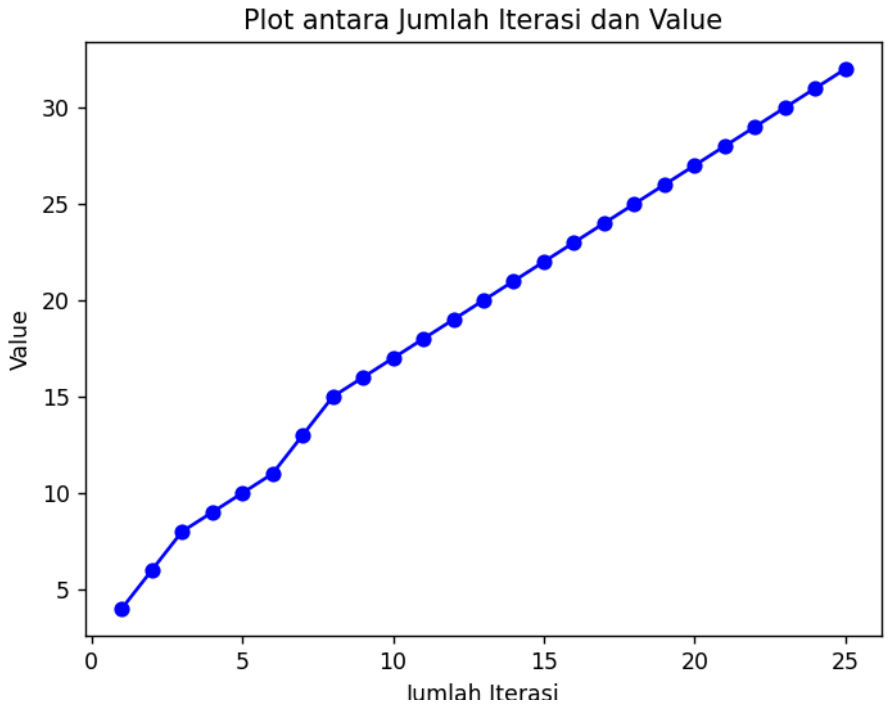
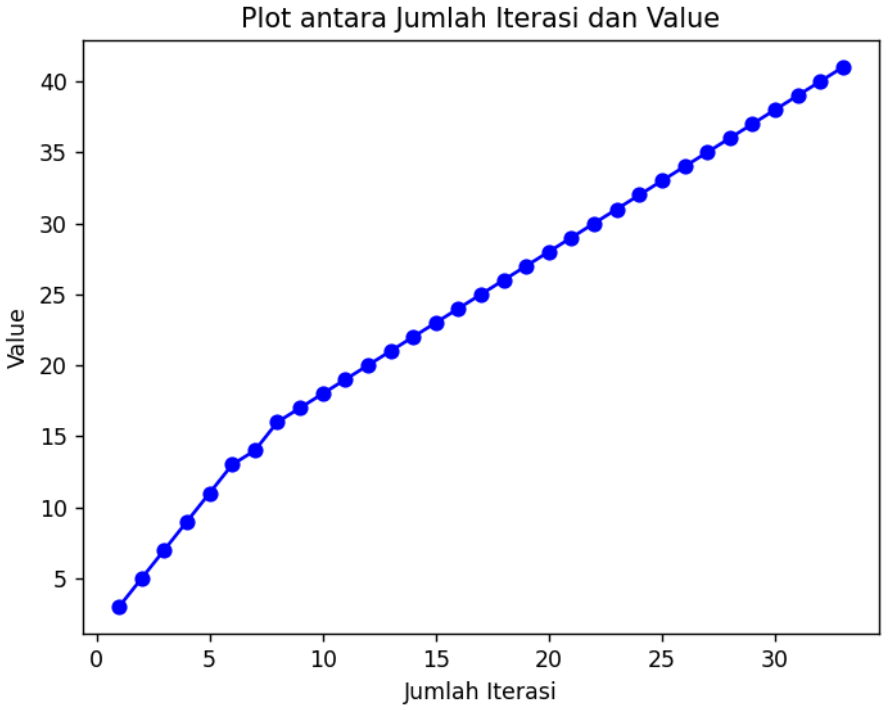
Plot	
Durasi	23.708 detik
Banyak Iterasi	67
Parameter Maksimum	40

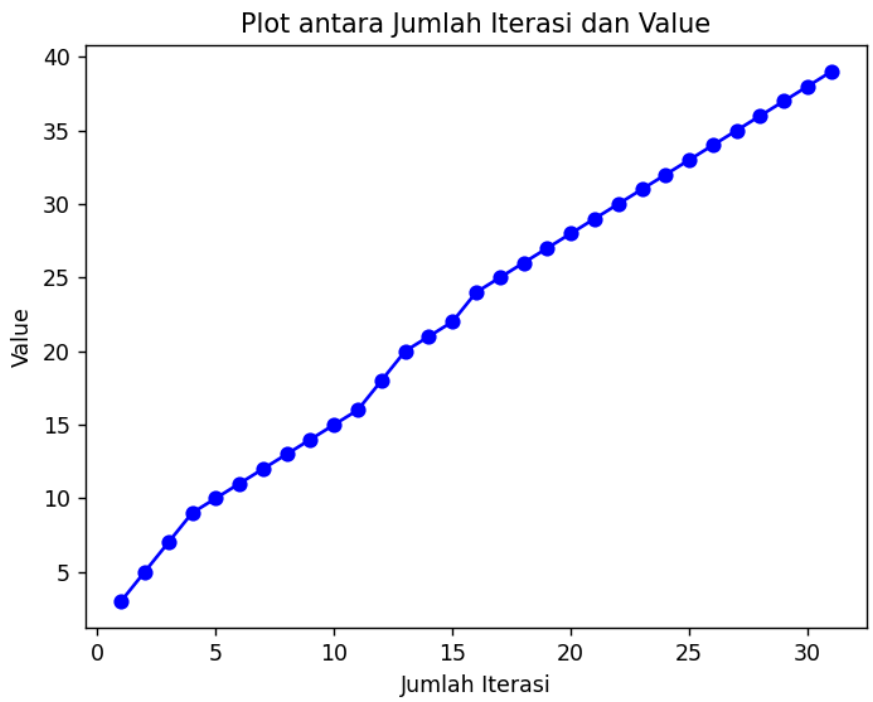
- Hill Climb : Random Restart

  Percobaan 1

State Awal	

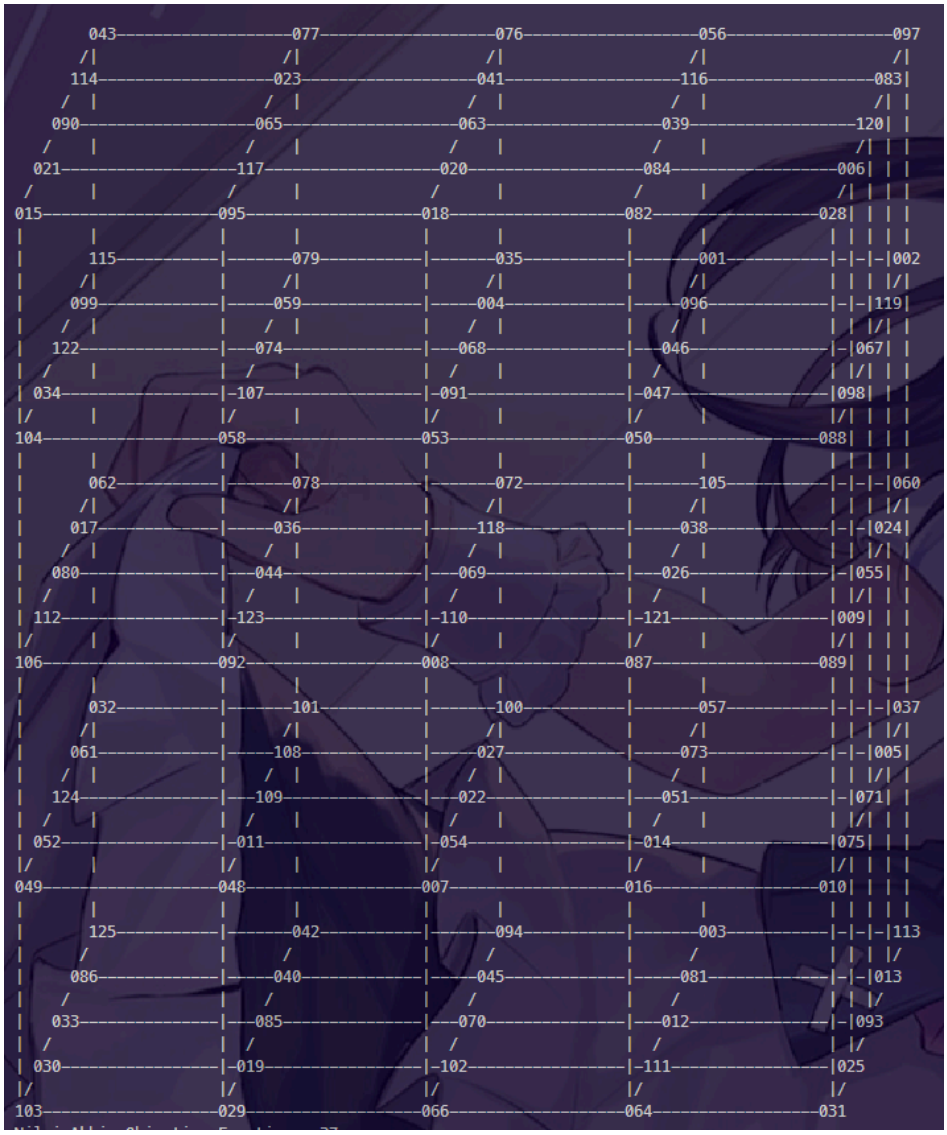
State Akhir	
Nilai Akhir Objective Function	39

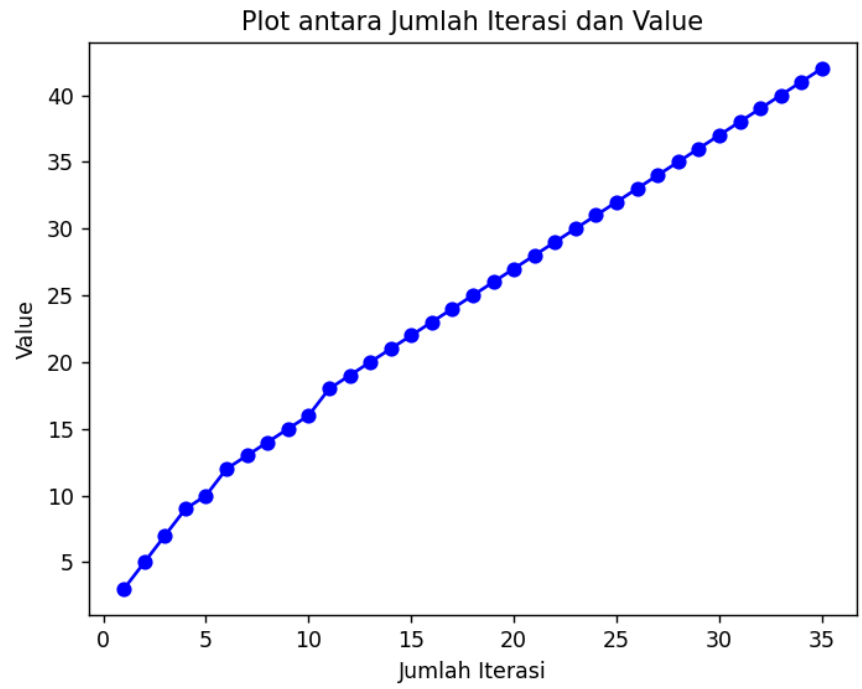
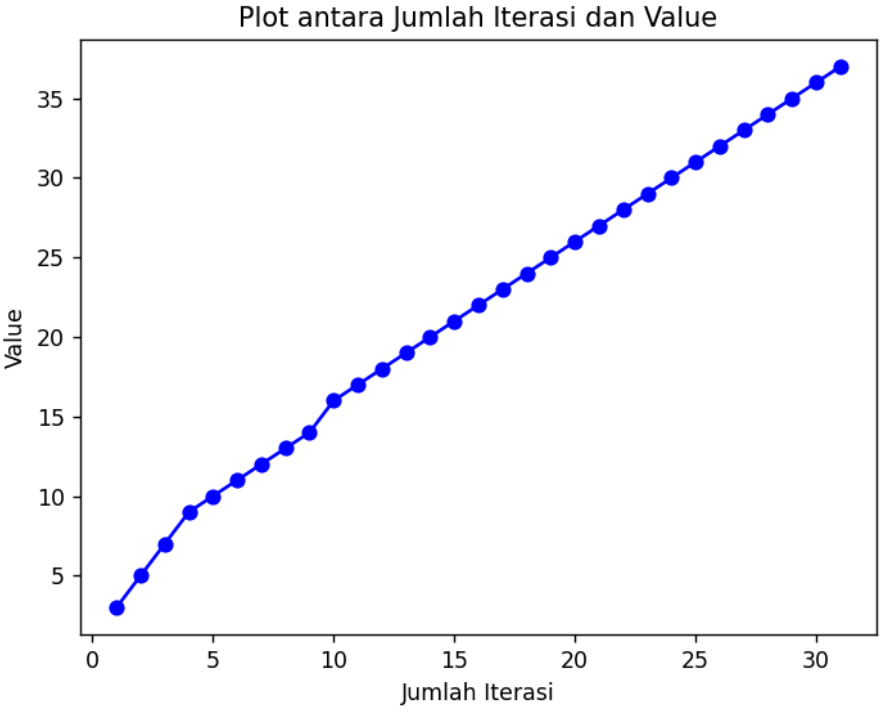
Plot	 Plot antara Jumlah Iterasi dan Value Value Jumlah Iterasi  Plot antara Jumlah Iterasi dan Value Value Jumlah Iterasi

	
Durasi	33.378 detik
Banyak Restart	3
Banyak Iterasi per Restart	25, 33, 31
Parameter Maksimum Restart	3

Percobaan 2

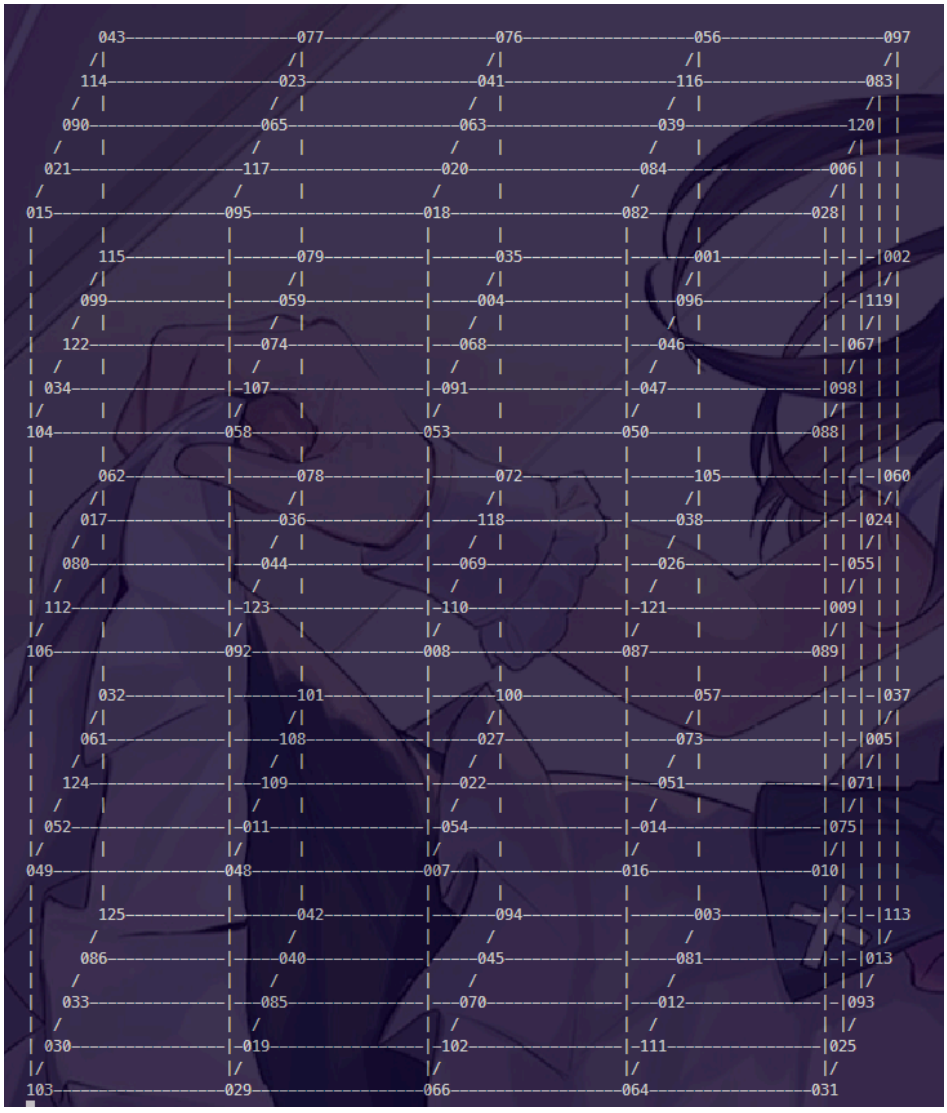
State Awal	

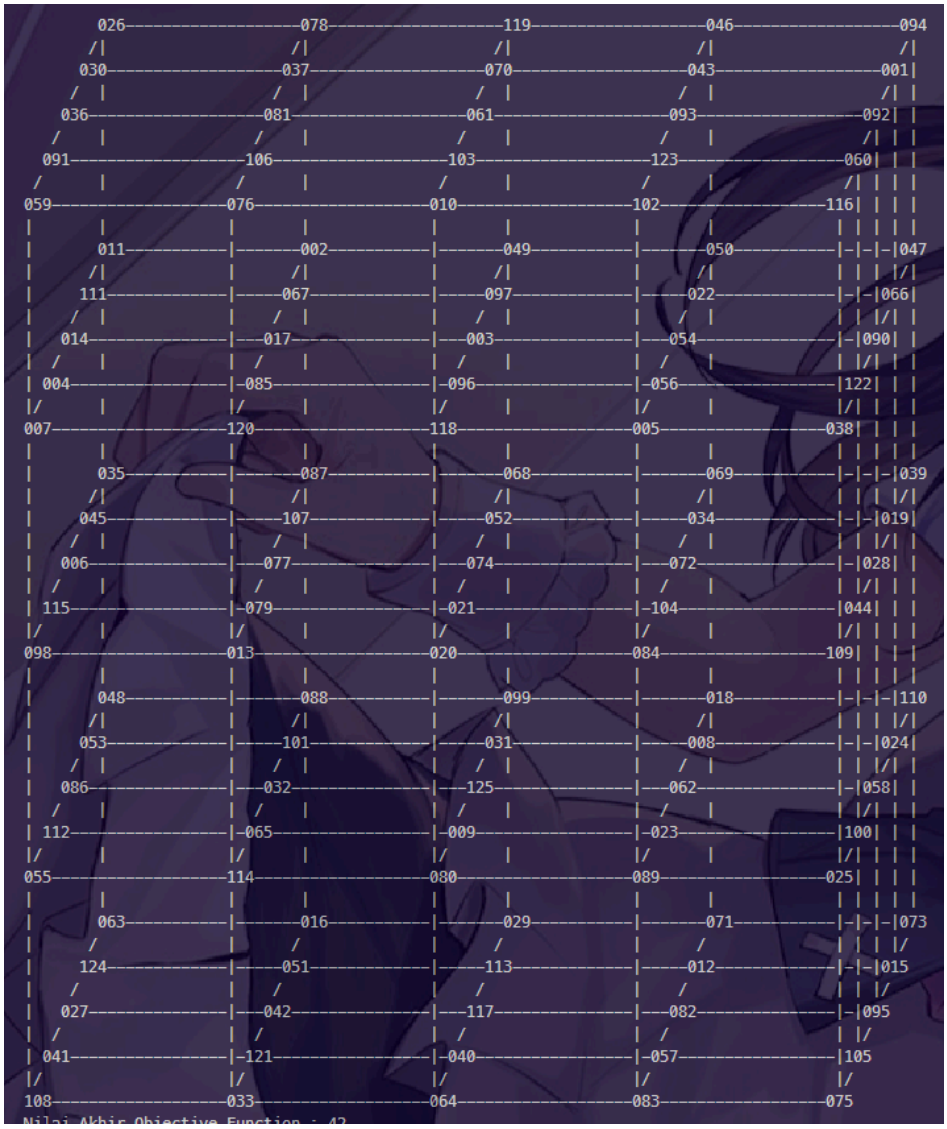
State Akhir	
Nilai Akhir Objective Function	37

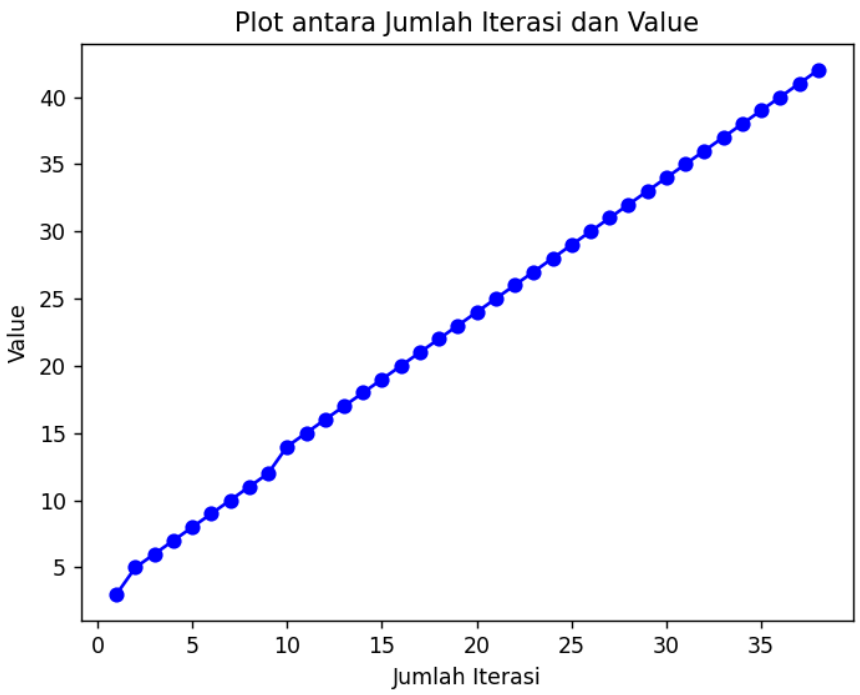
Plot	 
Durasi	24.52 detik
Banyak Restart	2

Banyak Iterasi per Restart	35, 31
Parameter Maksimum Restart	2

Percobaan 3

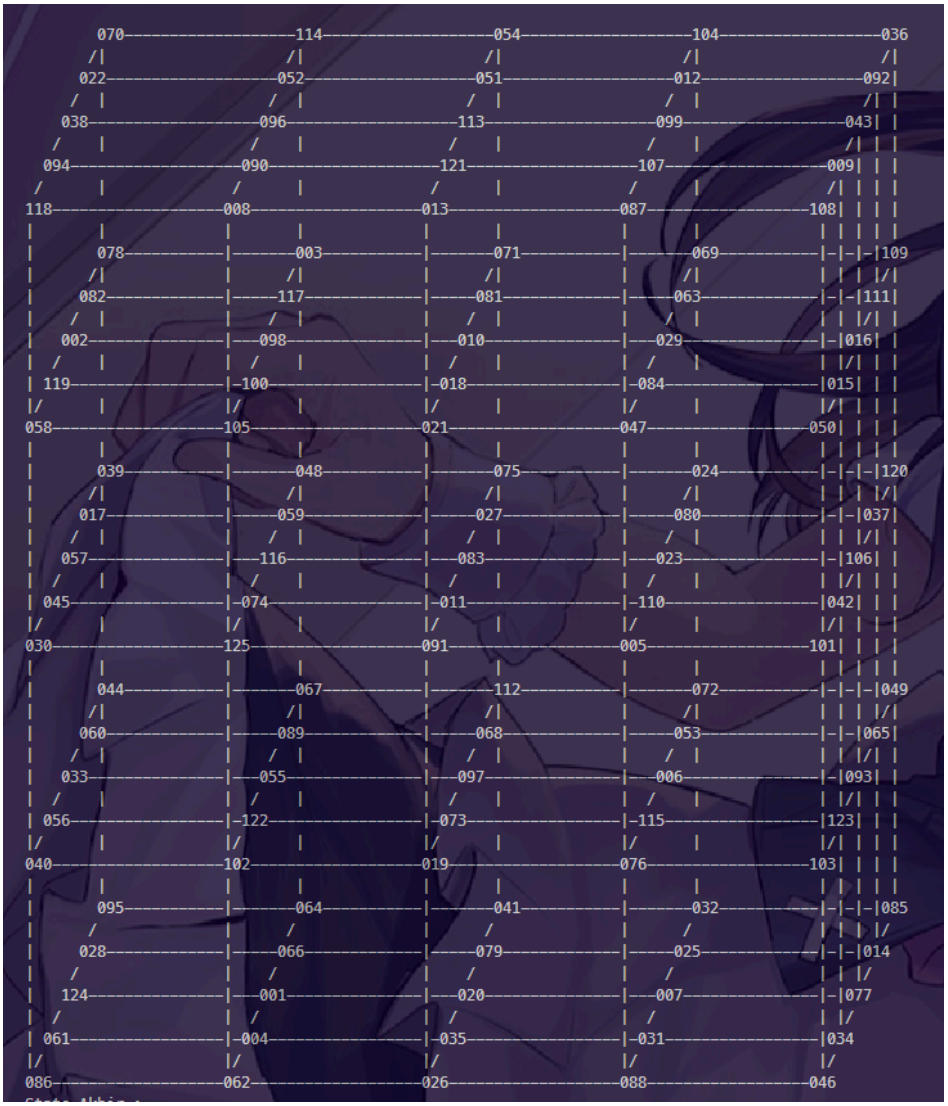
State Awal	

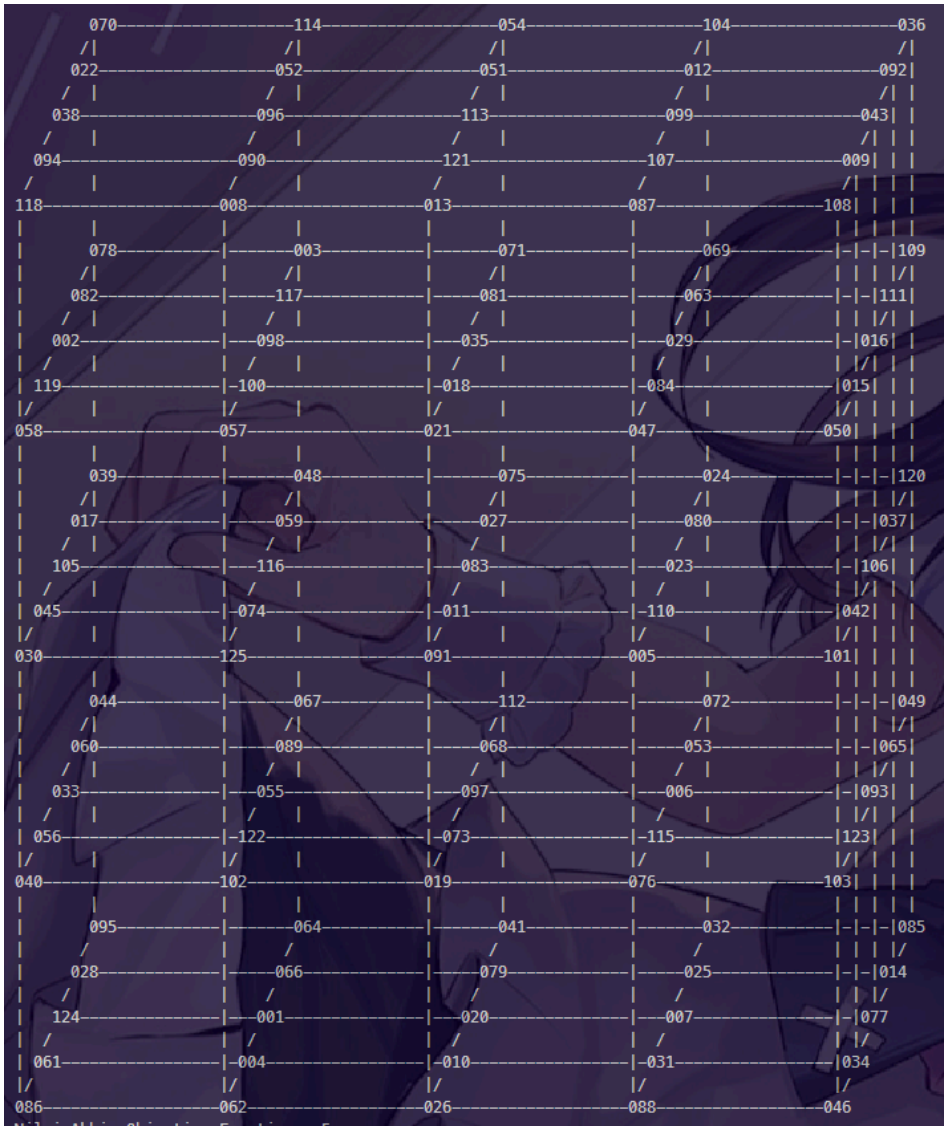
State Akhir	
Nilai Akhir Objective Function	42

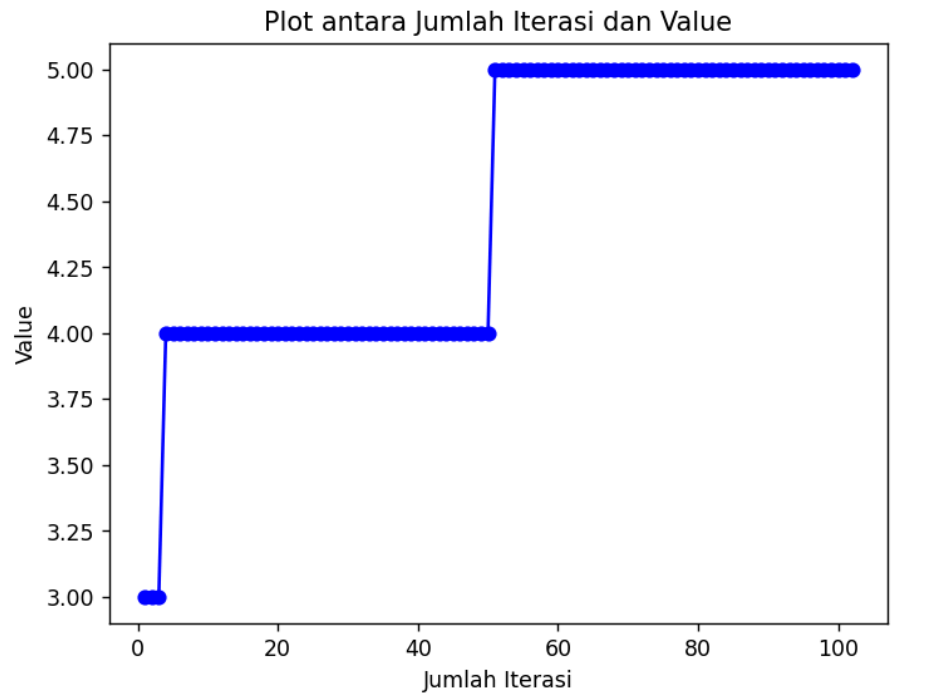
Plot	
Durasi	15.044 detik
Banyak Restart	1
Banyak Iterasi per Restart	38
Parameter Maksimum Restart	1

- Hill Climb : Stochastic

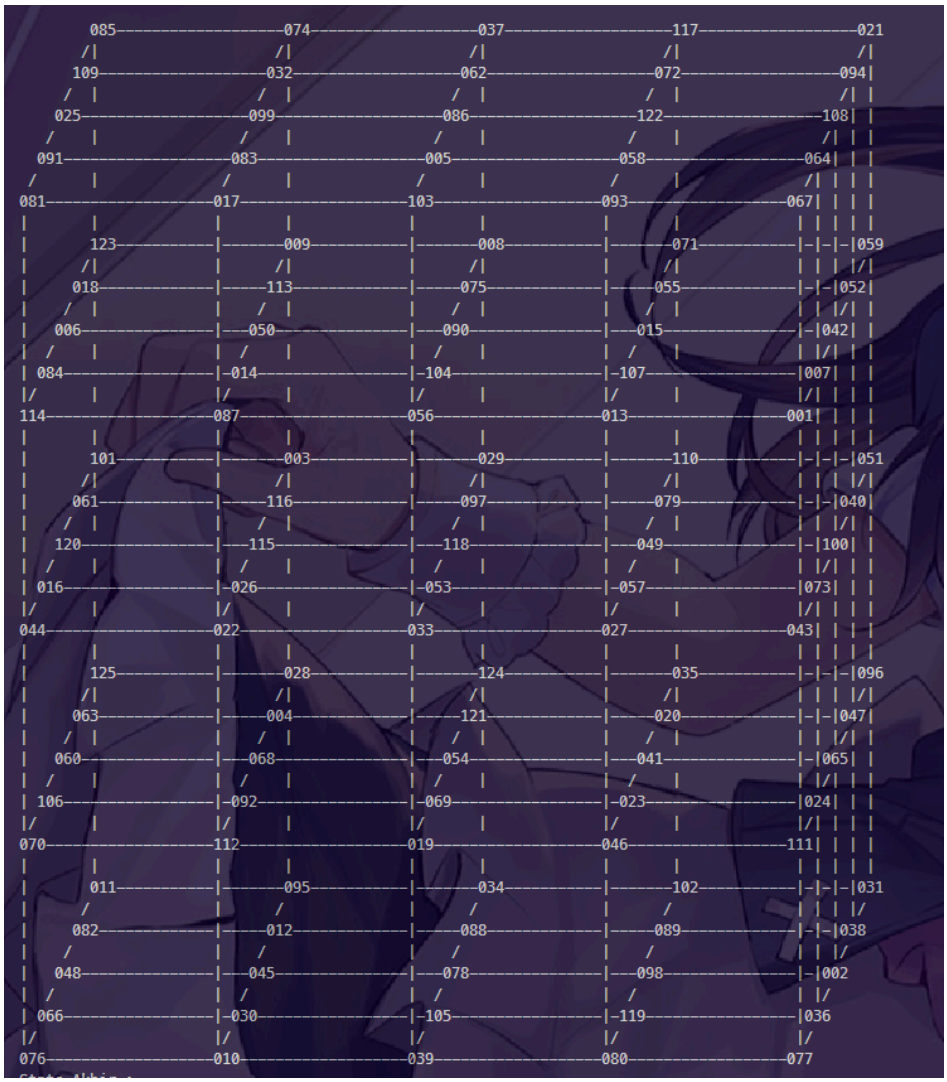
  Percobaan 1

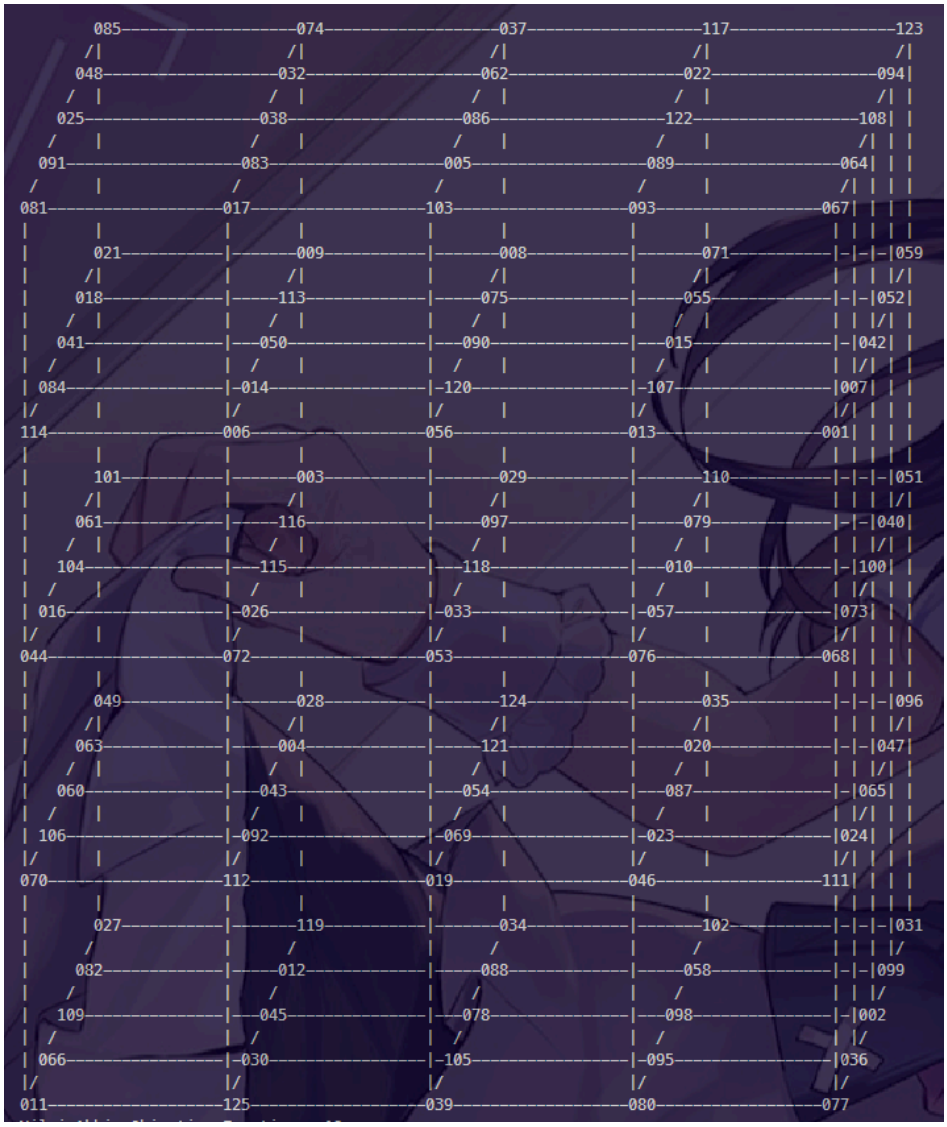
State Awal	

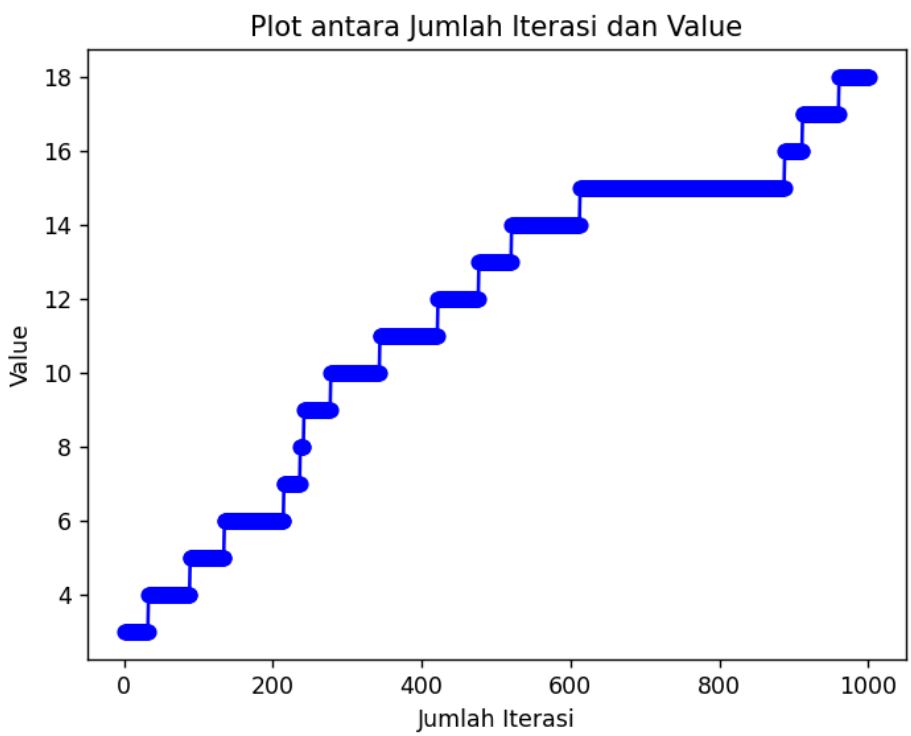
State Akhir	
Nilai Akhir Objective Function	5

Plot	
Durasi	0.002 detik
Banyak Iterasi	100

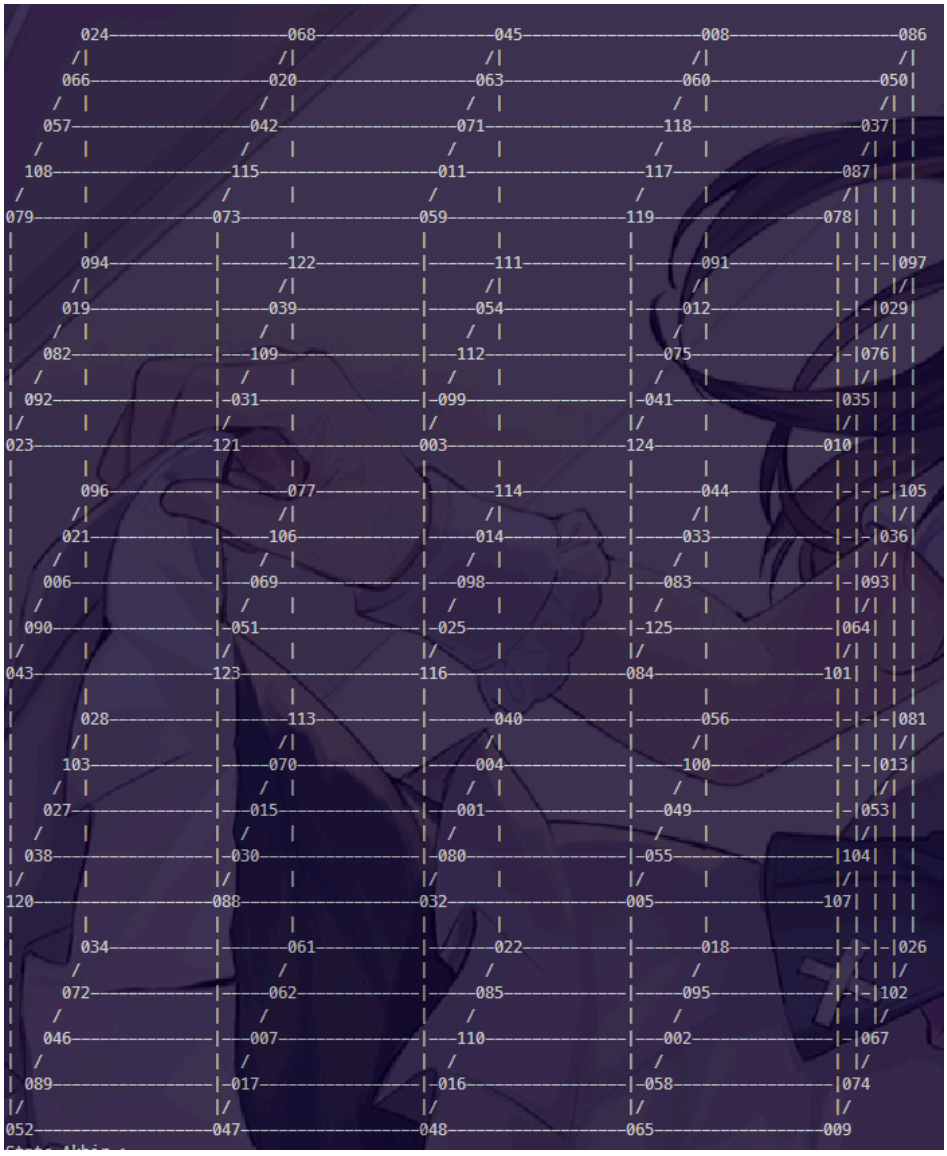
Percobaan 2

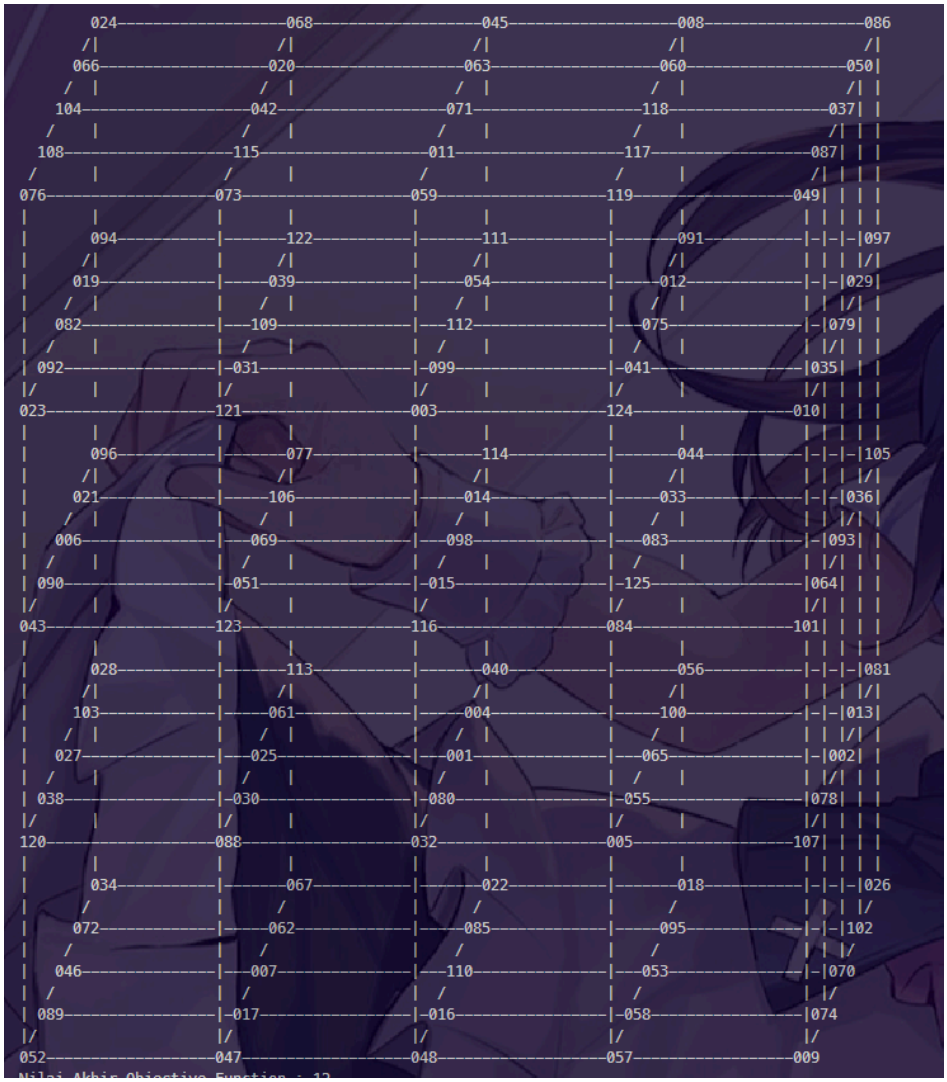
State Awal	

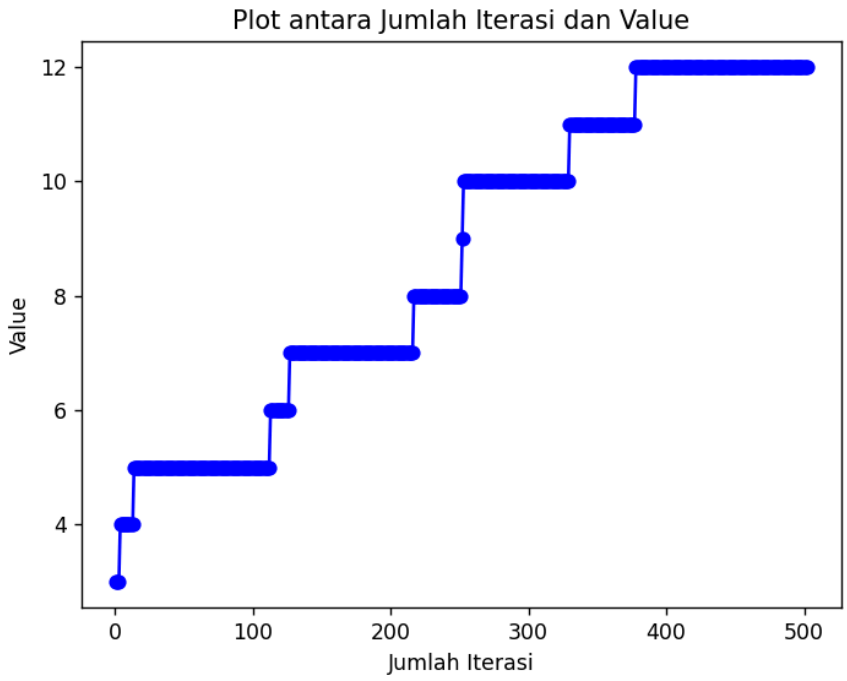
State Akhir	
Nilai Akhir Objective Function	18

Plot	
Durasi	0.029
Banyak Iterasi	1000

Percobaan 3

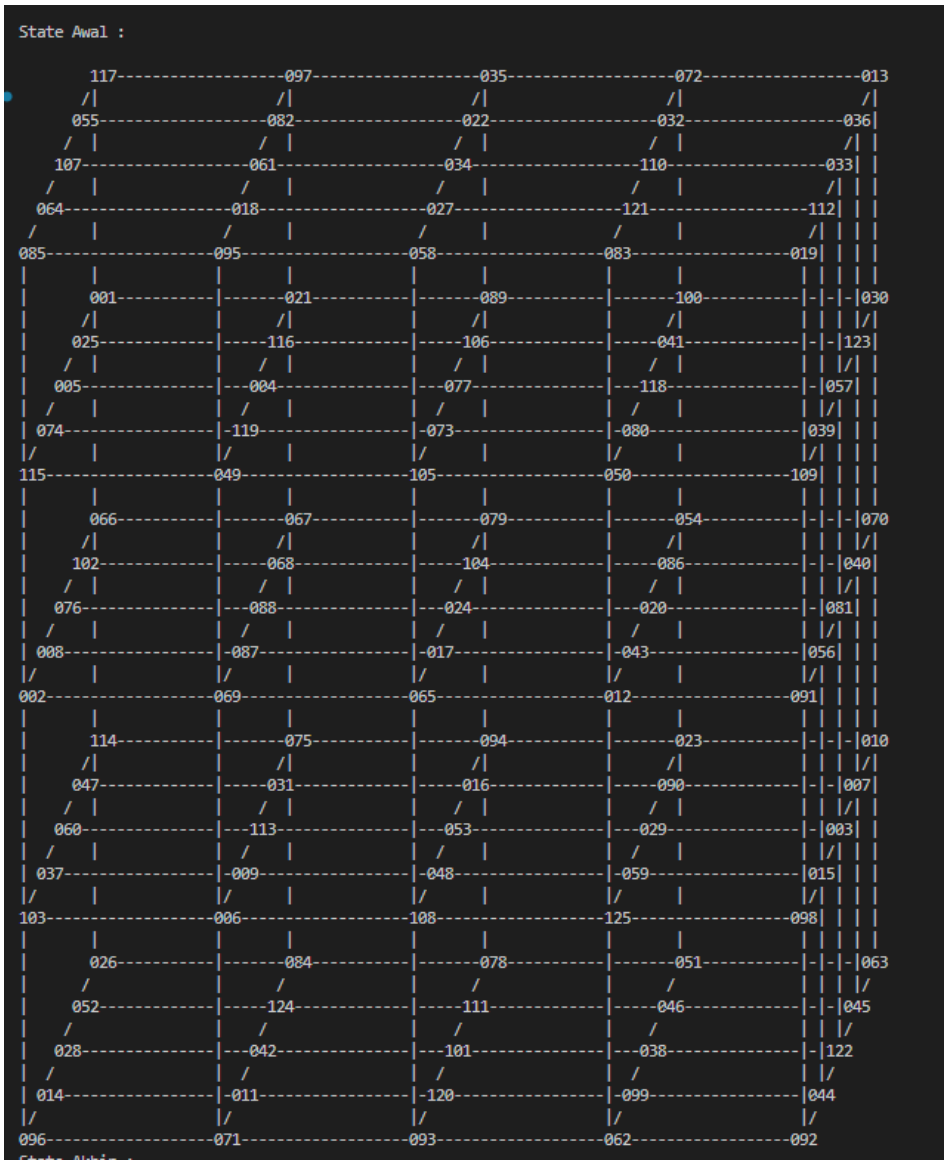
State Awal	

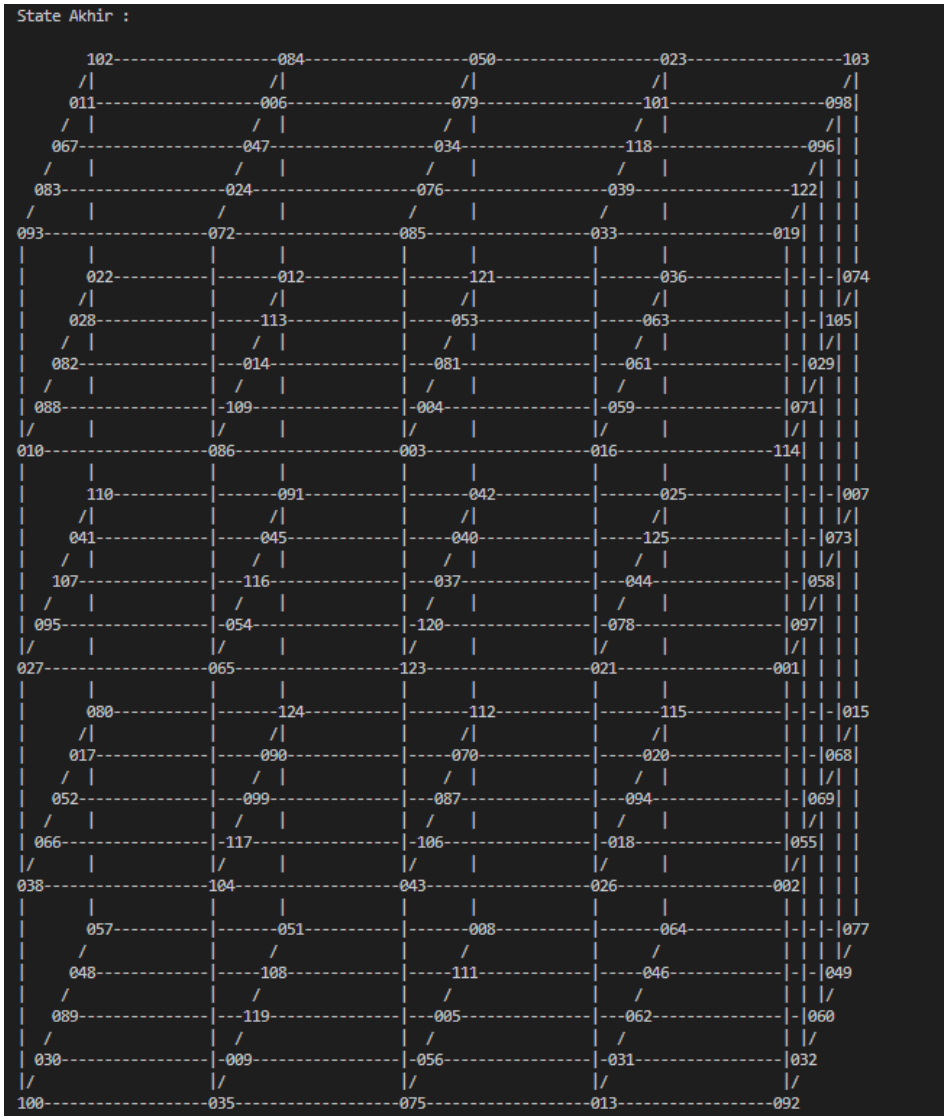
State Akhir	
Nilai Akhir Objective Function	12

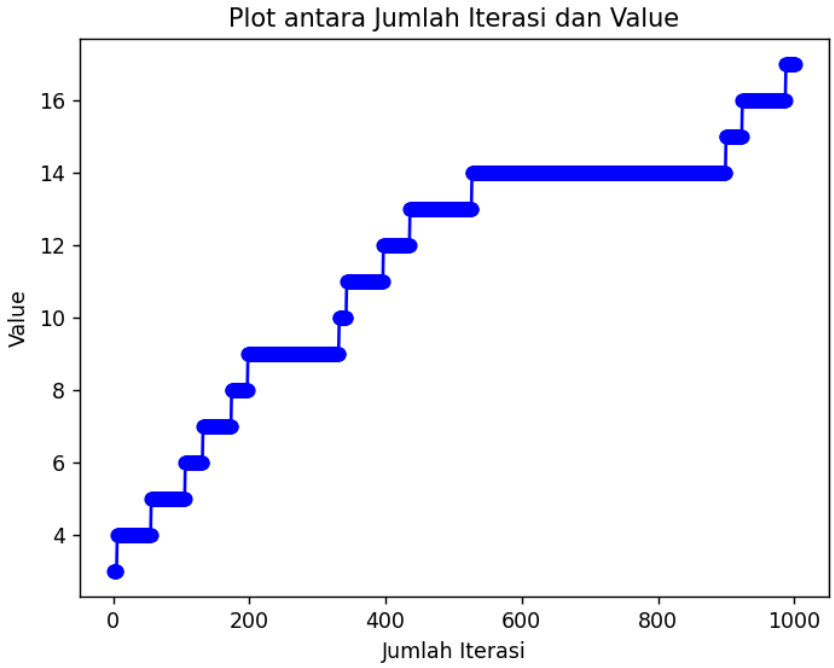
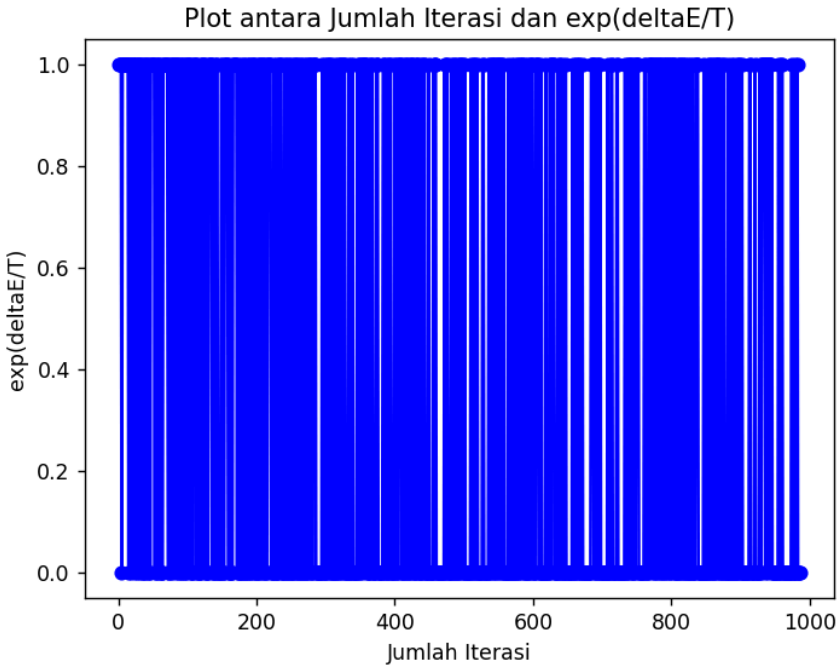
Plot	
Durasi	0.014 detik
Banyak Iterasi	500

- Simulated Annealing

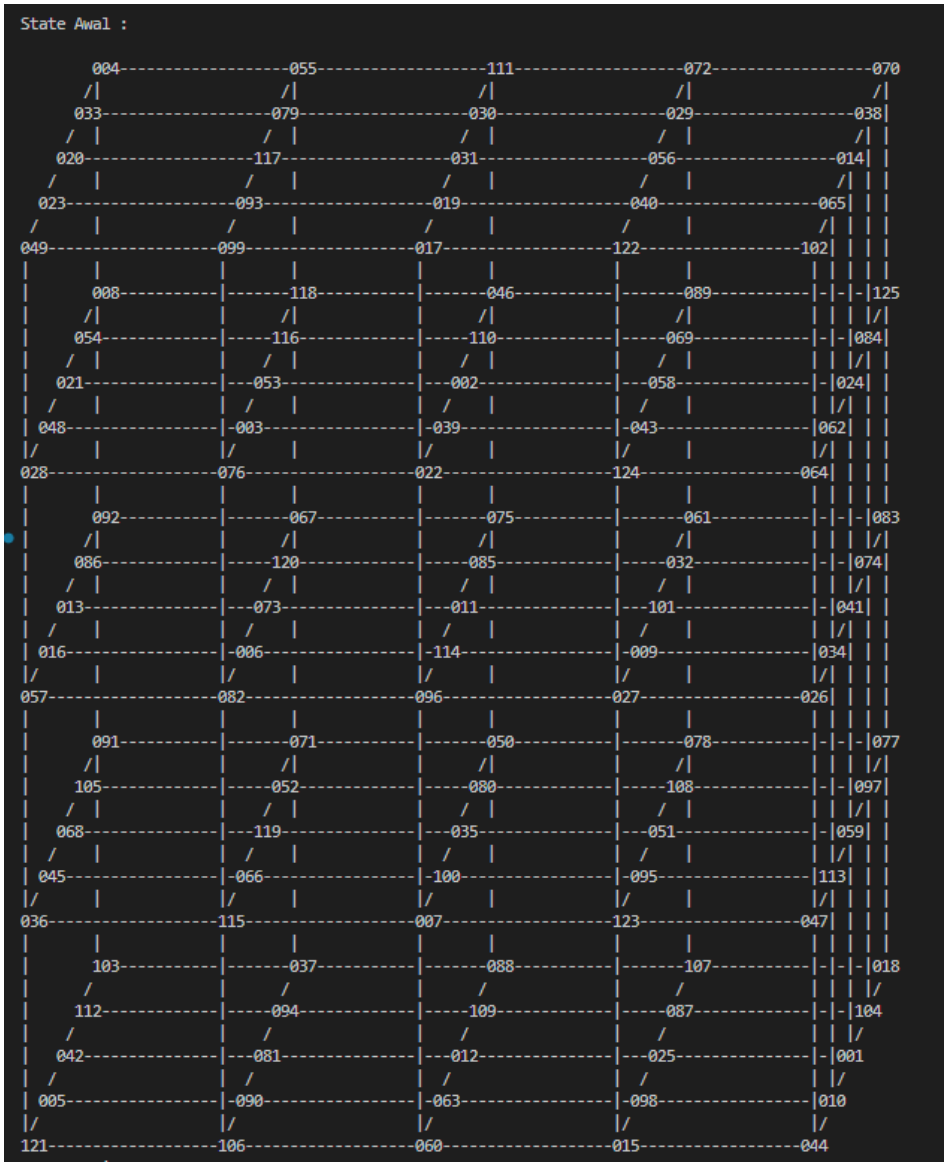
  Percobaan 1

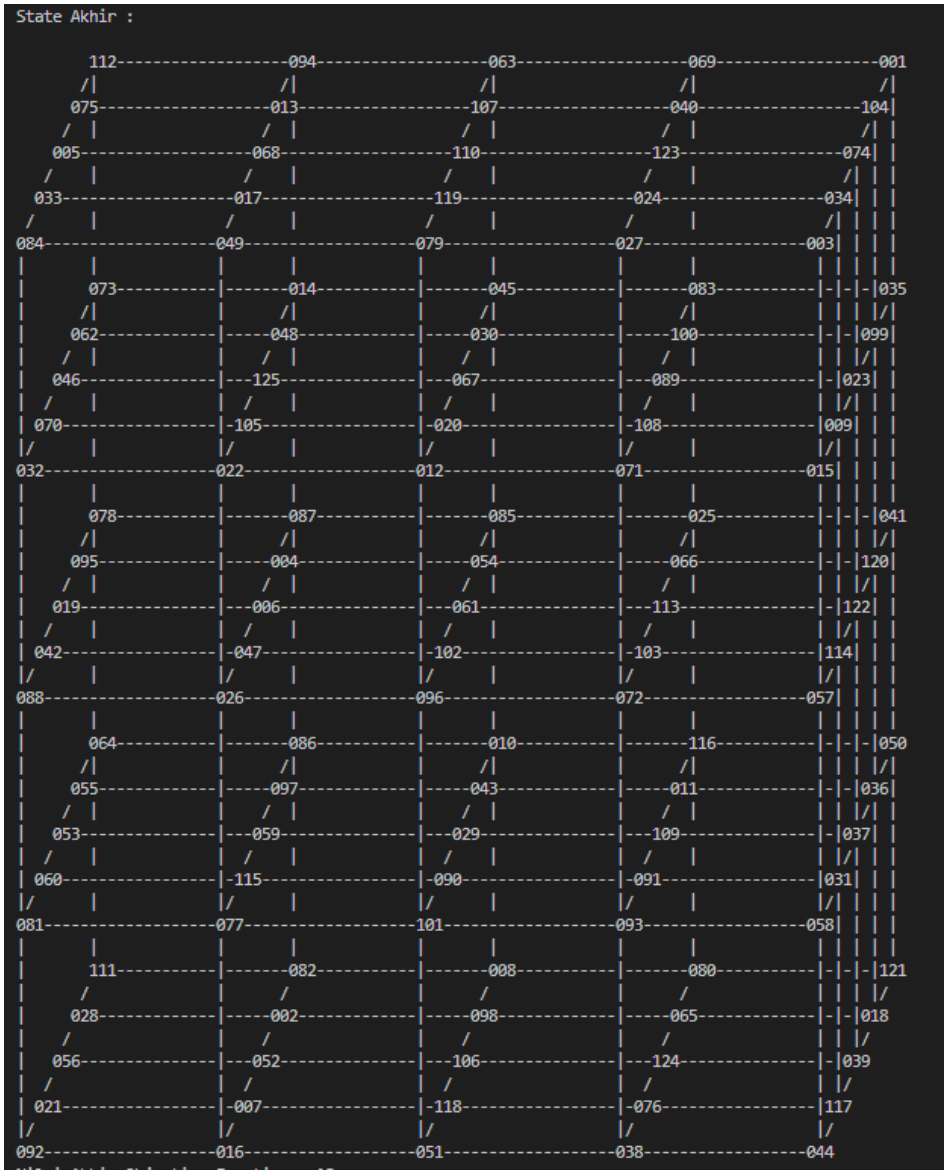
State Awal	

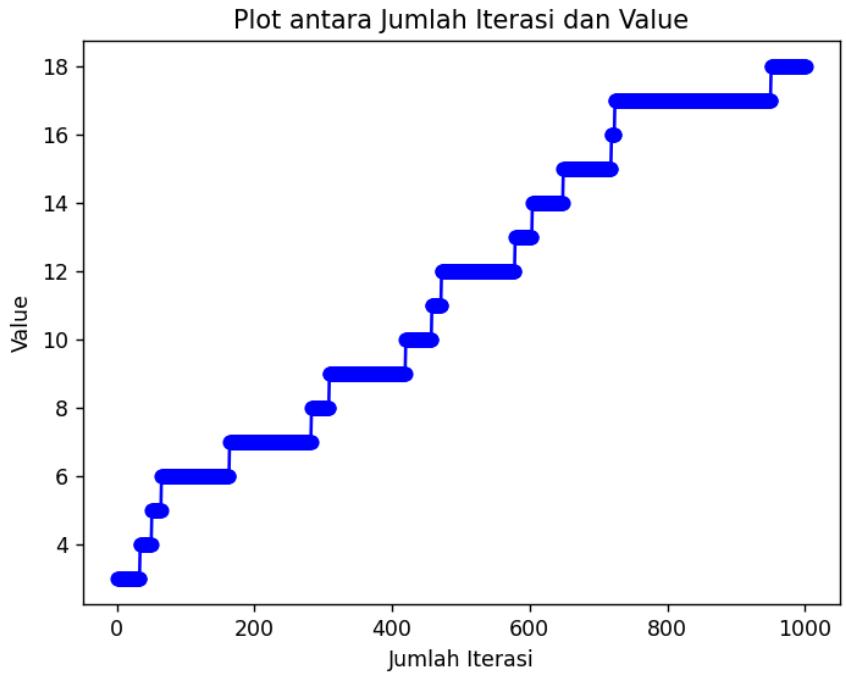
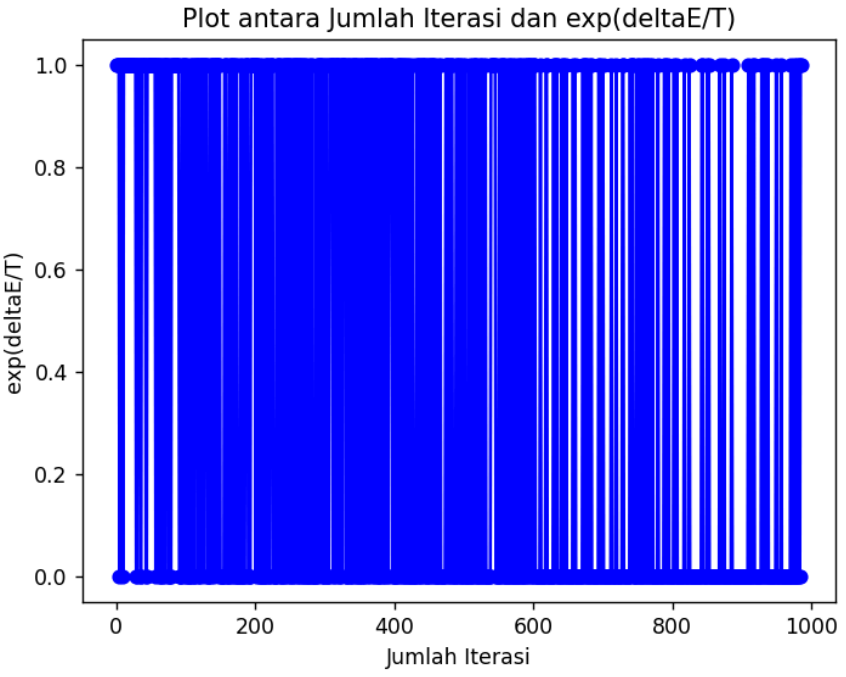
State Akhir	
Nilai Akhir Objective Function	17
Durasi	0.051 detik
Frekuensi stuck di lokal optima	986

Plot (objective function)	
Plot ($e^{\frac{\Delta E}{T}}$)	

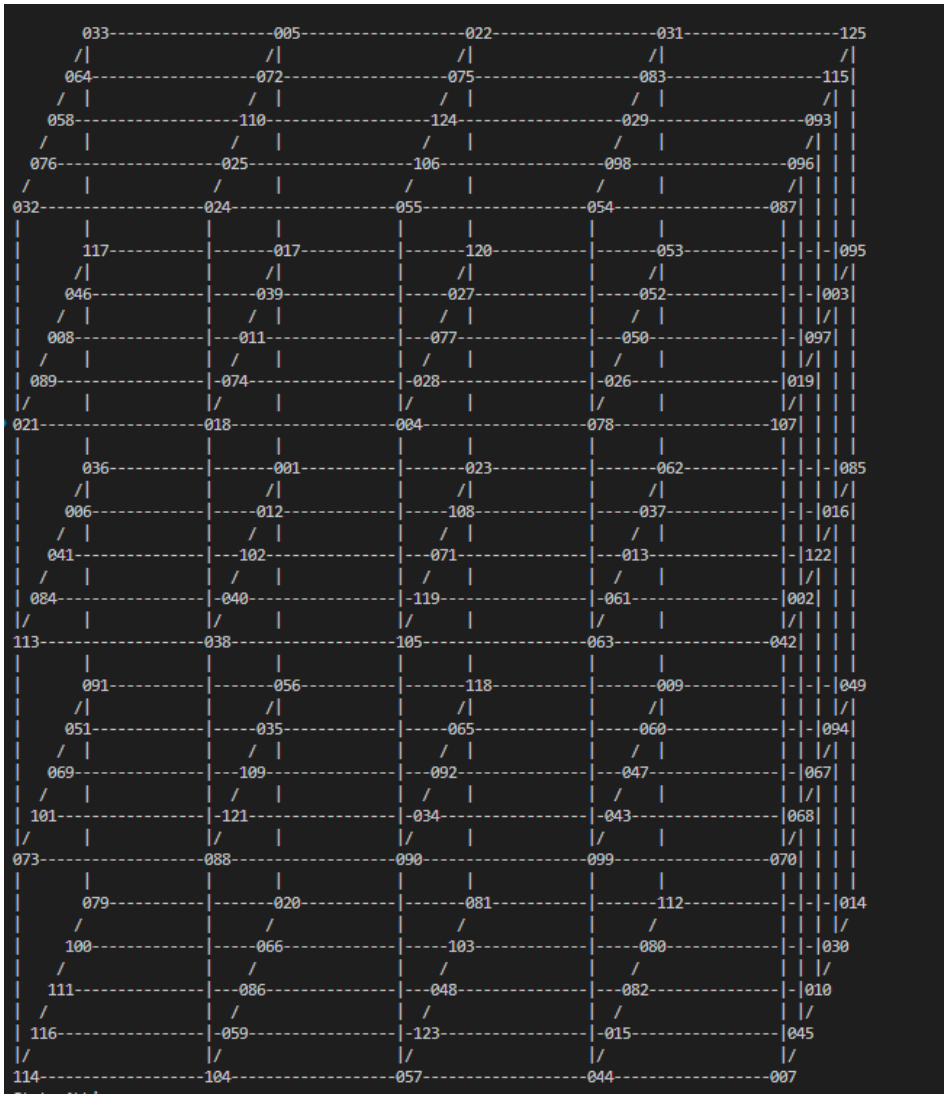
Percobaan 2

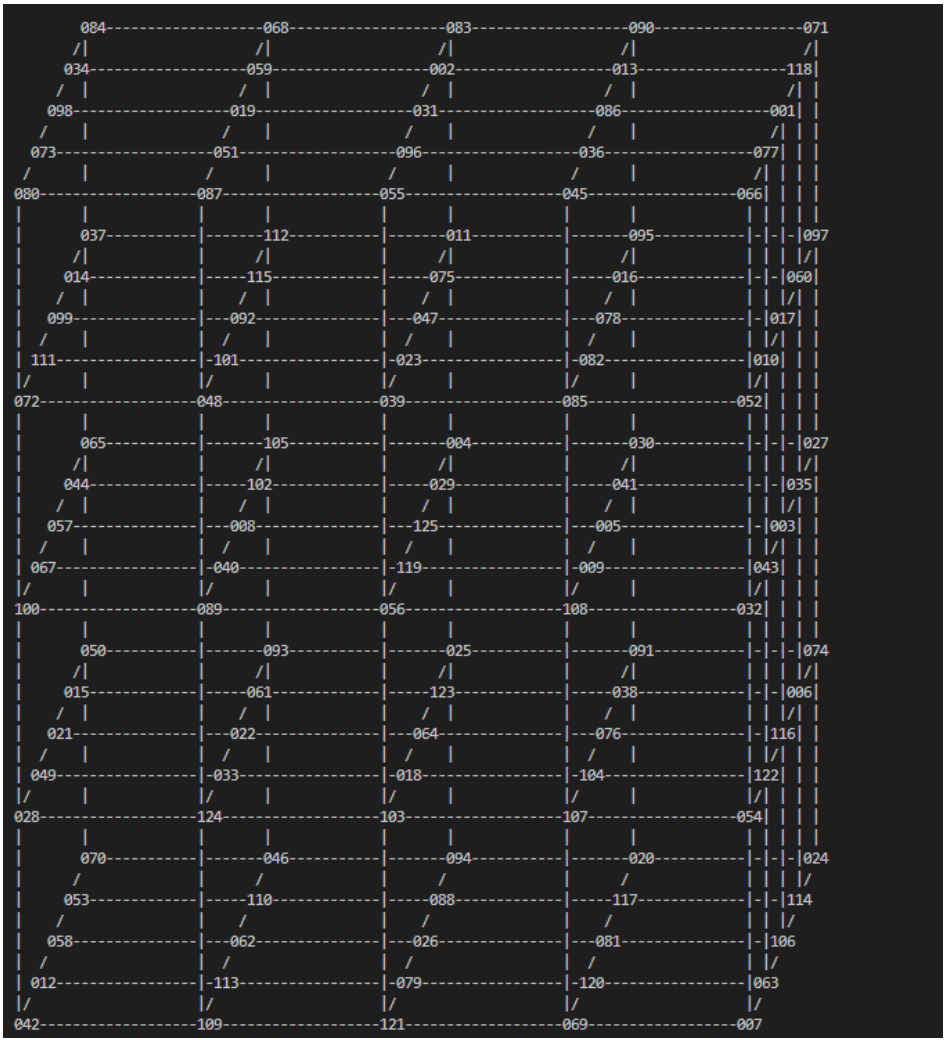
State Awal	

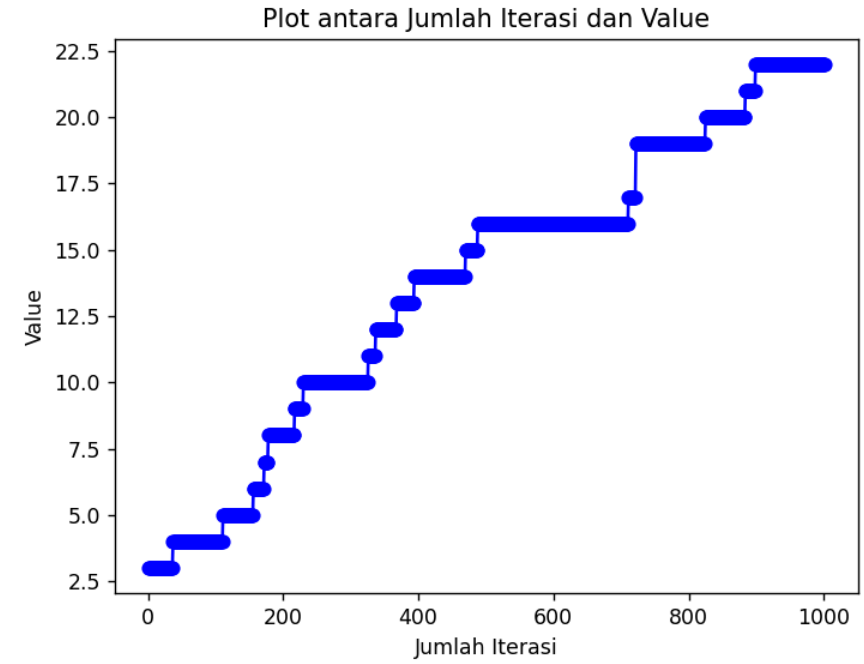
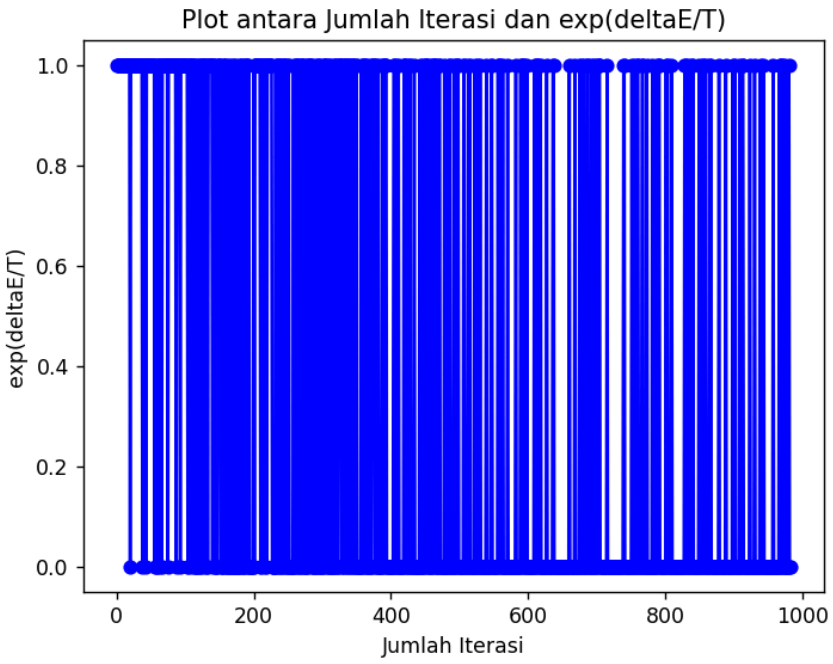
State Akhir	
Nilai Akhir Objective Function	18
Durasi	0.052 detik
Frekuensi stuck di lokal optima	985

Plot (objective function)	
Plot ($e^{\frac{\Delta E}{T}}$)	

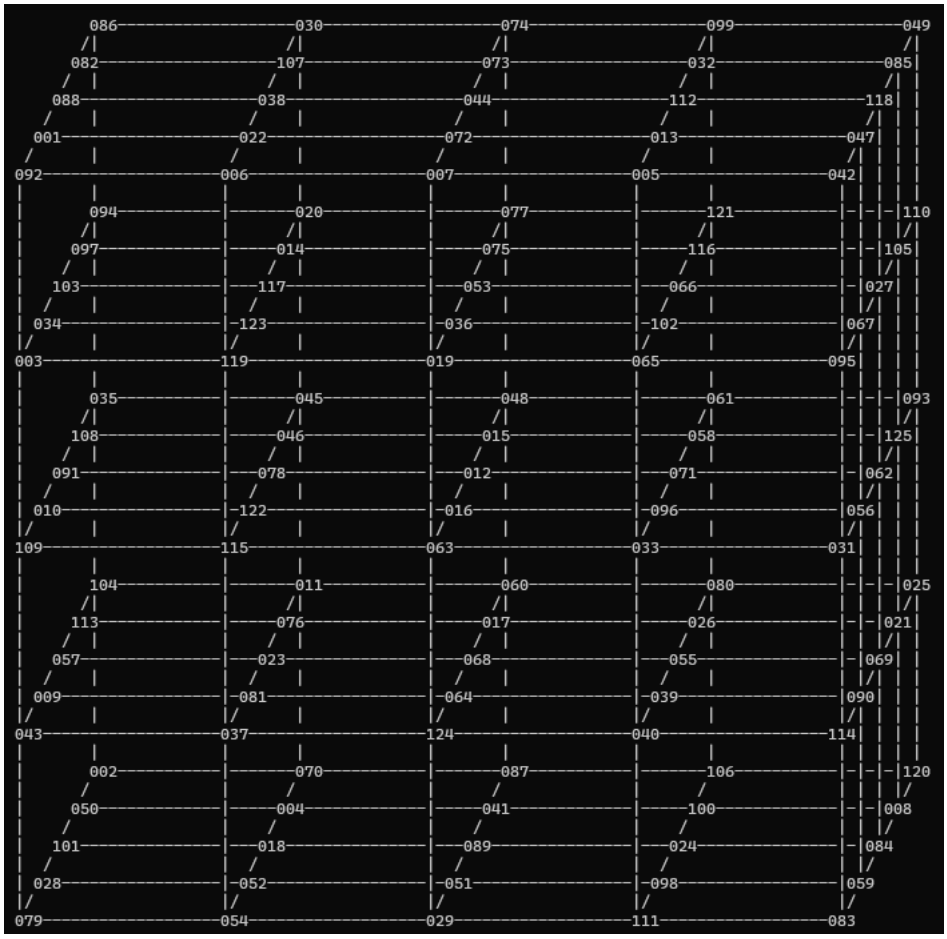
Percobaan 3

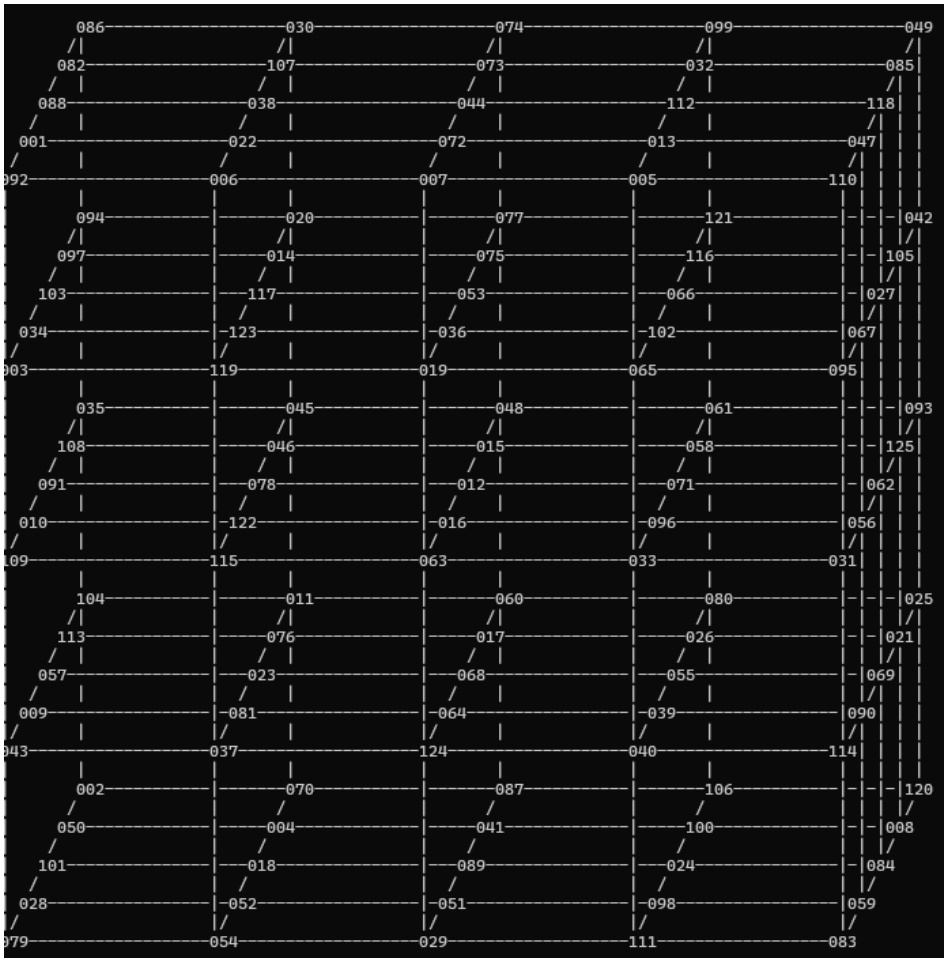
State Awal	

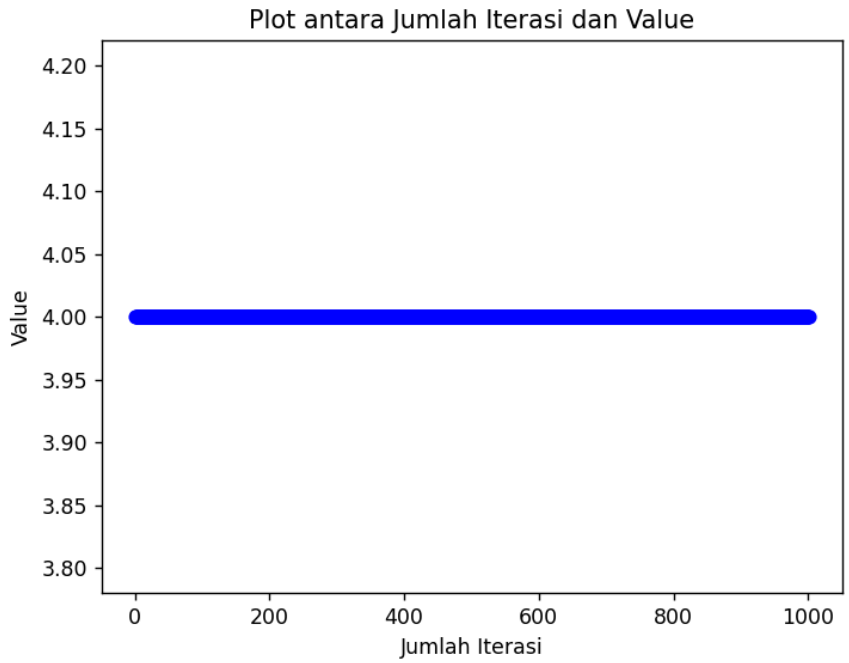
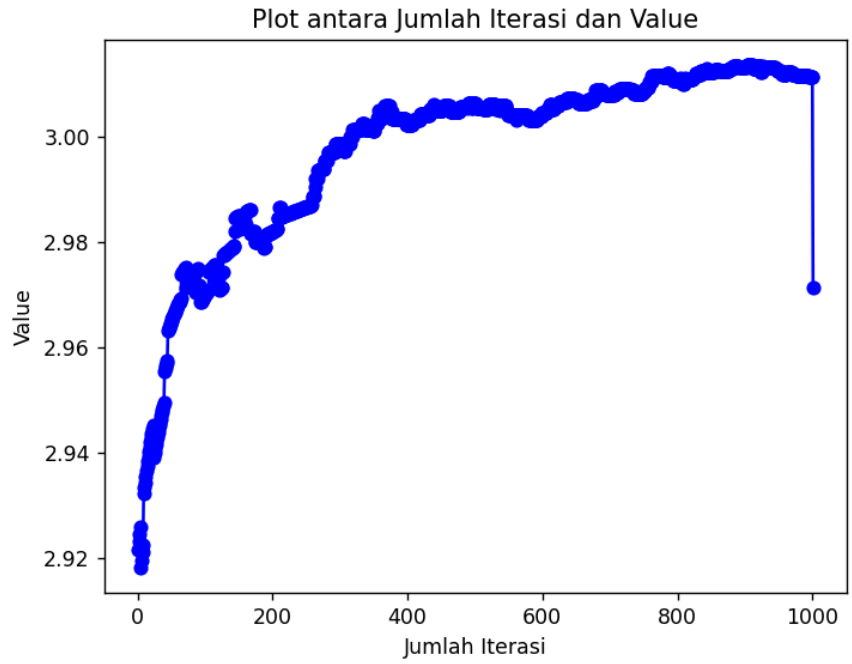
State Akhir	
Nilai Akhir Objective Function	22
Durasi	0.05 detik
Frekuensi stuck di lokal optima	982

Plot (objective function)	
Plot ($e^{\frac{\Delta E}{T}}$)	

- Genetic Algorithm
  Eksperimen 1-1, populasi sebagai kontrol

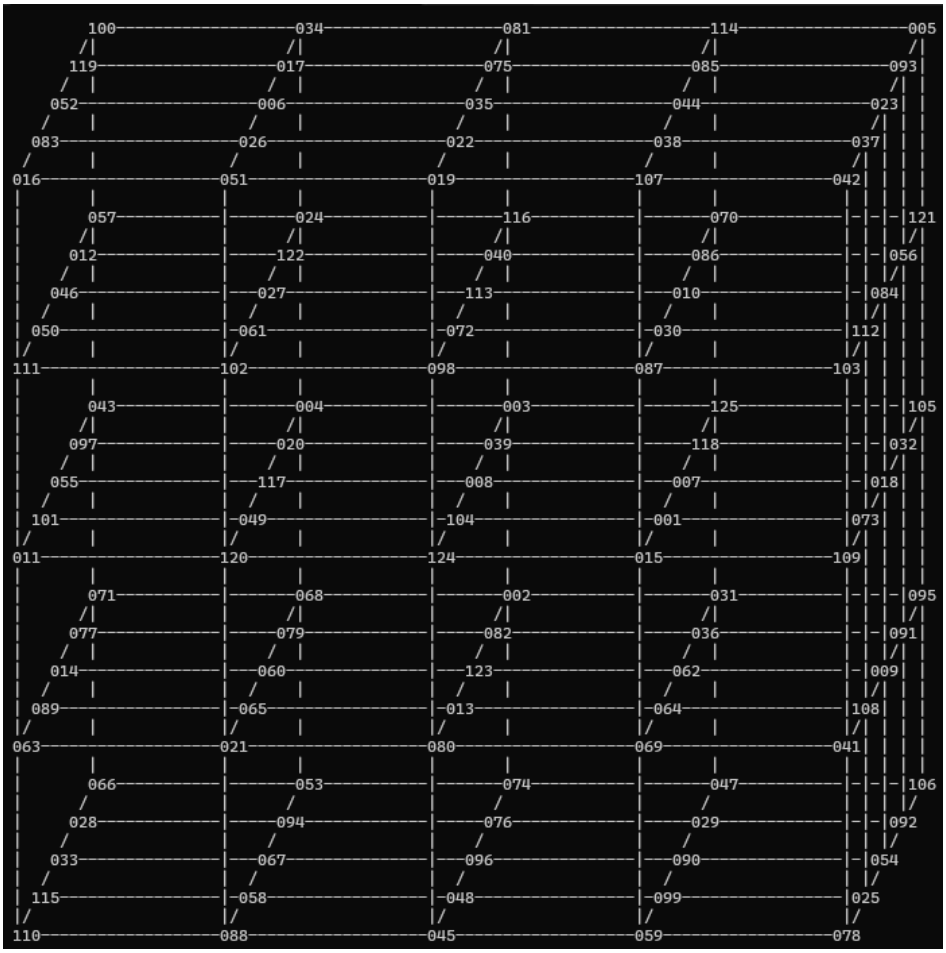
State Awal	

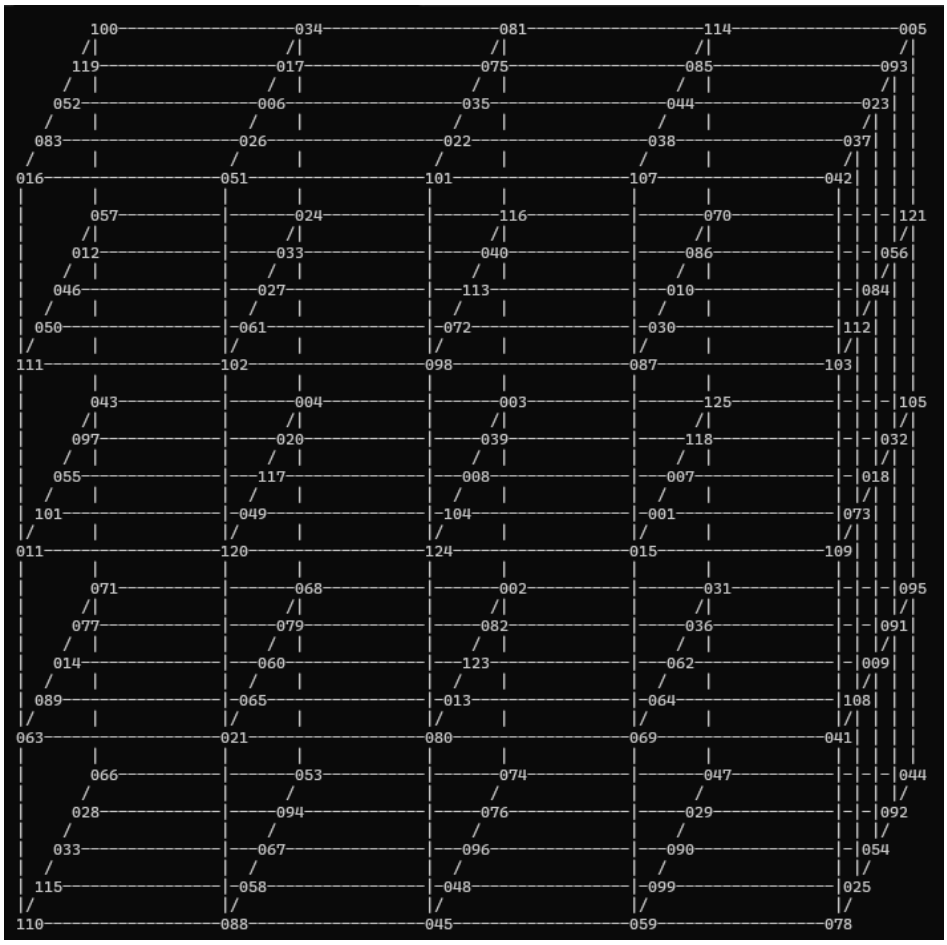
State Akhir	
Nilai Akhir Objective Function	4
Plot (objective function)	Nilai maksimal

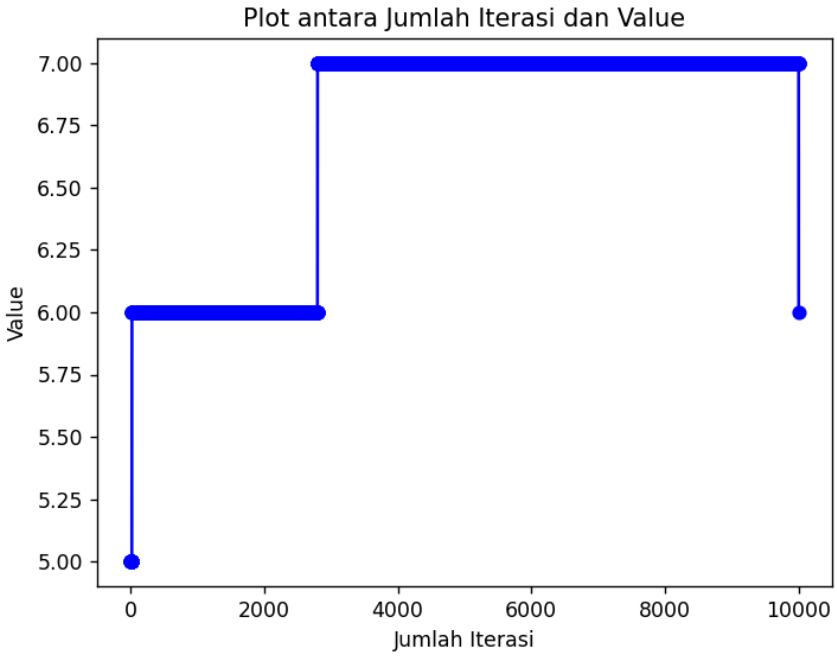
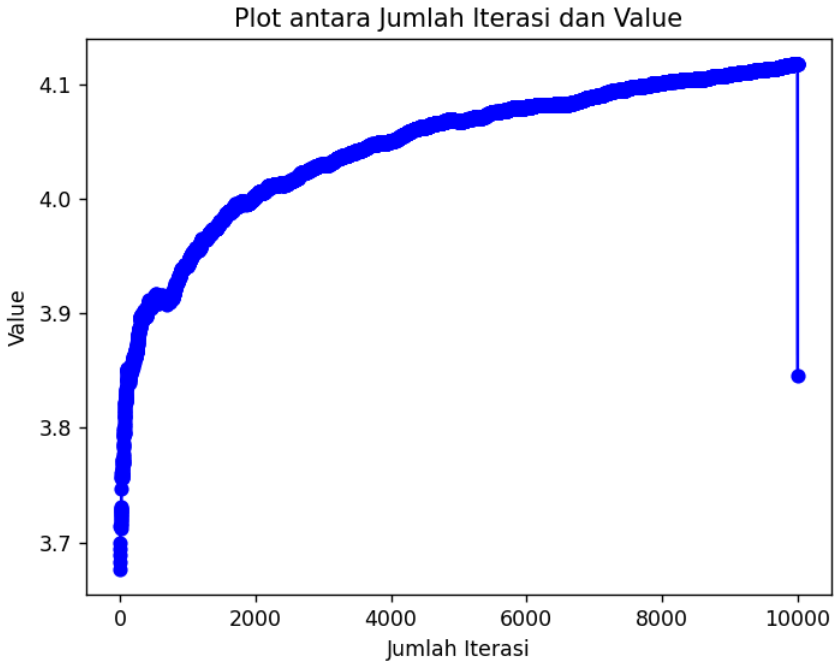
	 Nilai rata-rata 
Jumlah populasi	100
Banyak	1000

iterasi	
Durasi	0.206s

Eksperimen 1-2, populasi sebagai kontrol

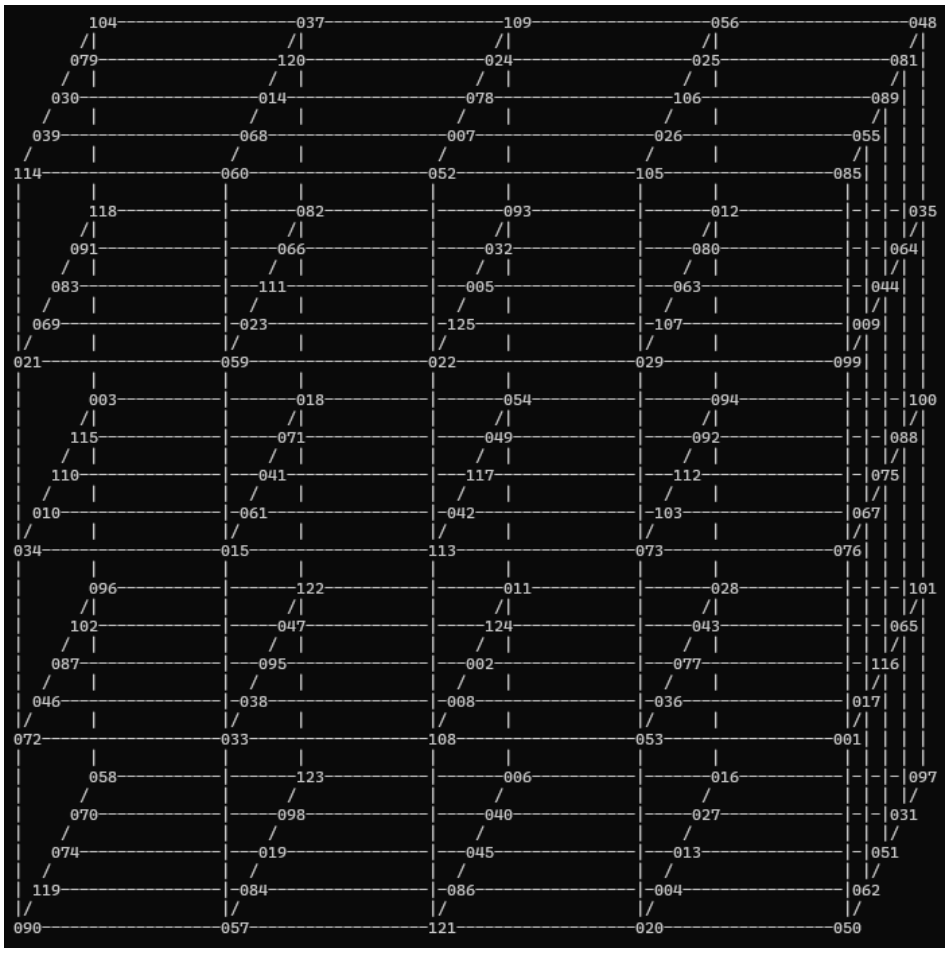
State Awal	

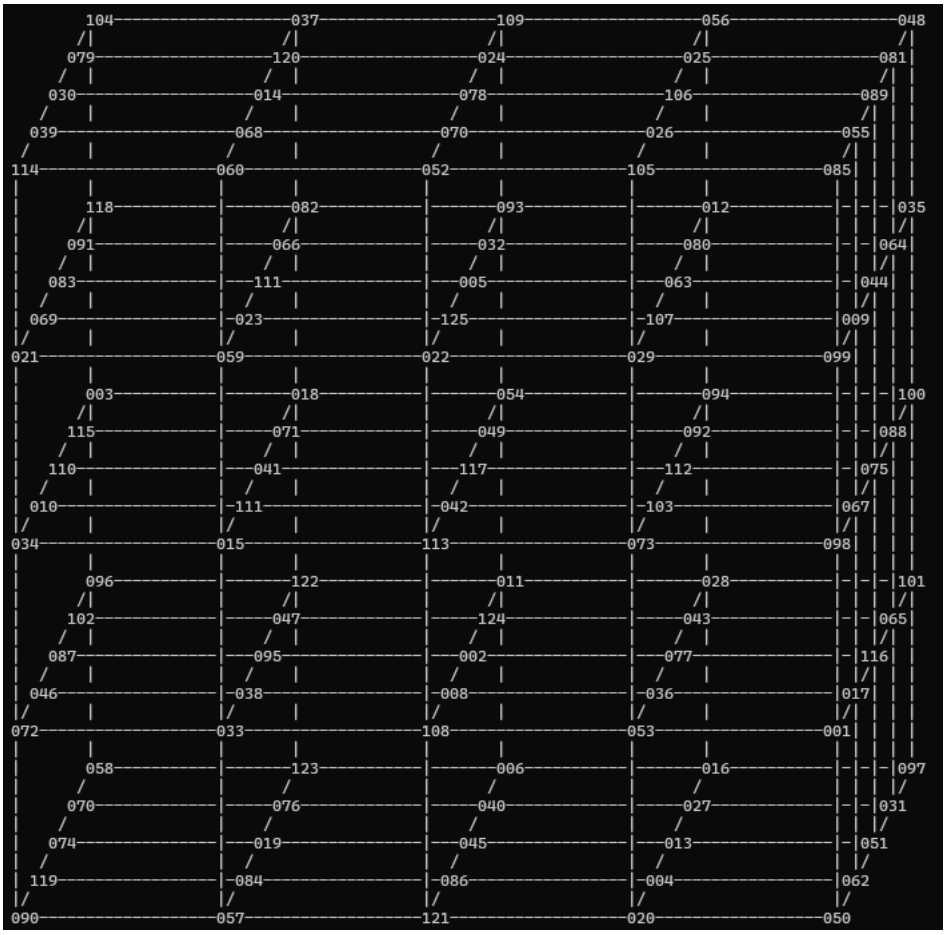
State Akhir	
Nilai Akhir Objective Function	7
Plot (objective function)	Nilai maksimal

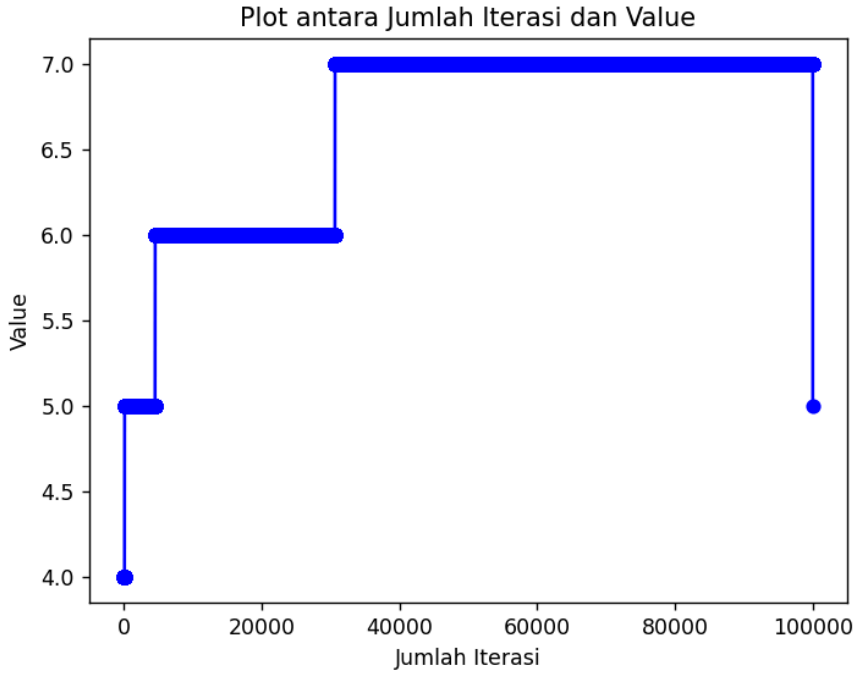
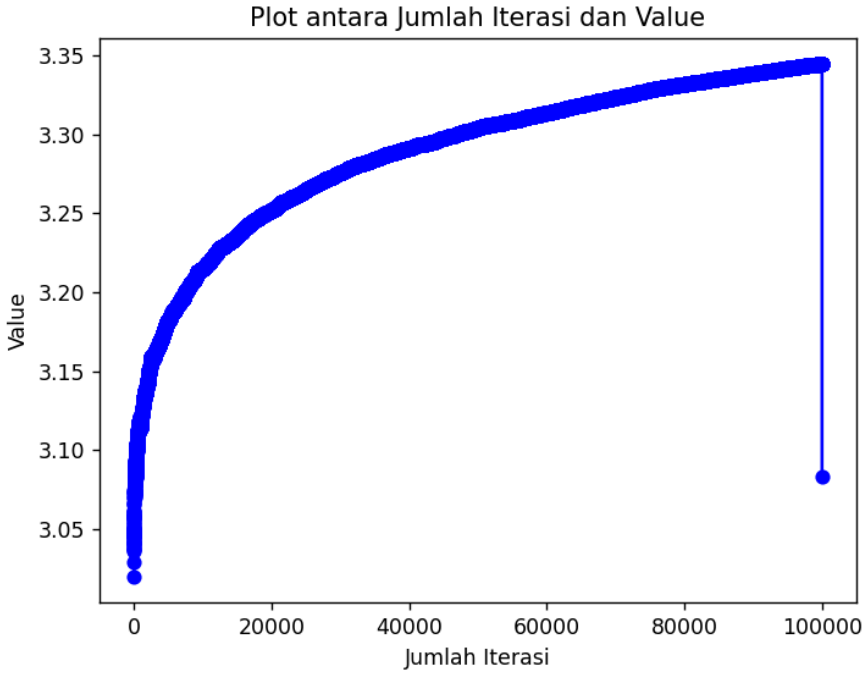
	 Nilai rata-rata 
Jumlah populasi	100
Banyak iterasi	10000

Durasi	4.225s

Eksperimen 1-3, populasi sebagai kontrol

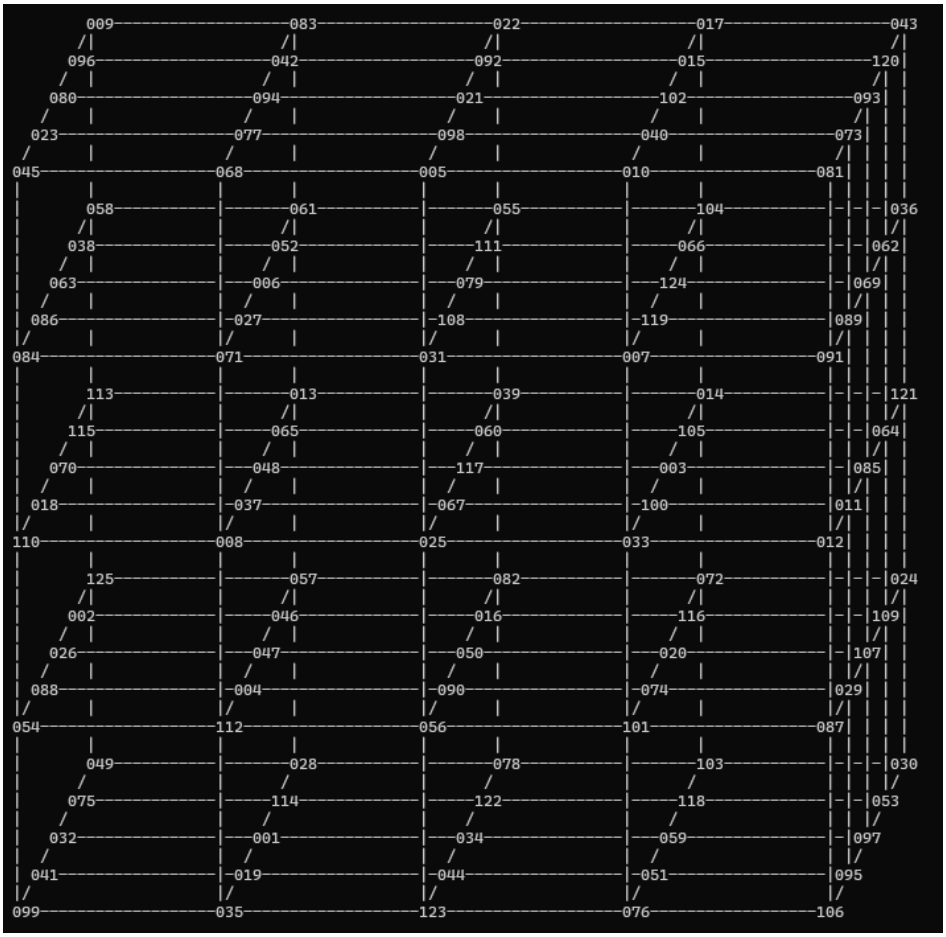
State Awal	

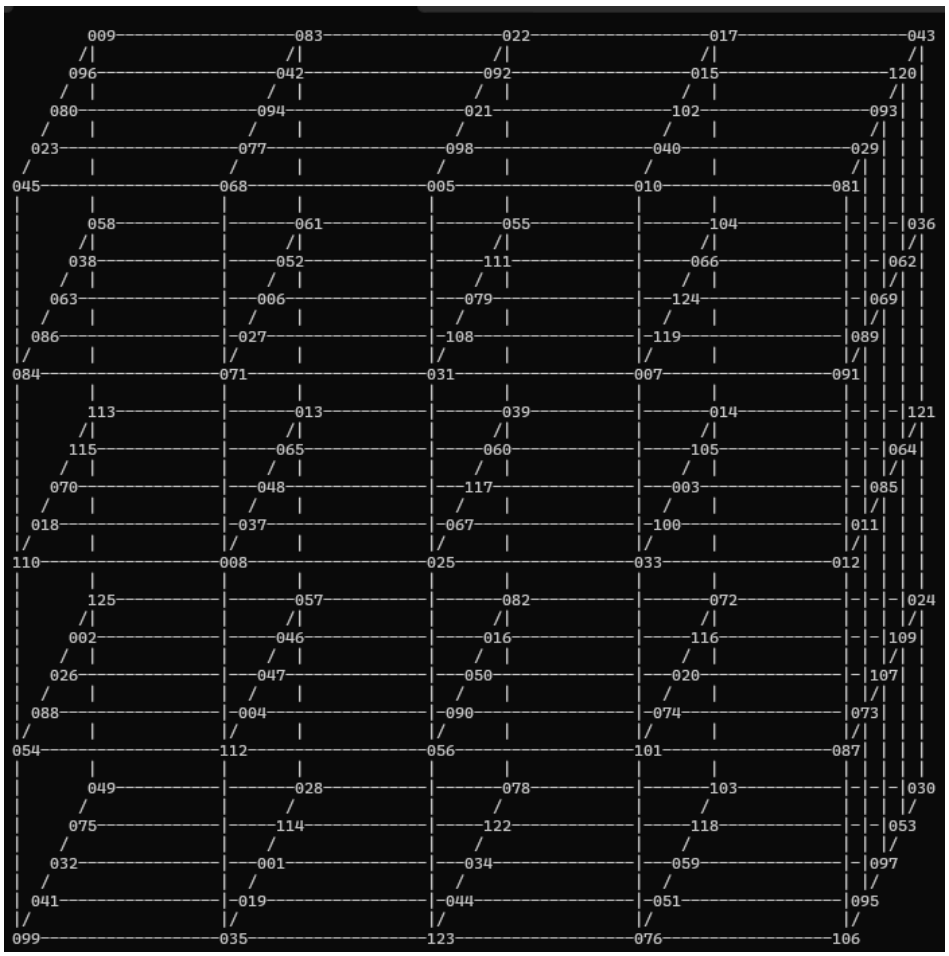
State Akhir	
Nilai Akhir Objective Function	7
Plot (objective function)	Nilai maksimum:

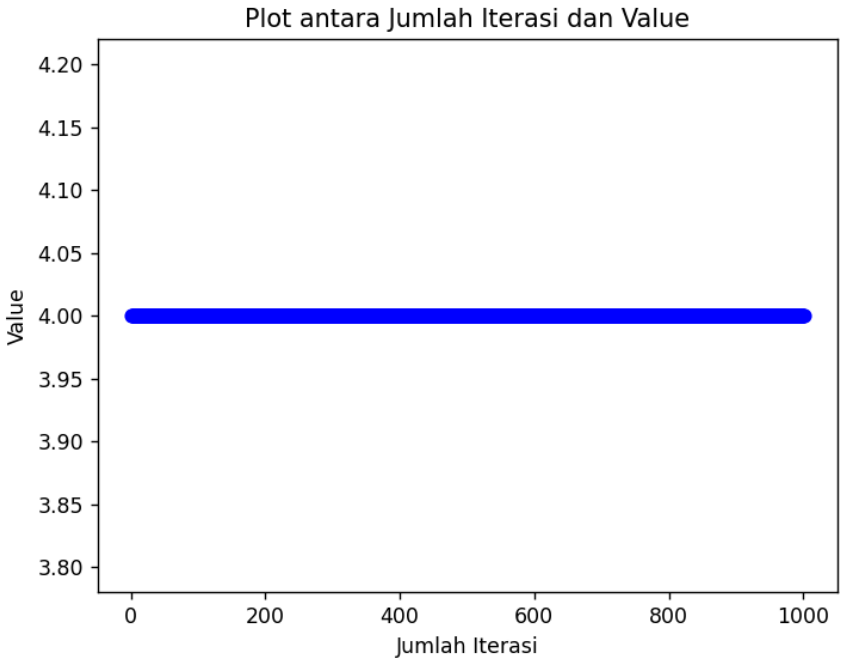
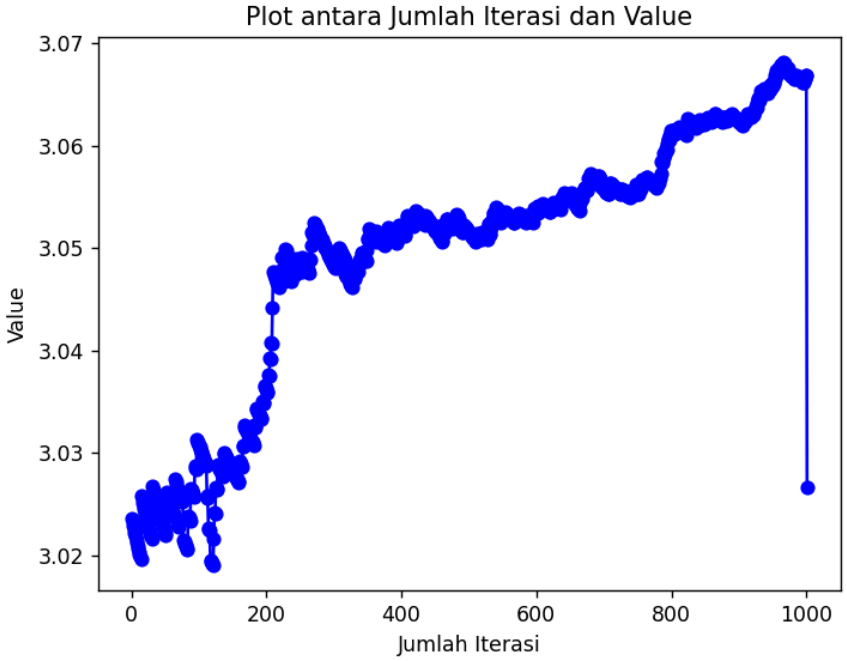
	 Nilai rata-rata: 
Jumlah populasi	100
Banyak	50000

iterasi	
Durasi	73.617s

Eksperimen 2-1, iterasi sebagai kontrol

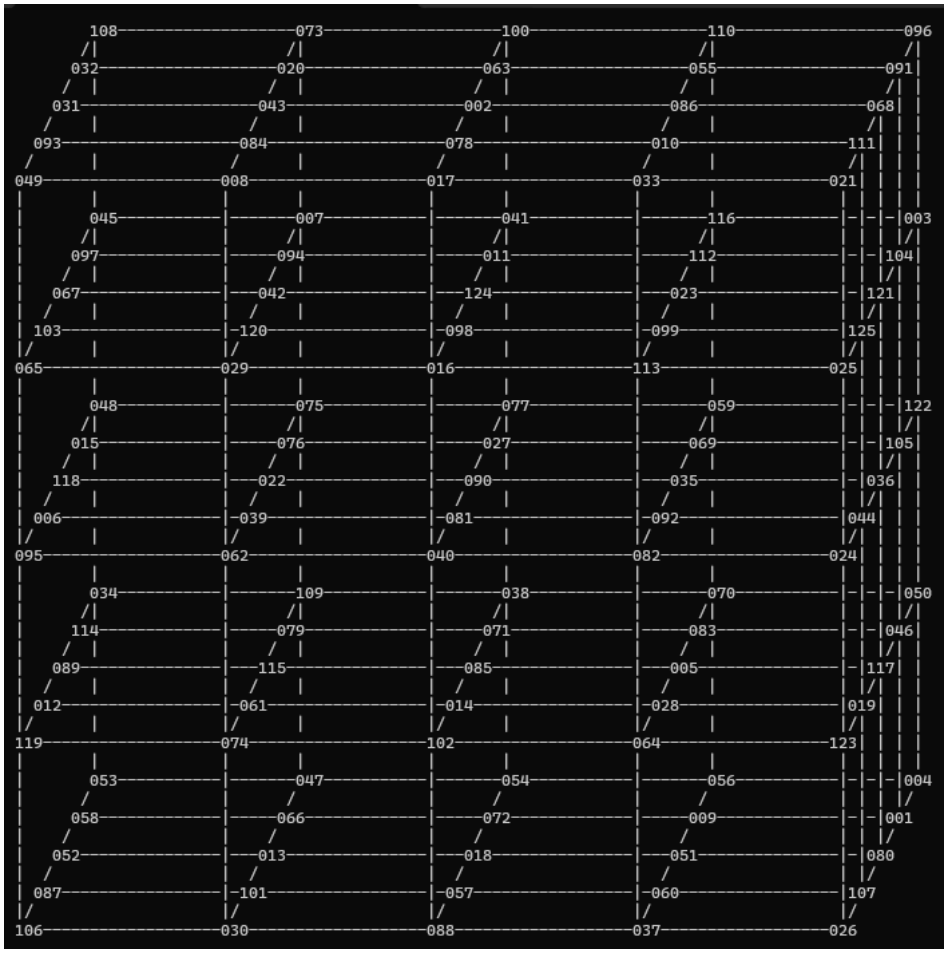
State Awal	

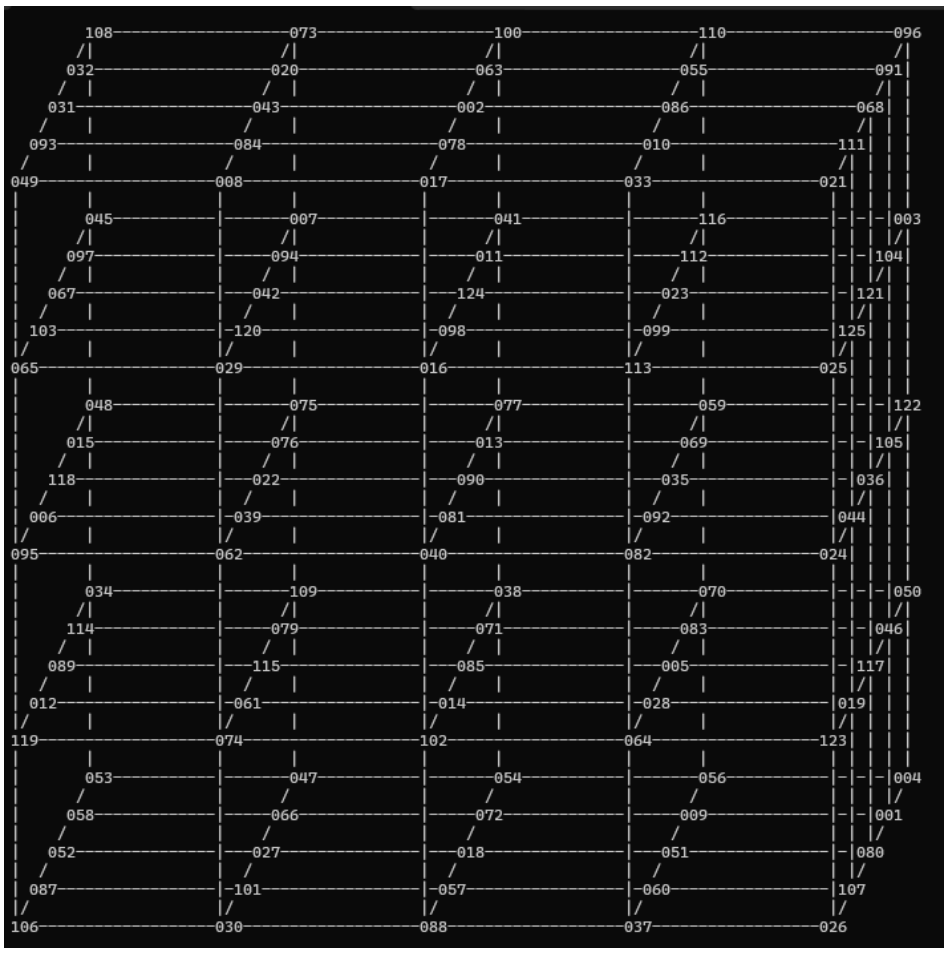
State Akhir	
Nilai Akhir Objective Function	4
Plot (objective function)	Nilai maksimum

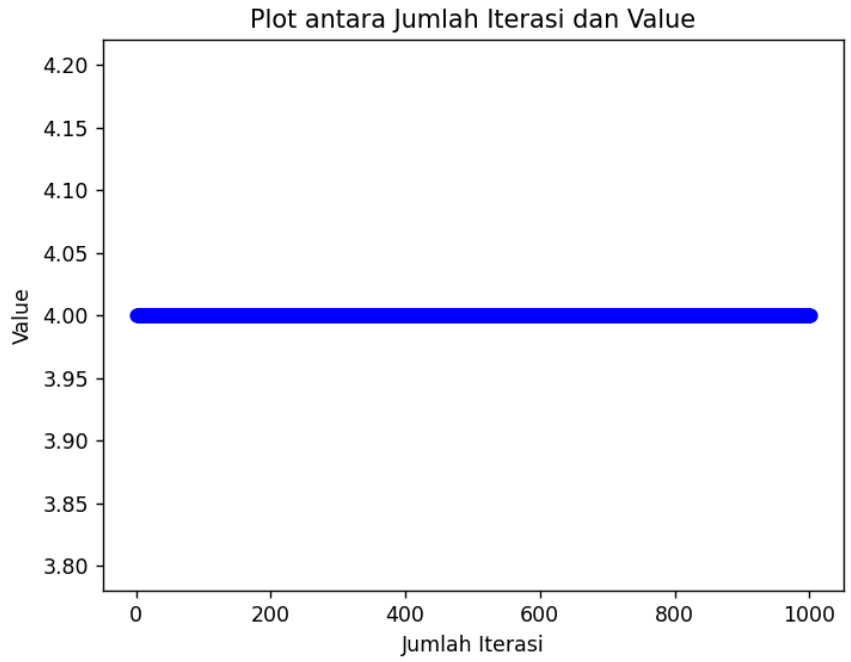
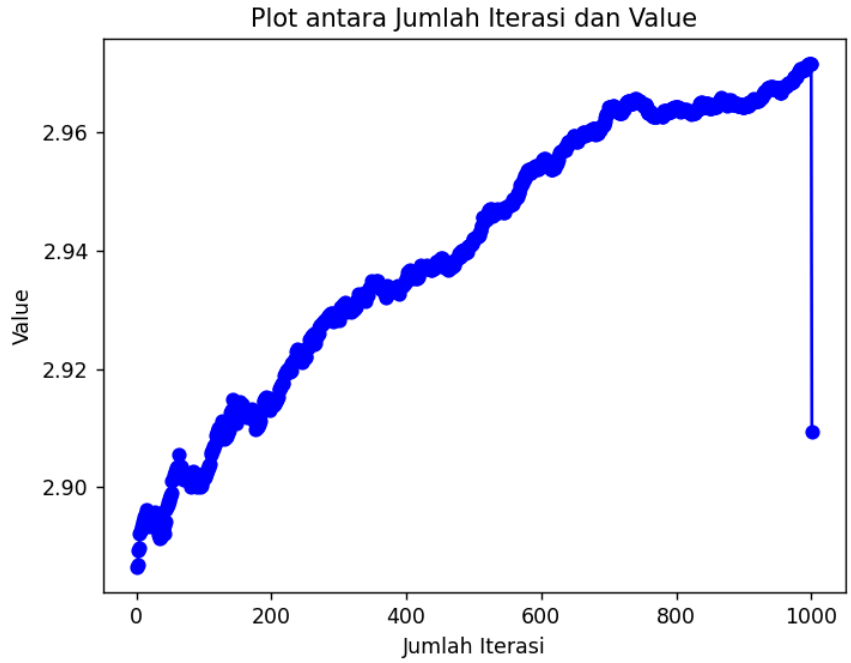
	 Nilai rata-rata 
Jumlah populasi	125
Banyak	1000

iterasi	
Durasi	0.176s

Eksperimen 2-2, iterasi sebagai kontrol

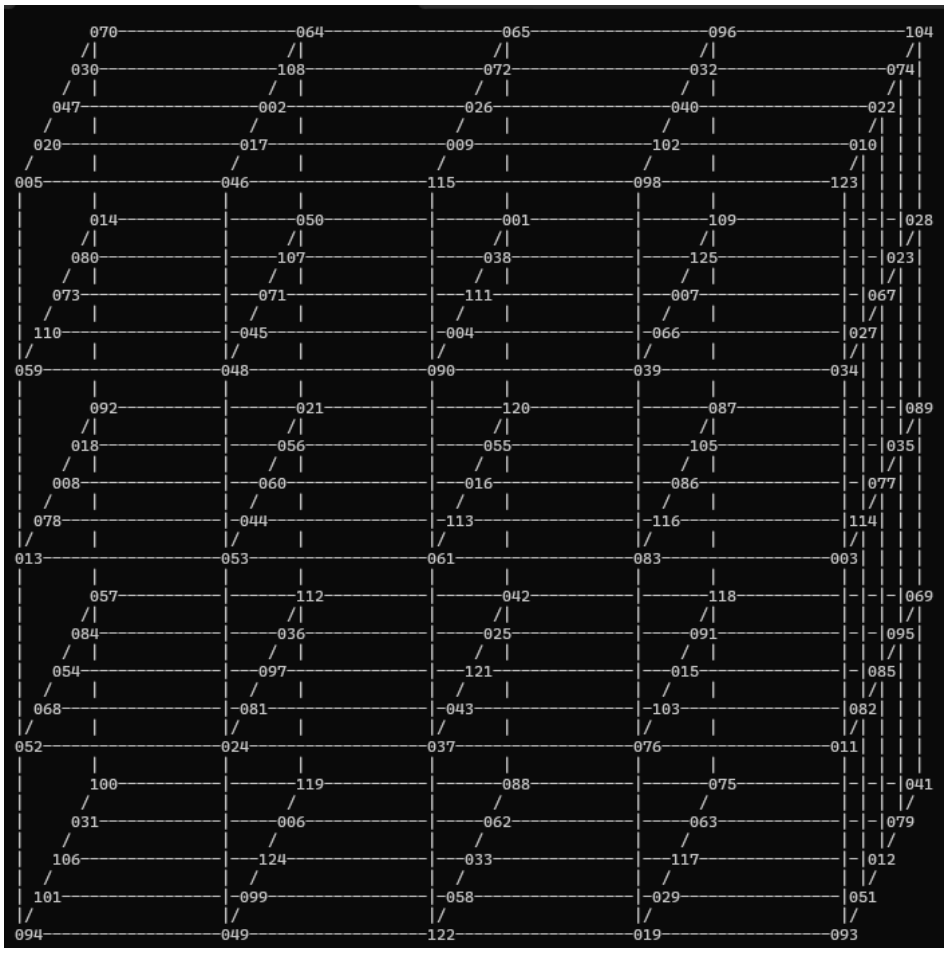
State Awal	

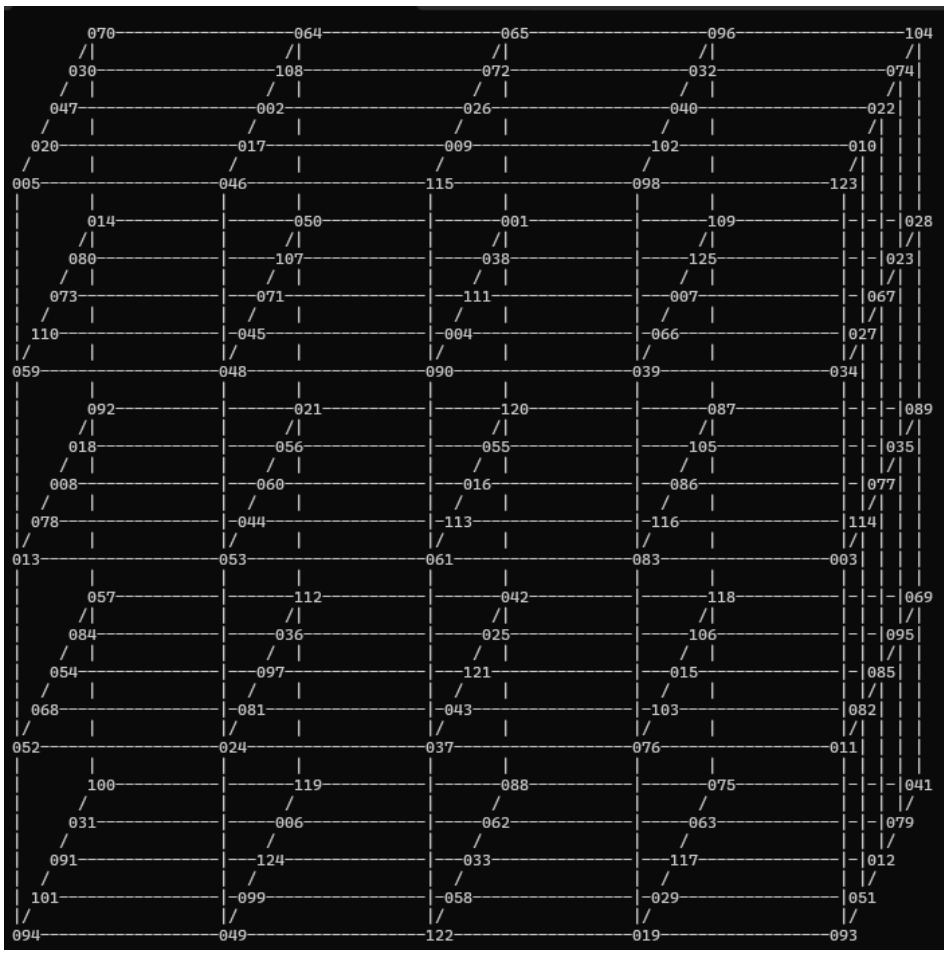
State Akhir	
Nilai Akhir Objective Function	4
Plot (objective function)	Nilai maksimum

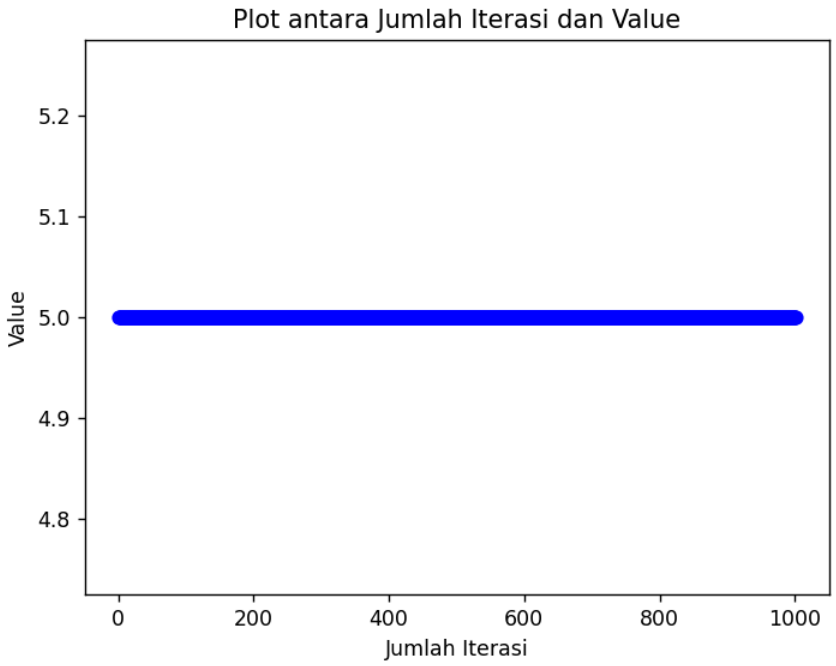
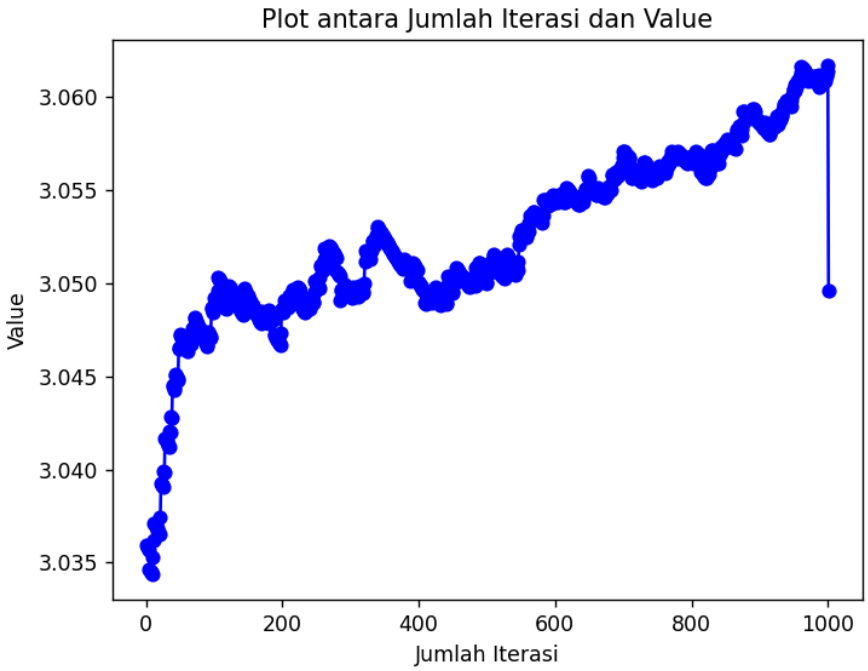
	 Nilai rata-rata 
Jumlah populasi	500
Banyak iterasi	1000

Durasi	0.257s

Eksperimen 2-3, iterasi sebagai kontrol

State Awal	

State Akhir	
Nilai Akhir Objective Function	5
Plot (objective function)	Nilai maksimum

	 Nilai rata-rata 
Jumlah populasi	1000
Banyak	1000

iterasi	
Durasi	0.172s

2.3.2. Analisis

- Setiap algoritma memiliki jarak yang lumayan jauh dari global optima teoritis yaitu 109, tiap algoritma dengan waktu dijalankan hanya < 1 menit hanya dapat mendapatkan nilai maksimal sekitar 3. Hal ini dikarenakan berbagai hal seperti kurangnya iterasi yang dilakukan, dan kemungkinan yang cukup kecil untuk naik ke global optima (mengingat bahwa ada 3844 total state yang mungkin, sedangkan hanya ada 1 global optima.
- Untuk Hill Climb, dengan parameter yang tepat dapat dihasilkan hasil yang mirip pada setiap varian.
- Pada Algoritma Hill Climb, variasi stochastic dapat menghasilkan hasil akhir yang sama dengan waktu yang jauh lebih cepat dibandingkan ke 3 varian lainnya yang cukup mirip.
- Untuk Hill Climb Steepest Ascent didapatkan hasil yang cukup konsisten dimana waktu sekitar 20-30 detik, banyak iterasi 30-40, dan nilai akhir objective 30-40. Untuk Hill Climb with Sideways Move dengan maksimum sideways move 5-10 tetap didapat hasil yang cukup konsisten dimana waktu sekitar 20-30 detik, banyak iterasi 30-40, dan nilai akhir objective 30-40. Hill Climb Random Restart waktunya proporsional dengan banyaknya restart yang diinputkan, banyak iterasi di sekitar 30, dan nilai akhir di sekitar 40. Untuk Hill Climb Stochastic bisa didapatkan nilai objektif akhir 30-40 dengan waktu < 5 detik dengan jumlah langkah maksimal 10000-50000.
- Pada Algoritma Simmulated Annealing, nilai objektif konsisten dihasilkan dalam range 17-22, dengan frekuensi stuck sekitar 980, dan waktu sekitar 0.05 detik. Plot nilai objektif menunjukkan bentuk tangga naik. Tidak adanya nilai objektif yang turun mungkin disebabkan fungsi schedule $T = 0.6^{t-1}$ yang menurunkan temperatur secara cepat. Plot nilai exp(deltaE/T) menunjukkan nilainya sering berganti antara 0 dan 1, dan sangat jarang berada diantaranya. Terlihat pula Simmulated Annealing mirip dengan stochastic hill climbing, terutama jika dibandingkan dengan maksimum iterasi 1000, dengan nilai objektif 18 dan kecepatan 0.025.
- Pada Algoritma Genetik, nilai objektif yang dihasilkan merupakan yang paling rendah dibandingkan algoritma lain, yaitu rata-rata hanya 5-7 dan hanya meningkat 0-1 dari nilai objektif pada state awalnya, untuk jumlah iterasi dibawah 10000 (sekitar 2 detik). Hal ini terjadi karena Algoritma Genetik berdasarkan pada konsep *breeding*/penurunan gen di dunia asli. Karena penurunan gen, anak dari state *parent* yang dipilih akan cenderung mirip dengan kedua *parent*nya, sehingga sangat susah untuk meningkatkan nilai fungsi objektif di algoritma ini.
- Menurut hasil eksperimen, banyak iterasi dan jumlah populasi awal pada Algoritma Genetik masing-masing berbanding lurus dengan nilai objective function akhir. Perubahan jumlah iterasi memiliki peningkatan yang lebih teratur dan terlihat dibandingkan perubahan jumlah populasi awal. Peningkatan nilai akhir dari perubahan

jumlah iterasi berbentuk kurva logaritmik, yang berarti peningkatan tersebut akan menjadi stagnan setelah jumlah iterasi tertentu.

Bab 3: Kesimpulan dan Saran

3.1. Kesimpulan

Kesimpulan yang kita dapat dari membandingkan seluruh algoritma yang digunakan untuk melakukan local search, yaitu kita dapat mengetahui setiap algoritma mempunyai kelebihannya masing-masing. Sebagai contoh, hill-climb secara general menghasilkan solusi yang cukup optimal dan konsisten, namun sangat jarang mencapai solusi yang mendekati global optimum. Untuk kasus simulated annealing dan genetic algorithm, walaupun solusi yang dihasilkan tidak konsisten, tetapi ada kemungkinan solusi yang didapatkan mendekati global optima (walaupun kemungkinannya juga kecil).

3.2. Saran

Sebaiknya melakukan riset tech stack terlebih dahulu untuk mencari tech stack mana yang paling optimal untuk mengerjakan persoalan yang diberikan. Selain itu juga sebaiknya tidak memulai tugas ini H-1.

Pembagian tugas tiap anggota kelompok

No	NIM	Tugas
1	13522010	Genetic Algorithm
2	13522044	Primitif magicCube, Plotting
3	13522059	Steepest Ascent, Sideway Move, Random Restart, dan Stochastic Hill Climbing
4	13522065	Simulated Annealing

Referensi

https://www.magischvierkant.com/three-dimensional-eng/5x5x5/half-pandiagonal-shift/