Tugas Besar 1 IF3170 Inteligensi Artifisial

Pencarian Solusi Diagonal Magic Cube dengan Local Search

Kelompok 41

13522004	Eduardus Alvito Kristiadi
13522007	Irfan Sidiq Permana
13522047	Farel Winalda
13522051	Kharris Khisunica

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2024

DESKRIPSI PERSOALAN

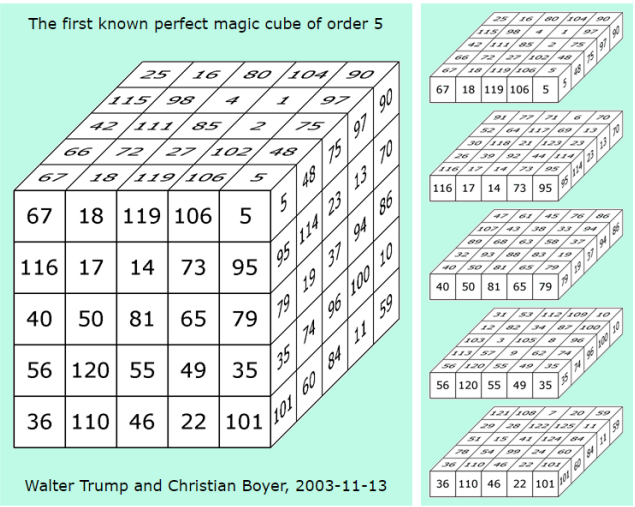
A. Tujuan

Dalam tugas besar 1 ini akan dibahas mengenai pencarian solusi untuk *Diagonal Magic Cube* berukuran 5x5x5 dengan algoritma *local search*. tubes ini bertujuan untuk membuat implementasi beberapa algoritma *local search* yakni *Hill Climbing* dan beberapa variasinya, *Simulated Annealing,* dan *Genetic Algorithm*. Dalam tubes ini juga akan dieksplorasi algoritma mana yang ternyata lebih efektif secara praktik untuk kasus ini.

B. Deskripsi Persoalan dan Penjelasan Singkat

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*

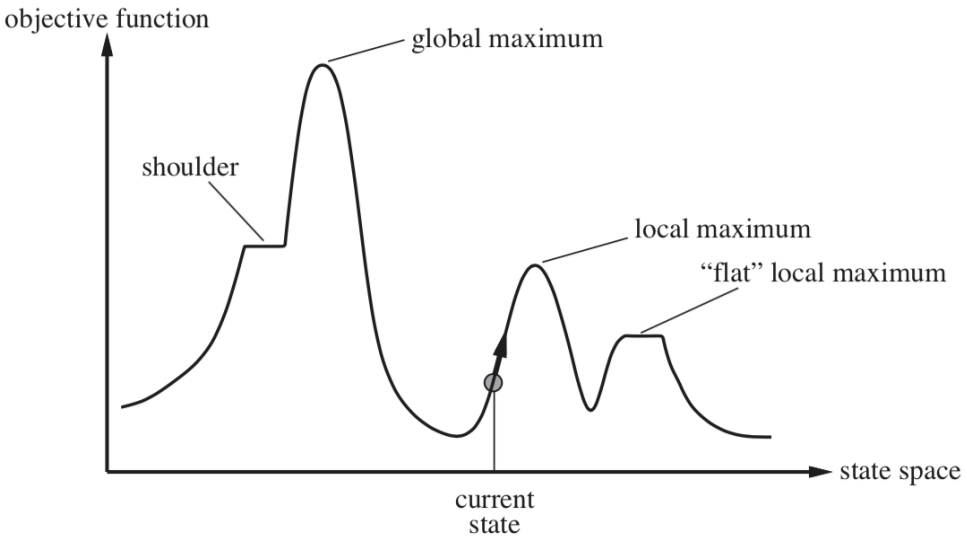


Gambar 1.2 *Magic Cube* berukuran 5 × 5 × 5

Local Search Algorithm / Algoritma Pencarian Lokal merupakan algoritma pengoptimasian yang digunakan untuk menemukan solusi terbaik dalam pencarian ruang tertentu. Berbeda dengan metode pencarian global yang mengeksplor seluruh ruang solusi, algoritma pencarian lokal berfokus pada membuat perubahan secara bertahap untuk meningkatkan solusi saat ini hingga mencapai solusi optimal yang bersifat lokal.

Algoritma *Local Search* berjalan dengan menggunakan satu *current node* dan pada umumnya melakukan perpindahan ke node tetangga tersebut. Algoritma ini memberikan keuntungan, yaitu menggunakan memori yang lebih sedikit karena hanya mempertahankan informasi tentang *node* atau solusi saat ini dan lingkungannya, tidak seperti metode yang lebih global yang memerlukan pencatatan banyak solusi atau status, dapat menemukan solusi yang baik / masuk akal dalam ruang pencarian yang sangat besar atau tak terbatas. Namun, akan tetapi dapat juga menyebabkan terjebaknya di lokal maksimum yang belum tentu merupakan solusi global terbaik namun merupakan solusi lokal terbaik sehingga terkadang terdapat beberapa kasus yang menyebabkan solusi tidak optimal.

Untuk menggambarkan pencarian dari suatu lingkup solusi dengan nilai dari fungsi objektif / heuristik yang ingin dicapai dapat dinyatakan seperti gambar berikut



Gambar 1.2 Grafik ruang state dan solusi

Keterangan:

- Sumbu-x : ruang state
- Sumbu-y : ruang solusi / nilai fungsi objektif
- Global maxima : puncak yang memiliki nilai fungsi objektif paling tinggi yang merupakan solusi dari pencarian
- Local maxima : puncak yang memiliki nilai fungsi objektif lebih tinggi daripada state sebelumnya, namun tidak lebih tinggi daripada global maxima
- Plateau / flat : puncak yang datar yang memiliki nilai fungsi objektif sama untuk beberapa state yang beruntun

PEMBAHASAN

A. Objective Function

Sebuah *Diagonal Magic Cube* memiliki satu angka *magic cube* unik yang bergantung dari panjang sisinya. Untuk kubus bersisi n, maka *Magic cube number* nya adalah $\frac{n(n^3+1)}{2}$. Untuk kubus 5x5x5, panjang sisinya adalah 5. Maka bisa didapat *Magic cube number* dari kubus 5x5x5 = M = 315. Angka ini adalah angka yang didapat dengan menjumlahkan setiap garis lurus yang dapat dibentuk dalam satu kubus.

Dalam satu kubus, ada $3m^2 + 6m + 4$ garis. Dimana terdapat m^2 baris, m^2 kolom, m^2 tiang, $2 \times 3 \times m$ diagonal sisi, dan 4 diagonal ruang. Untuk kubus bersisi 5, maka terdapat 25 baris, 25 kolom, 25 tiang, 30 diagonal sisi, dan 4 diagonal ruang

Misal B_i adalah jumlah angka dalam baris ke i, K_i adalah jumlah angka dalam kolom ke i, T_i adalah jumlah angka dalam kolom ke i, DS_i adalah jumlah angka dalam diagonal sisi ke i, dan DR_i adalah jumlah angka dalam diagonal ruang ke i.

Sebuah kubus disebut *Diagonal Magic Cube* jika dan hanya jika jumlah dari setiap garis sama dengan M. Jika terdapat selisih antara jumlah angka dalam satu garis dengan M, maka kubus itu belum mencapai *Diagonal Magic Cube*.

Total selisih setiap garis dengan *Magic Number*:

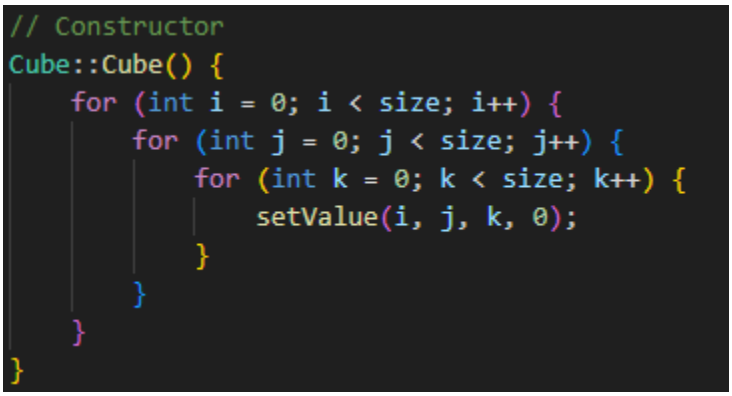
$$\sum_{1}^{25}\left|B_i - M\right| + \sum_{1}^{25}\left|K_i - M\right| + \sum_{1}^{25}\left|T_i - M\right| + \sum_{1}^{30}\left|DS_i - M\right| + \sum_{1}^{4}\left|DR_i - M\right|$$

. *Diagonal Magic Cube* akan menghasilkan selisih 0 jika setiap garisnya dimasukkan ke dalam persamaan di atas, dan non *Diagonal Magic Cube* akan menghasilkan selisih >0. Maka *Objective Function* dari permasalahan ini adalah:

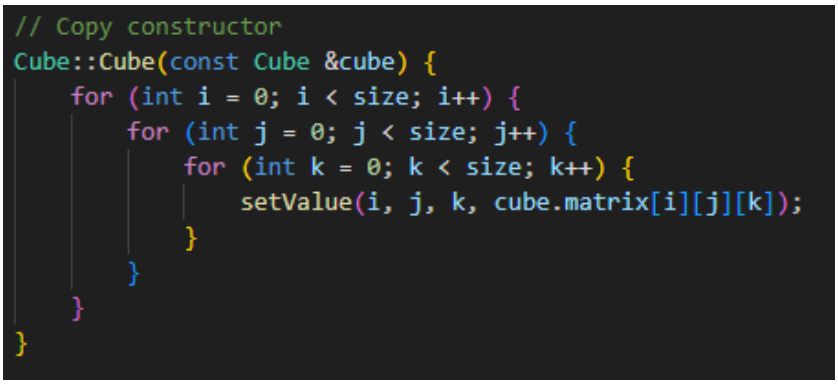
0 - Total selisih setiap garis dengan *Magic Number*, dan persoalan ini untuk mencari Global Maksimum, yakni 0.

B. Algoritma Local Search

0. Class Cube dan Utils Umum

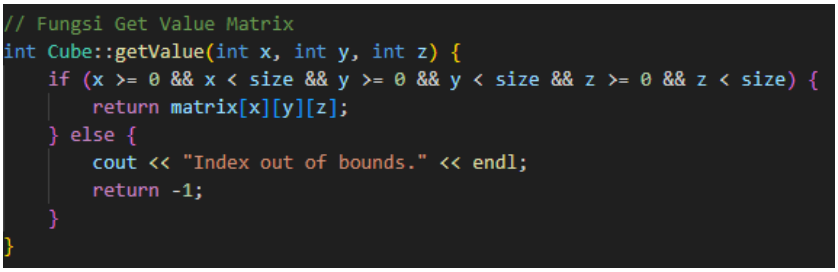
Nama Fungsi	Fungsi Cube()
Deskripsi Fungsi	Fungsi constructor pembuatan objek Cube
Source Code	

```
// Constructor
Cube::Cube() {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            for (int k = 0; k < size; k++) {
                setValue(i, j, k, 0);
            }
        }
    }
}
```

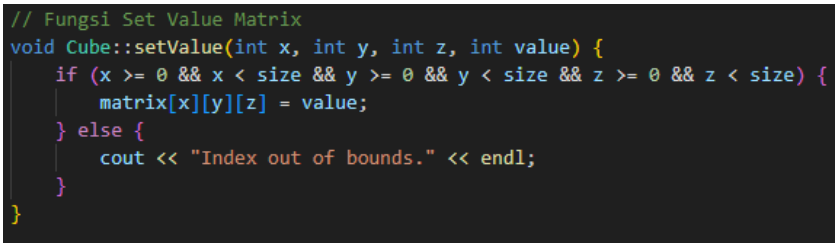
Nama Fungsi	Fungsi Cube(const Cube &cube)
Deskripsi Fungsi	Fungsi copy constructor pembuatan objek Cube
Source Code	

```
// Copy constructor
Cube::Cube(const Cube &cube) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            for (int k = 0; k < size; k++) {
                setValue(i, j, k, cube.matrix[i][j][k]);
            }
        }
    }
}
```

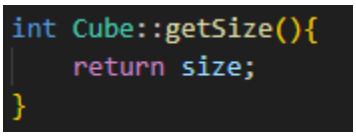
Nama Fungsi	int getValue(int x, int y, int z)
Deskripsi Fungsi	Fungsi Primitif mendapatkan value cube

Source Code	

```
// Fungsi Get Value Matrix
int Cube::getValue(int x, int y, int z) {
    if (x >= 0 && x < size && y >= 0 && y < size && z >= 0 && z < size) {
        return matrix[x][y][z];
    } else {
        cout << "Index out of bounds." << endl;
        return -1;
    }
}
```

Nama Fungsi	void setValue(int x, int y, int z, int value)
Deskripsi Fungsi	Fungsi Primitif Set value matrix Cube
Source Code	

```
// Fungsi Set Value Matrix
void Cube::setValue(int x, int y, int z, int value) {
    if (x >= 0 && x < size && y >= 0 && y < size && z >= 0 && z < size) {
        matrix[x][y][z] = value;
    } else {
        cout << "Index out of bounds." << endl;
    }
}
```

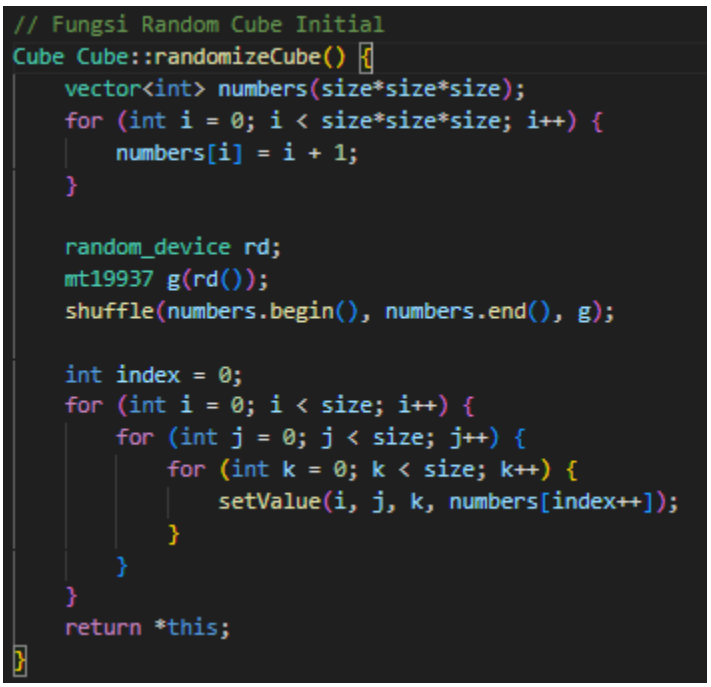
Nama Fungsi	int getSize()
Deskripsi Fungsi	Fungsi Primitif mendapatkan size cube
Source Code	

```
int Cube::getSize(){
    return size;
}
```

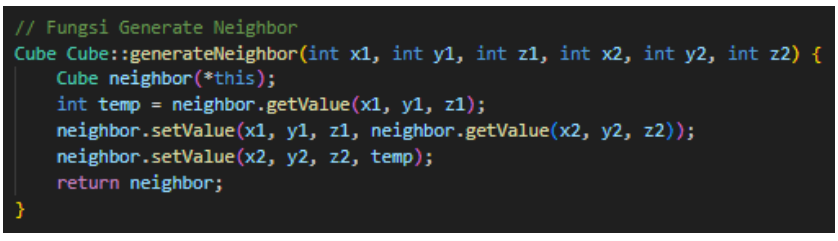
Nama Fungsi	int objectiveFunction()
Deskripsi Fungsi	Fungsi mencari nilai objektif dari cube sekarang

Source Code

```
// Fungsi menghitung State Value
int Cube::objectiveFunction() {
    int totalDifference = 0;

    // Sumbu X
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            int rowSum = 0;
            for (int k = 0; k < size; k++) {
                rowSum += this->getValue(i, j, k);
            }
            totalDifference += abs(rowSum - magicnumber);
        }
    }

    // Sumbu Y
    for (int j = 0; j < size; j++) {
        for (int k = 0; k < size; k++) {
            int colSum = 0;
            for (int i = 0; i < size; i++) {
                colSum += this->getValue(i, j, k);
            }
            totalDifference += abs(colSum - magicnumber);
        }
    }

    // Sumbu Z
    for (int i = 0; i < size; i++) {
        for (int k = 0; k < size; k++) {
            int pillarSum = 0;
            for (int j = 0; j < size; j++) {
                pillarSum += this->getValue(i, j, k);
            }
            totalDifference += abs(pillarSum - magicnumber);
        }
    }

    // Semua diagonal sisi
    for (int i = 0; i < size; i++) {
        int mainDiagSum1 = 0;
        int mainDiagSum2 = 0;
        for (int j = 0; j < size; j++) {
            mainDiagSum1 += this->getValue(i, j, j);
            mainDiagSum2 += this->getValue(i, j, (size - 1) - j);
        }
        totalDifference += abs(mainDiagSum1 - magicnumber);
        totalDifference += abs(mainDiagSum2 - magicnumber);
    }

    // Semua diagonal ruang
    int spaceDiag1 = 0, spaceDiag2 = 0, spaceDiag3 = 0, spaceDiag4 = 0;
    for (int i = 0; i < size; i++) {
        spaceDiag1 += this->getValue(i, i, i);
        spaceDiag2 += this->getValue(i, i, (size - 1) - i);
        spaceDiag3 += this->getValue(i, (size - 1) - i, i);
        spaceDiag4 += this->getValue(i, (size - 1) - i, (size - 1) - i);
    }
    totalDifference += abs(spaceDiag1 - magicnumber);
    totalDifference += abs(spaceDiag2 - magicnumber);
    totalDifference += abs(spaceDiag3 - magicnumber);
    totalDifference += abs(spaceDiag4 - magicnumber);

    return 0 - totalDifference;
}
```

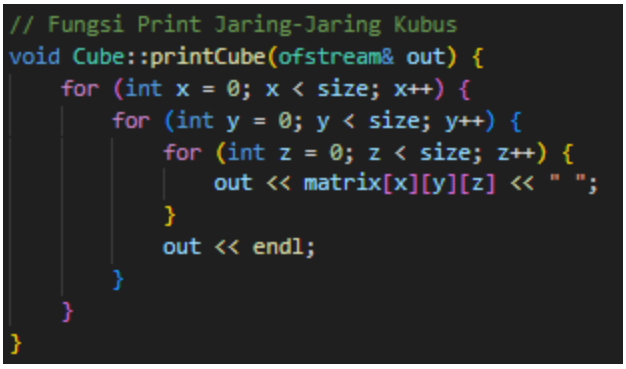
Nama Fungsi	Cube randomizeCube()
Deskripsi Fungsi	Fungsi seeding secara random ke Cube
Source Code	

```cpp
// Fungsi Random Cube Initial
Cube Cube::randomizeCube() {
    vector<int> numbers(size*size*size);
    for (int i = 0; i < size*size*size; i++) {
        numbers[i] = i + 1;
    }

    random_device rd;
    mt19937 g(rd());
    shuffle(numbers.begin(), numbers.end(), g);

    int index = 0;
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            for (int k = 0; k < size; k++) {
                setValue(i, j, k, numbers[index++]);
            }
        }
    }
    return *this;
}
```

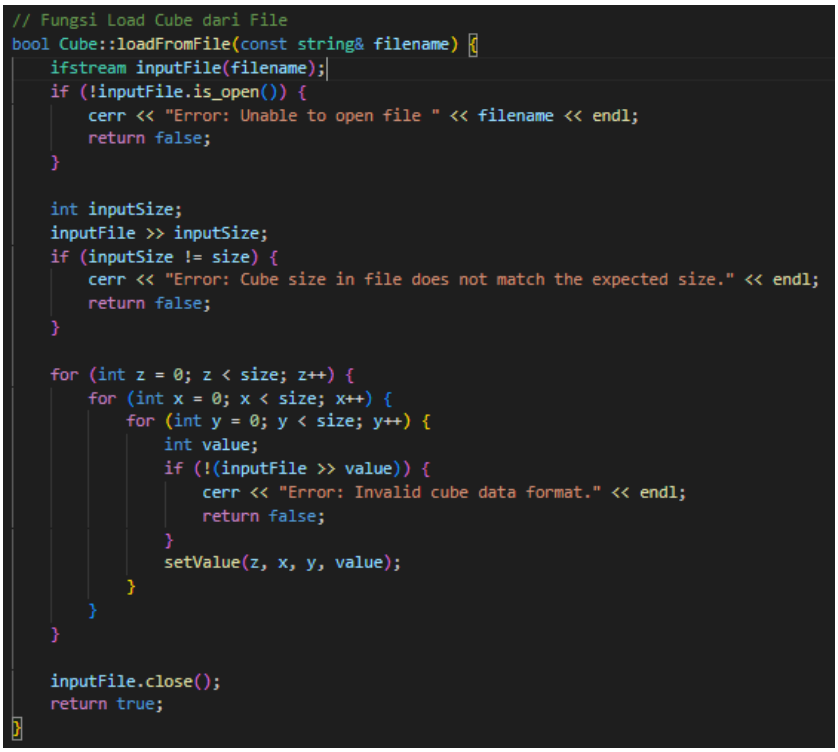
Nama Fungsi	Cube generateNeighbor(int x1, int y1, int z1, int x2, int y2, int z2)
Deskripsi Fungsi	Fungsi membuat cube baru
Source Code	

```cpp
// Fungsi Generate Neighbor
Cube Cube::generateNeighbor(int x1, int y1, int z1, int x2, int y2, int z2) {
    Cube neighbor(*this);
    int temp = neighbor.getValue(x1, y1, z1);
    neighbor.setValue(x1, y1, z1, neighbor.getValue(x2, y2, z2));
    neighbor.setValue(x2, y2, z2, temp);
    return neighbor;
}
```

Nama Fungsi	void printCube(ofstream& out)
Deskripsi Fungsi	Fungsi print Cube ke file

Source Code	

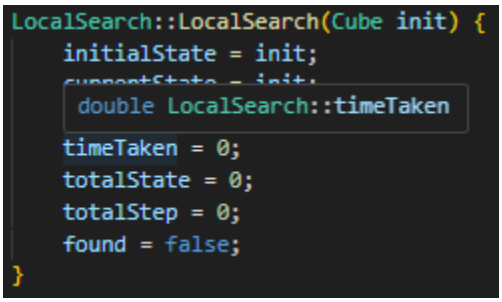
```
// Fungsi Print Jaring-Jaring Kubus
void Cube::printCube(ofstream& out) {
    for (int x = 0; x < size; x++) {
        for (int y = 0; y < size; y++) {
            for (int z = 0; z < size; z++) {
                out << matrix[x][y][z] << " ";
            }
            out << endl;
        }
    }
}
```

Nama Fungsi	bool loadFromFile(const string& filename)
Deskripsi Fungsi	Fungsi Load Cube dari File
Source Code	

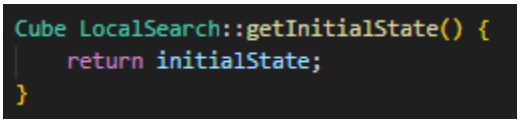
```
// Fungsi Load Cube dari File
bool Cube::loadFromFile(const string& filename) {
    ifstream inputFile(filename);
    if (!inputFile.is_open()) {
        cerr << "Error: Unable to open file " << filename << endl;
        return false;
    }

    int inputSize;
    inputFile >> inputSize;
    if (inputSize != size) {
        cerr << "Error: Cube size in file does not match the expected size." << endl;
        return false;
    }

    for (int z = 0; z < size; z++) {
        for (int x = 0; x < size; x++) {
            for (int y = 0; y < size; y++) {
                int value;
                if (!(inputFile >> value)) {
                    cerr << "Error: Invalid cube data format." << endl;
                    return false;
                }
                setValue(z, x, y, value);
            }
        }
    }

    inputFile.close();
    return true;
}
```

1. *Class Local Search (Parent)*

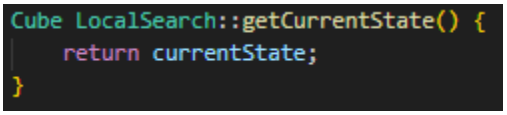
Deskripsi Algoritma	Class Parent yang berisikan atribut-atribut dasar dari Local Search dan beberapa primitf serta function-function global lainnya

Nama Fungsi	LocalSearch(Cube init)
Deskripsi Fungsi	Fungsi constructor Class LocalSearch
Source Code	

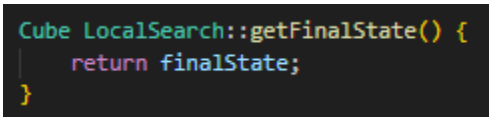
```
LocalSearch::LocalSearch(Cube init) {
    initialState = init;
    currentState = init;
    double LocalSearch::timeTaken
    timeTaken = 0;
    totalState = 0;
    totalStep = 0;
    found = false;
}
```

Nama Fungsi	Cube getInitialState()
Deskripsi Fungsi	Fungsi primitif get cube awal
Source Code	

```
Cube LocalSearch::getInitialState() {
    return initialState;
}
```

Nama Fungsi	Cube getCurrentState()
Deskripsi Fungsi	Fungsi primitif get cube sekarang
Source Code	

```
Cube LocalSearch::getCurrentState() {
    return currentState;
}
```

Nama Fungsi	Cube getFinalState()
Deskripsi Fungsi	Fungsi primitif get cube final
Source Code	

```
Cube LocalSearch::getFinalState() {
    return finalState;
}
```

Nama Fungsi	double getTimeTaken()
Deskripsi Fungsi	Fungsi primitif get waktu yang dipakai

Source Code	`double LocalSearch::getTimeTaken() { return timeTaken; }`

Nama Fungsi	int getTotalState()
Deskripsi Fungsi	Fungsi primitif get total Cube yang dieksplor
Source Code	`int LocalSearch::getTotalState() { return totalState; }`

Nama Fungsi	bool isFound()
Deskripsi Fungsi	Fungsi primitf get solusi ketemu atau tidak
Source Code	`bool LocalSearch::isFound() { return found; }`

Nama Fungsi	vector<tuple<Cube, int, int>> getAllSteps()
Deskripsi Fungsi	Fungsi get semua step beserta value yang berubah
Source Code	`vector<tuple<Cube, int, int>> LocalSearch::getAllSteps() { return allStep; }`

Nama Fungsi	int getTotalStep()
Deskripsi Fungsi	Fungsi primitif get total cube hingga iterasi terakhir
Source Code	`int LocalSearch::getTotalStep() { return totalStep; }`

Nama Fungsi	void addNextStep(Cube nextState, int value1, int value2)

Deskripsi Fungsi	Fungsi memasukkan cube next step
Source Code	

```
void LocalSearch::addNextStep(Cube nextState, int value1, int value2) {
    allStep.emplace_back(nextState, value1, value2);
    totalStep++;
}
```

Nama Fungsi	tuple<Cube, int, int, int> generateRandomNeighbor(Cube state)
Deskripsi Fungsi	Fungsi Generate Random Neighbor
Source Code	

```
tuple<Cube, int, int, int> LocalSearch::generateRandomNeighbor(Cube state)
{
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<int> dist(0, state.getSize() - 1);

    vector<int> position1(3);
    vector<int> position2(3);
    bool isMoveValid = false;

    while (!isMoveValid) {
        for (int i = 0; i < 3; ++i) {
            position1[i] = dist(gen);
            position2[i] = dist(gen);
        }

        if (position1 != position2) {
            isMoveValid = true;
        }
    }

    Cube neighbor = state.generateNeighbor(position1[0], position1[1], position1[2], position2[0], position2[1], position2[2]);
    int value1 = state.getValue(position1[0], position1[1], position1[2]);
    int value2 = state.getValue(position2[0], position2[1], position2[2]);

    return make_tuple(neighbor, neighbor.objectiveFunction(), value1, value2);
}
```

Nama Fungsi	vector<tuple<Cube, int, int, int>> generateAllNeighbors()
Deskripsi Fungsi	Fungsi Generate semua neighbor
Source Code	

```
std::vector<std::tuple<Cube, int, int, int>> LocalSearch::generateAllNeighbors() {
    vector<tuple<Cube, int, int, int>> neighborValues;

    for (int x1 = 0; x1 < initialState.getSize(); ++x1) {
        for (int y1 = 0; y1 < initialState.getSize(); ++y1) {
            for (int z1 = 0; z1 < initialState.getSize(); ++z1) {
                for (int x2 = 0; x2 < initialState.getSize(); ++x2) {
                    for (int y2 = 0; y2 < initialState.getSize(); ++y2) {
                        for (int z2 = 0; z2 < initialState.getSize(); ++z2) {
                            if (x1 != x2 || y1 != y2 || z1 != z2) {
                                int value1 = currentState.getValue(x1, y1, z1);
                                int value2 = currentState.getValue(x2, y2, z2);

                                Cube neighbor = currentState.generateNeighbor(x1, y1, z1, x2, y2, z2);
                                int neighborObjectiveValue = neighbor.objectiveFunction();
                                neighborValues.push_back(make_tuple(neighbor, neighborObjectiveValue, value1, value2));
                                totalState++;
                            }
                        }
                    }
                }
            }
        }
    }

    return neighborValues;
}
```

Nama Fungsi	int generateRandomInteger(int minValue, int maxValue)

Deskripsi Fungsi	Fungsi Generate random integer

Source Code	`int LocalSearch::generateRandomInteger(int minValue, int maxValue)`
	`{`
	`  random_device rd;`
	`  mt19937 gen(rd());`
	`  uniform_int_distribution<int> dist(minValue, maxValue);`
	`  return dist(gen);`
	`}`

Nama Fungsi	double generateRandomRealNumber(double minValue, double maxValue)
Deskripsi Fungsi	Fungsi Generate random real number

Source Code	`double LocalSearch::generateRandomRealNumber(double minValue, double maxValue)`
	`{`
	`  random_device rd;`
	`  mt19937 gen(rd());`
	`  uniform_real_distribution<> dist(0.0, 1.0);`
	`  return dist(gen);`
	`}`

2. *Steepest Ascent Hill Climbing*

Deskripsi Algoritma	Class Algoritma Steepest Ascent Hill Climbing yang menginherit Class LocalSearch

Nama Fungsi	SteepestAscent(Cube init)
Deskripsi Fungsi	Fungsi constructor untuk class SteepestAscent
Source Code	`// Constructor` `SteepestAscent::SteepestAscent(Cube init) : LocalSearch(init) {}`

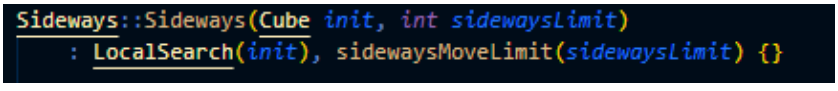
Nama Fungsi	void Algorithm()
Deskripsi Fungsi	Fungsi yang berisi algoritma Steepest Ascent mulai dari state awal hingga state final

Source Code	

```
void SteepestAscent::Algorithm() {
    auto start = high_resolution_clock::now();

    currentState = initialState;
    int currentObjectiveValue = currentState.objectiveFunction();

    bool foundBetter = true;

    while (foundBetter) {
        foundBetter = false;
        generateAllNeighbors();

        Cube bestNeighbor;
        int bestNeighborValue = currentObjectiveValue;
        int value1 = 0, value2 = 0;

        for (auto &neighborTuple : neighborValues) {
            Cube neighbor = std::get<0>(neighborTuple);
            int neighborObjectiveValue = std::get<1>(neighborTuple);
            int currentValue1 = std::get<2>(neighborTuple);
            int currentValue2 = std::get<3>(neighborTuple);

            if (neighborObjectiveValue > bestNeighborValue) {
                bestNeighbor = neighbor;
                bestNeighborValue = neighborObjectiveValue;
                value1 = currentValue1;
                value2 = currentValue2;
                foundBetter = true;
            }
        }

        if (foundBetter) {
            currentState = bestNeighbor;
            currentObjectiveValue = bestNeighborValue;
            addNextStep(currentState, value1, value2);
        }
    }

    finalState = currentState;
    if (finalState.objectiveFunction() == 0) {
        found = true;
    }

    auto end = high_resolution_clock::now();
    timeTaken = duration_cast<duration<double>>(end - start).count();
}
```

3. *Hill Climbing with Sideways Move*

Deskripsi Algoritma	Class Algoritma Hill Climbing with Sideways Move yang menginherit Class LocalSearch

Nama Fungsi	Sideways(Cube init, int sidewaysLimit)
Deskripsi Fungsi	Fungsi constructor untuk class Sideways
Source Code	

```
Sideways::Sideways(Cube init, int sidewaysLimit)
    : LocalSearch(init), sidewaysMoveLimit(sidewaysLimit) {}
```

Nama Fungsi	void Sideways::Algorithm()

Deskripsi Fungsi	Fungsi yang berisi algoritma Sideways mulai dari state awal hingga state final
Source Code	

```
void Sideways::Algorithm() {
    auto start = high_resolution_clock::now();

    currentState = initialState;
    int currentObjectiveValue = currentState.objectiveFunction();
    int sidewaysMoves = 0;
    bool foundBetter = true;

    while (foundBetter) {
        foundBetter = false;
        generateAllNeighbors();

        for (const auto &neighborTuple : neighborValues) {
            Cube neighbor = get<0>(neighborTuple);
            int neighborObjectiveValue = get<1>(neighborTuple);
            int swappedValue1 = get<2>(neighborTuple);
            int swappedValue2 = get<3>(neighborTuple);

            if (neighborObjectiveValue > currentObjectiveValue) {
                currentState = neighbor;
                currentObjectiveValue = neighborObjectiveValue;
                foundBetter = true;
                sidewaysMoves = 0;
                addNextStep(currentState, swappedValue1, swappedValue2);
                break;
            } else if (neighborObjectiveValue == currentObjectiveValue && sidewaysMoves <
            sidewaysMoveLimit) {
                currentState = neighbor;
                foundBetter = true;
                sidewaysMoves++;
                addNextStep(currentState, swappedValue1, swappedValue2);
                break;
            }
        }

        if (!foundBetter || sidewaysMoves >= sidewaysMoveLimit) {
            break;
        }
    }

    finalState = currentState;
    if (finalState.objectiveFunction() == 0) {
        found = true;
    }

    auto end = high_resolution_clock::now();
    timeTaken = duration_cast<duration<double>>(end - start).count();
}
```

4. *Hill Climbing with Random Restart*

Deskripsi Algoritma	Class Algoritma Hill Climbing Random Restart yang menginherit Class LocalSearch

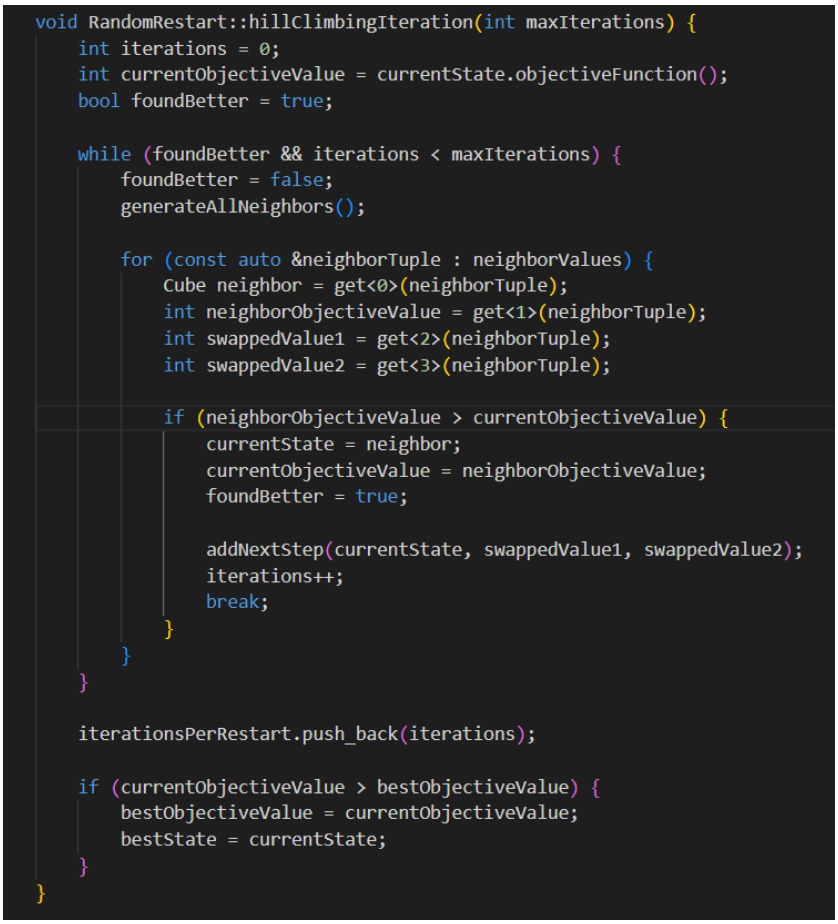
Nama Fungsi	RandomRestart(Cube init, int maxRestarts, int maxIterationsPerRestart);
Deskripsi Fungsi	Fungsi konstruktor untuk Random Restart

Source Code	`RandomRestart::RandomRestart(Cube init, int maxRestarts, int maxIterationsPerRestart)`
	`: LocalSearch(init), maxRestarts(maxRestarts), maxIterationsPerRestart(maxIterationsPerRestart) {`
	`bestObjectiveValue = INT_MIN;`
	`}`

Nama Fungsi	int getTotalRestarts() const
Deskripsi Fungsi	Fungsi untuk mendapatkan total restart
Source Code	`int getTotalRestarts() const { return maxRestarts; }`

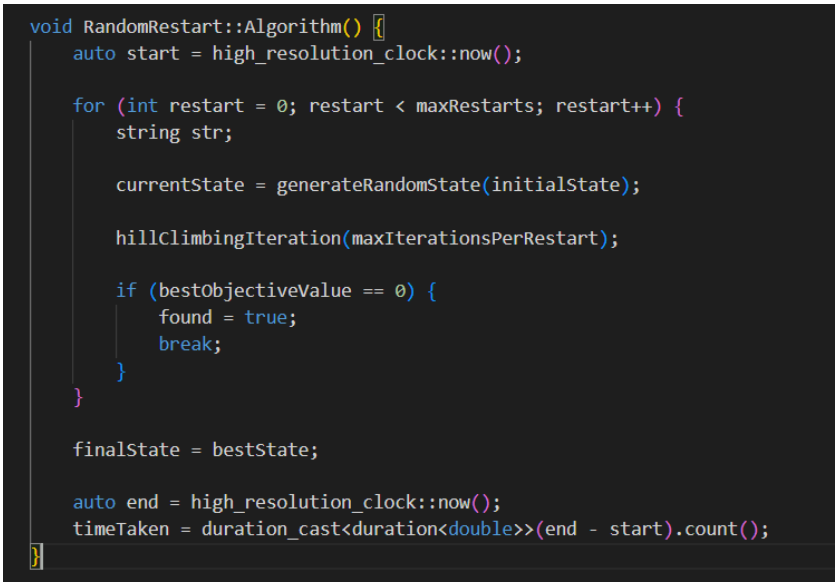
Nama Fungsi	vector<int> getIterationsPerRestart() cons
Deskripsi Fungsi	Fungsi untuk mendapatkan berapa iterasi yang diperlukan sampai mencapai lokal optima per restart
Source Code	`std::vector<int> getIterationsPerRestart() const { return iterationsPerRestart; }`

Nama Fungsi	void hillClimbingIteration(int maxIterations);
Deskripsi Fungsi	Prosedur untuk melakukan iterasi dari steepest Ascent Hill Climbing sebanyak max Iteration

Source Code	

```
void RandomRestart::hillClimbingIteration(int maxIterations) {
    int iterations = 0;
    int currentObjectiveValue = currentState.objectiveFunction();
    bool foundBetter = true;

    while (foundBetter && iterations < maxIterations) {
        foundBetter = false;
        generateAllNeighbors();

        for (const auto &neighborTuple : neighborValues) {
            Cube neighbor = get<0>(neighborTuple);
            int neighborObjectiveValue = get<1>(neighborTuple);
            int swappedValue1 = get<2>(neighborTuple);
            int swappedValue2 = get<3>(neighborTuple);

            if (neighborObjectiveValue > currentObjectiveValue) {
                currentState = neighbor;
                currentObjectiveValue = neighborObjectiveValue;
                foundBetter = true;

                addNextStep(currentState, swappedValue1, swappedValue2);
                iterations++;
                break;
            }
        }
    }

    iterationsPerRestart.push_back(iterations);

    if (currentObjectiveValue > bestObjectiveValue) {
        bestObjectiveValue = currentObjectiveValue;
        bestState = currentState;
    }
}
```

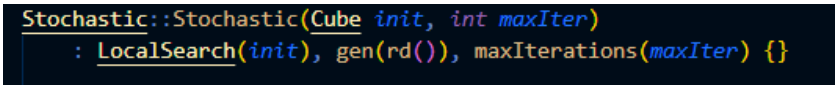
Nama Fungsi	void algorithm()
Deskripsi Fungsi	Fungsi untuk melakukan algoritma Random Restart dari state awal ke state akhir

Source Code	

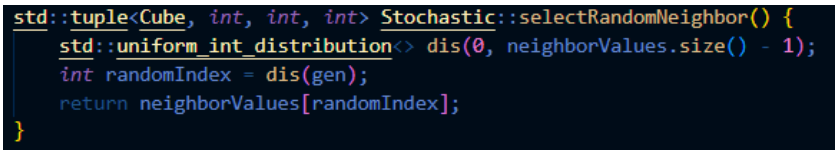
```
void RandomRestart::Algorithm() {
    auto start = high_resolution_clock::now();

    for (int restart = 0; restart < maxRestarts; restart++) {
        string str;

        currentState = generateRandomState(initialState);

        hillClimbingIteration(maxIterationsPerRestart);

        if (bestObjectiveValue == 0) {
            found = true;
            break;
        }
    }

    finalState = bestState;

    auto end = high_resolution_clock::now();
    timeTaken = duration_cast<duration<double>>(end - start).count();
}
```

5. *Stochastic Hill Climbing*

Deskripsi Algoritma	Class Algoritma Stochastic Hill Climbing yang menginherit Class LocalSearch

Nama Fungsi	Stochastic(Cube init, int maxIter)
Deskripsi Fungsi	Fungsi konstruktor untuk Random Restart
Source Code	

```
Stochastic::Stochastic(Cube init, int maxIter)
    : LocalSearch(init), gen(rd()), maxIterations(maxIter) {}
```

Nama Fungsi	std::tuple<Cube, int, int, int> Stochastic::selectRandomNeighbor()
Deskripsi Fungsi	Fungsi untuk memanggil neighbor secara random
Source Code	

```
std::tuple<Cube, int, int, int> Stochastic::selectRandomNeighbor() {
    std::uniform_int_distribution<> dis(0, neighborValues.size() - 1);
    int randomIndex = dis(gen);
    return neighborValues[randomIndex];
}
```

Nama Fungsi	void Stochastic::Algorithm()
Deskripsi Fungsi	Fungsi yang berisi algoritma Stochastic mulai dari state awal hingga state final
Source Code	

```
void Stochastic::Algorithm() {
    auto start = high_resolution_clock::now();

    currentState = initialState;
    int currentObjectiveValue = currentState.objectiveFunction();
    int iterations = 0;
    bool improved = true;

    while (improved && iterations < maxIterations) {
        improved = false;
        generateAllNeighbors();

        int k = 10;
        for (int i = 0; i < k; i++) {
            auto neighborTuple = selectRandomNeighbor();
            Cube neighbor = get<0>(neighborTuple);
            int neighborObjectiveValue = get<1>(neighborTuple);
            int swappedValue1 = get<2>(neighborTuple);
            int swappedValue2 = get<3>(neighborTuple);

            if (neighborObjectiveValue > currentObjectiveValue) {
                currentState = neighbor;
                currentObjectiveValue = neighborObjectiveValue;
                improved = true;
                iterations++;

                addNextStep(currentState, swappedValue1, swappedValue2);
                break;
            }
        }

        if (currentObjectiveValue == 0) {
            found = true;
            break;
        }
    }

    finalState = currentState;

    auto end = high_resolution_clock::now();
    timeTaken = duration_cast<duration<double>>(end - start).count();
}
```

6. *Simulated Annealing*

Deskripsi Algoritma	Class Algoritma Simulated Annealing yang menginherit Class LocalSearch

Nama Fungsi	SimulatedAnnealing(Cube init, double init_temp, double cutoff, double coolingRate, double minTemp, int maxIter, int

	totalLocalOptCount)
Deskripsi Fungsi	Fungsi Konstruktor dari Simulated Annealing
Source Code	`SimulatedAnnealing::SimulatedAnnealing(Cube init, double init_temp, double cutoff, double coolingRate, double minTemp, int maxIter, int totalLocalOptCount) : LocalSearch(init), init_temp(init_temp), cutoff(cutoff), coolingRate(coolingRate), minTemp(minTemp), maxIter(maxIter), totalLocalOptCount(totalLocalOptCount) {}`

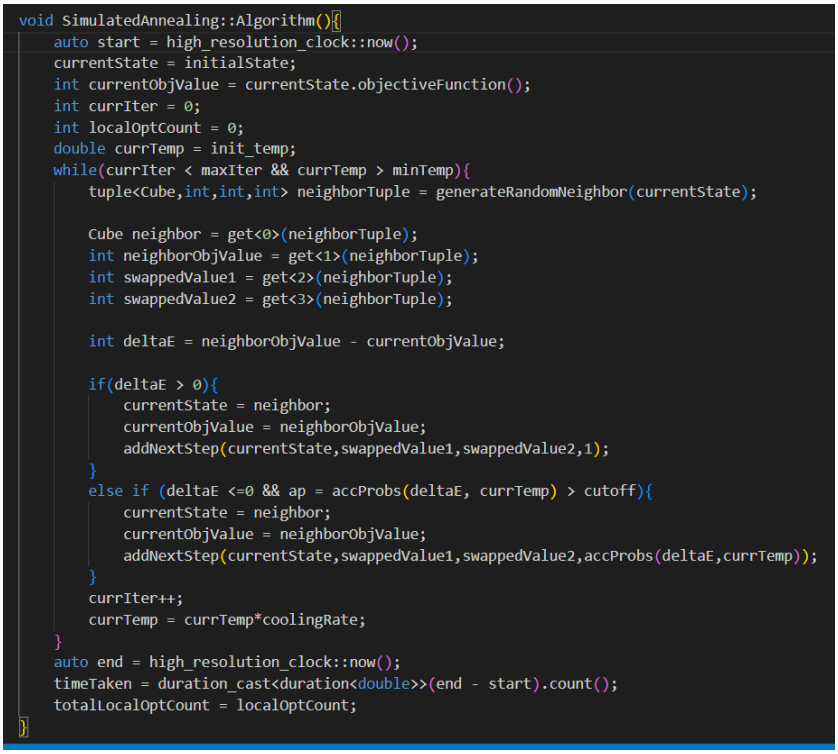
Nama Fungsi	double accProbs(int deltaE,double currTemp);
Deskripsi Fungsi	Fungsi untuk menghitung nilai Acceptability Probability untuk suatu deltaE dan currTemp.

Source Code	`double SimulatedAnnealing::accProbs(int deltaE,double currTemp){`
	`  return exp(-deltaE/currTemp);`
	`}`

Nama Fungsi	void addNextStep(Cube nextState, int value1, int value2, double value3) override;
Deskripsi Fungsi	Fungsi untukoverride fungsi addNextStep dan menambah value3, yakni Acceptability Probability pada next step.

Source Code	`void SimulatedAnnealing::addNextStep(Cube nextState, int value1, int value2, int value3)`
	`{`
	`  allStep.emplace_back(nextState, value1, value2,value3);`
	`  totalStep++;`
	`}`

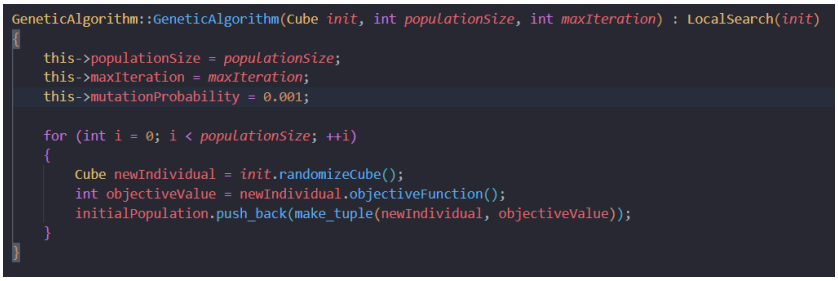
Nama Fungsi	void Algorithm()
Deskripsi Fungsi	Fungsi yang berisi algoritma Simulated Annealing mulai dari state awal hingga state final

Source Code	

```
void SimulatedAnnealing::Algorithm(){
    auto start = high_resolution_clock::now();
    currentState = initialState;
    int currentObjValue = currentState.objectiveFunction();
    int currIter = 0;
    int localOptCount = 0;
    double currTemp = init_temp;
    while(currIter < maxIter && currTemp > minTemp){
        tuple<Cube,int,int,int> neighborTuple = generateRandomNeighbor(currentState);

        Cube neighbor = get<0>(neighborTuple);
        int neighborObjValue = get<1>(neighborTuple);
        int swappedValue1 = get<2>(neighborTuple);
        int swappedValue2 = get<3>(neighborTuple);

        int deltaE = neighborObjValue - currentObjValue;

        if(deltaE > 0){
            currentState = neighbor;
            currentObjValue = neighborObjValue;
            addNextStep(currentState,swappedValue1,swappedValue2,1);
        }
        else if (deltaE <=0 && ap = accProbs(deltaE, currTemp) > cutoff){
            currentState = neighbor;
            currentObjValue = neighborObjValue;
            addNextStep(currentState,swappedValue1,swappedValue2,accProbs(deltaE,currTemp));
        }
        currIter++;
        currTemp = currTemp*coolingRate;
    }
    auto end = high_resolution_clock::now();
    timeTaken = duration_cast<duration<double>>(end - start).count();
    totalLocalOptCount = localOptCount;
}
```

7. *Genetic Algorithm*

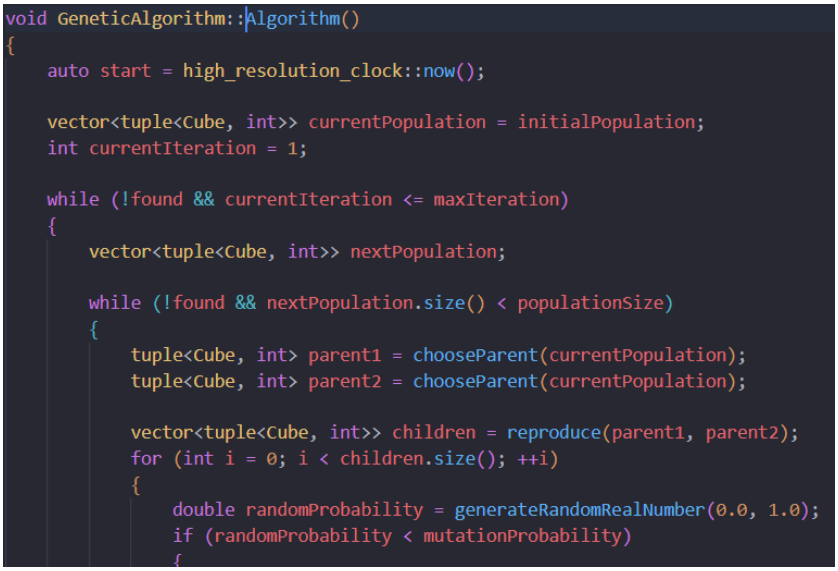
Deskripsi Algoritma	Class Algoritma Genetic Algorithm yang menginherit Class LocalSearch

Nama Fungsi	GeneticAlgorithm(Cube init, int populationSize, int maxIteration);
Deskripsi Fungsi	Fungsi konstruktor dari kelas Genetic Algorithm. Genetic Algorithm menerima populationSize dan maxIteration sebagai parameter yang nilainya berdasarkan input user.
Source Code	

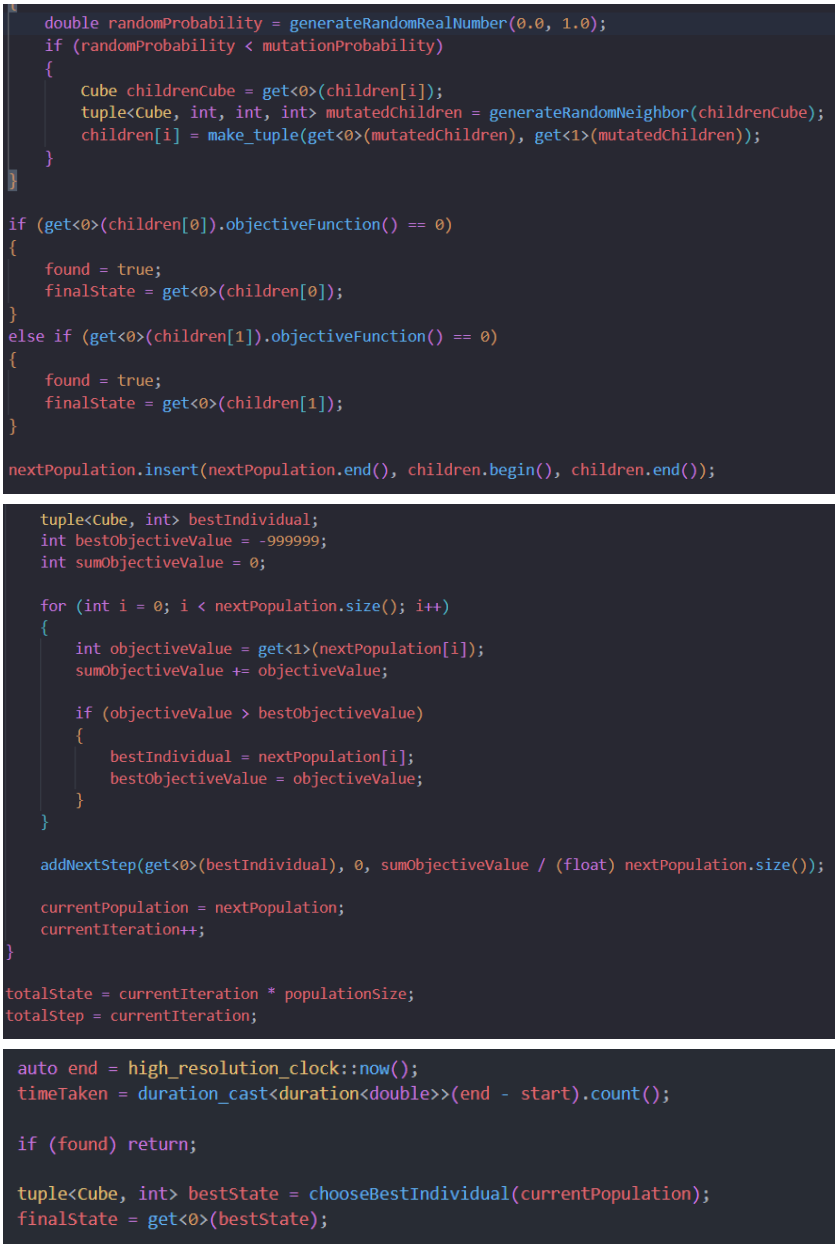
```
GeneticAlgorithm::GeneticAlgorithm(Cube init, int populationSize, int maxIteration) : LocalSearch(init)
{
    this->populationSize = populationSize;
    this->maxIteration = maxIteration;
    this->mutationProbability = 0.001;

    for (int i = 0; i < populationSize; ++i)
    {
        Cube newIndividual = init.randomizeCube();
        int objectiveValue = newIndividual.objectiveFunction();
        initialPopulation.push_back(make_tuple(newIndividual, objectiveValue));
    }
}
```

Nama Fungsi	tuple<Cube, int> chooseParent(vector<tuple<Cube, int>> &currentPopulation);
Deskripsi Fungsi	Fungsi untuk memilih satu parent yang akan digunakan untuk crossover. Fungsi ini menggunakan nilai dari fitnessFunction() dan mekanisme probability wheel untuk memilih parent.
Source Code	

```
vector<tuple<Cube, int>> GeneticAlgorithm::reproduce(tuple<Cube, int> &parent1, tuple<Cube, int> &parent2)
{
    Cube parentCube1 = get<0>(parent1);
    Cube parentCube2 = get<0>(parent2);

    int n = initialState.getSize();
    vector<Position> position1(pow(n, 3));
    vector<Position> position2(pow(n, 3));
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            for (int k = 0; k < n; ++k)
            {
                position1[parentCube1.getValue(i, j, k) - 1] = Position(i, j, k);
                position2[parentCube2.getValue(i, j, k) - 1] = Position(i, j, k);
            }
        }
    }

    vector<Position> positionChild1(pow(n, 3));
    vector<Position> positionChild2(pow(n, 3));
```

Nama Fungsi	void Algorithm();
Deskripsi Fungsi	Fungsi untuk menjalankan algoritma Genetic Algorithm untuk memulai pencarian solusi.
Source Code	

```
void GeneticAlgorithm::Algorithm()
{
    auto start = high_resolution_clock::now();

    vector<tuple<Cube, int>> currentPopulation = initialPopulation;
    int currentIteration = 1;

    while (!found && currentIteration <= maxIteration)
    {
        vector<tuple<Cube, int>> nextPopulation;

        while (!found && nextPopulation.size() < populationSize)
        {
            tuple<Cube, int> parent1 = chooseParent(currentPopulation);
            tuple<Cube, int> parent2 = chooseParent(currentPopulation);

            vector<tuple<Cube, int>> children = reproduce(parent1, parent2);
            for (int i = 0; i < children.size(); ++i)
            {
                double randomProbability = generateRandomRealNumber(0.0, 1.0);
                if (randomProbability < mutationProbability)
                {
```

```
    double randomProbability = generateRandomRealNumber(0.0, 1.0);
    if (randomProbability < mutationProbability)
    {
        Cube childrenCube = get<0>(children[i]);
        tuple<Cube, int, int, int> mutatedChildren = generateRandomNeighbor(childrenCube);
        children[i] = make_tuple(get<0>(mutatedChildren), get<1>(mutatedChildren));
    }
}

if (get<0>(children[0]).objectiveFunction() == 0)
{
    found = true;
    finalState = get<0>(children[0]);
}
else if (get<0>(children[1]).objectiveFunction() == 0)
{
    found = true;
    finalState = get<0>(children[1]);
}

nextPopulation.insert(nextPopulation.end(), children.begin(), children.end());
```

```
    tuple<Cube, int> bestIndividual;
    int bestObjectiveValue = -999999;
    int sumObjectiveValue = 0;

    for (int i = 0; i < nextPopulation.size(); i++)
    {
        int objectiveValue = get<1>(nextPopulation[i]);
        sumObjectiveValue += objectiveValue;

        if (objectiveValue > bestObjectiveValue)
        {
            bestIndividual = nextPopulation[i];
            bestObjectiveValue = objectiveValue;
        }
    }

    addNextStep(get<0>(bestIndividual), 0, sumObjectiveValue / (float) nextPopulation.size());

    currentPopulation = nextPopulation;
    currentIteration++;
}

totalState = currentIteration * populationSize;
totalStep = currentIteration;
```

```
auto end = high_resolution_clock::now();
timeTaken = duration_cast<duration<double>>(end - start).count();

if (found) return;

tuple<Cube, int> bestState = chooseBestIndividual(currentPopulation);
finalState = get<0>(bestState);
```

Nama Fungsi	vector<tuple<Cube, int>> reproduce(tuple<Cube, int> &parent1, tuple<Cube, int> &parent2);
Deskripsi Fungsi	Fungsi untuk melakukan crossover, yaitu menghasilkan dua individu anak dari dua parent state.

Source Code

```
vector<tuple<Cube, int>> GeneticAlgorithm::reproduce(tuple<Cube, int> &parent1, tuple<Cube, int> &parent2)
{
    Cube parentCube1 = get<0>(parent1);
    Cube parentCube2 = get<0>(parent2);

    int n = initialState.getSize();
    vector<Position> position1(pow(n, 3));
    vector<Position> position2(pow(n, 3));
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            for (int k = 0; k < n; ++k)
            {
                position1[parentCube1.getValue(i, j, k) - 1] = Position(i, j, k);
                position2[parentCube2.getValue(i, j, k) - 1] = Position(i, j, k);
            }
        }
    }

    vector<Position> positionChild1(pow(n, 3));
    vector<Position> positionChild2(pow(n, 3));
```

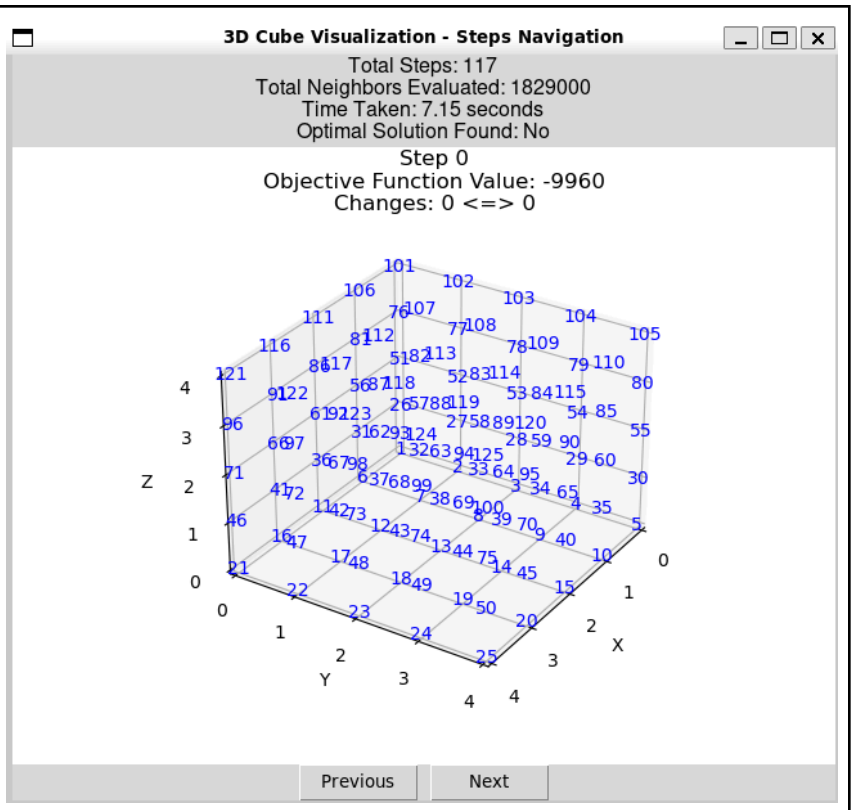
```
vector<Position> positionChild1(pow(n, 3));
vector<Position> positionChild2(pow(n, 3));

int halfPosSize = pow(n, 3) / 2;

for (int i = 0; i < halfPosSize; ++i)
{
    positionChild1[i] = position1[i];
    positionChild2[i] = position2[i];
}

vector<int> conflictingValues1;
vector<int> conflictingValues2;

for (int i = halfPosSize; i < pow(n, 3); ++i)
{
    bool isPositionConflict1 = false;
    bool isPositionConflict2 = false;
    int conflictIdx1 = -1;
    int conflictIdx2 = -1;

    for (int j = 0; j < halfPosSize; ++j)
    {
        // Child 1
        if (position1[j] == position2[i])
        {
            isPositionConflict1 = true;
            conflictIdx1 = j;
        }

        // Child 2
        if (position2[j] == position1[i])
        {
            isPositionConflict2 = true;
            conflictIdx2 = j;
        }
```

```
}

// Child 1
if (!isPositionConflict1)
{
    positionChild1[i] = position2[i];
}
else
{
    double randomProbability = generateRandomRealNumber(0.0, 1.0);
    if (randomProbability < 0.5)
    {
        conflictingValues1.push_back(i+1);
    }
    else
    {
        positionChild1[conflictIdx1] = Position();
        positionChild1[i] = position2[i];
        conflictingValues1.push_back(conflictIdx1 + 1);
    }
}

// Child 2
if (!isPositionConflict2)
{
    positionChild2[i] = position1[i];
}
else
{
    double randomProbability = generateRandomRealNumber(0.0, 1.0);
    if (randomProbability < 0.5)
    {
        conflictingValues2.push_back(i+1);
    }
```

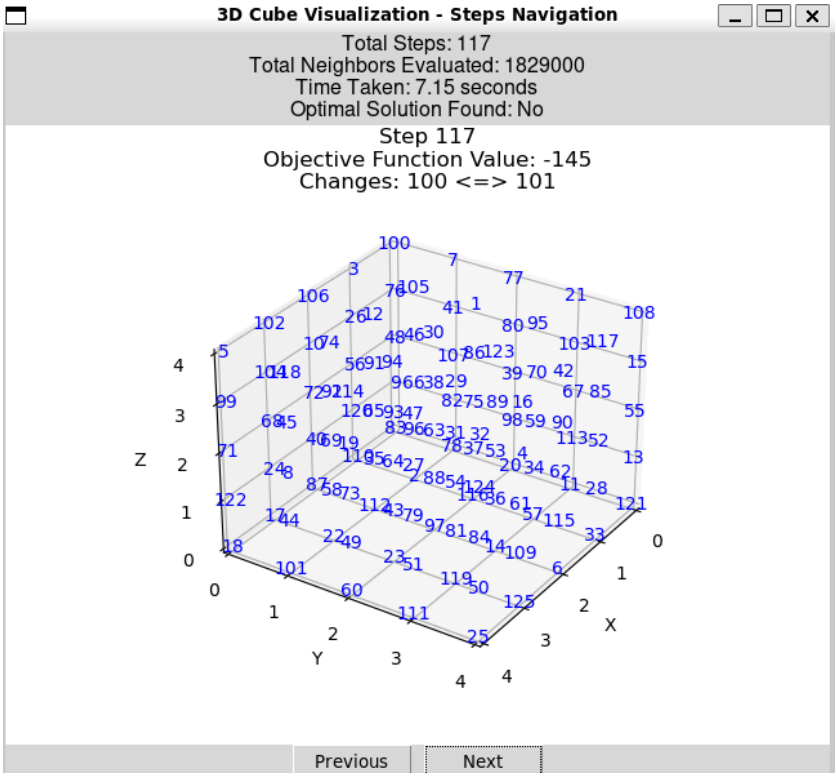
C. Hasil Eksperimen

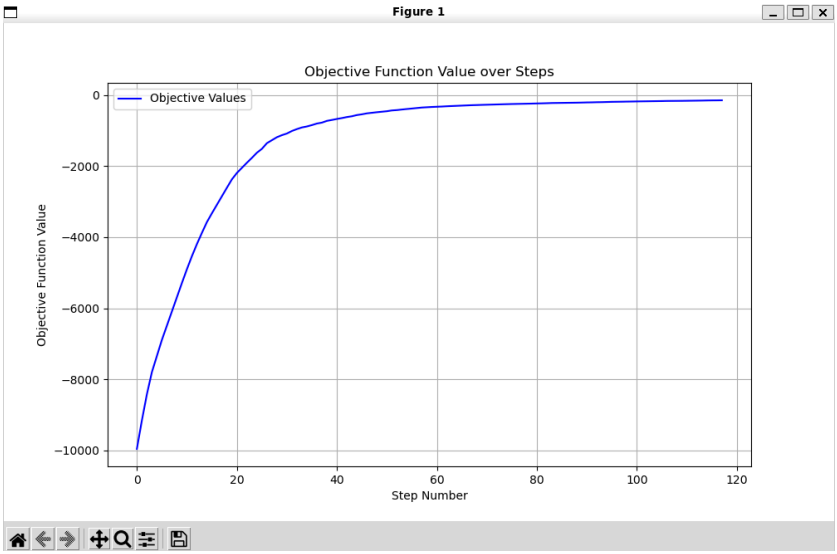
1. *Steepest Ascent Hill Climbing*

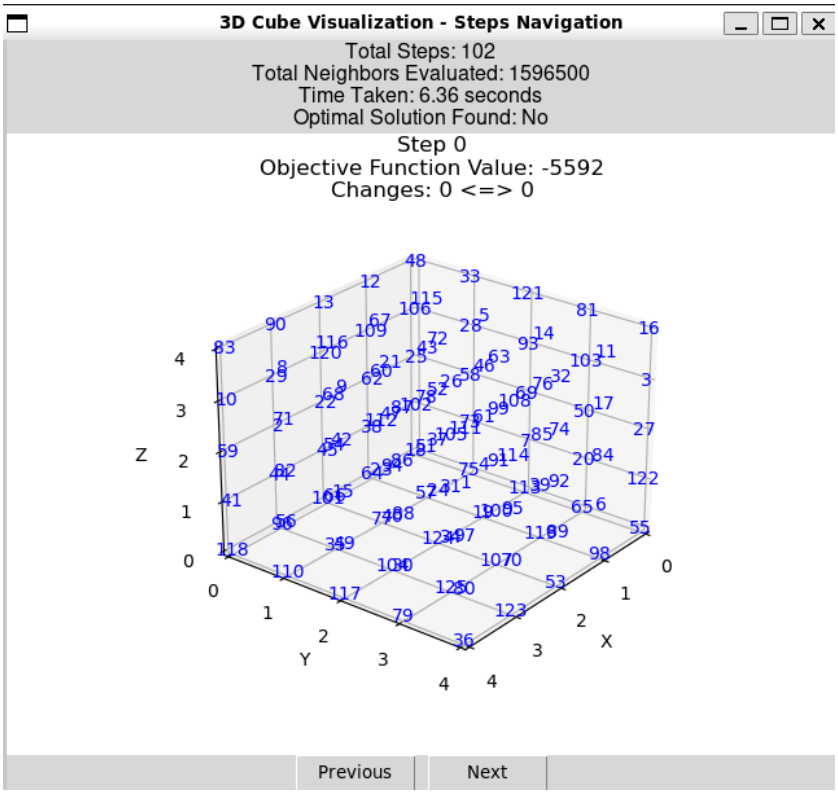
Cube Awal

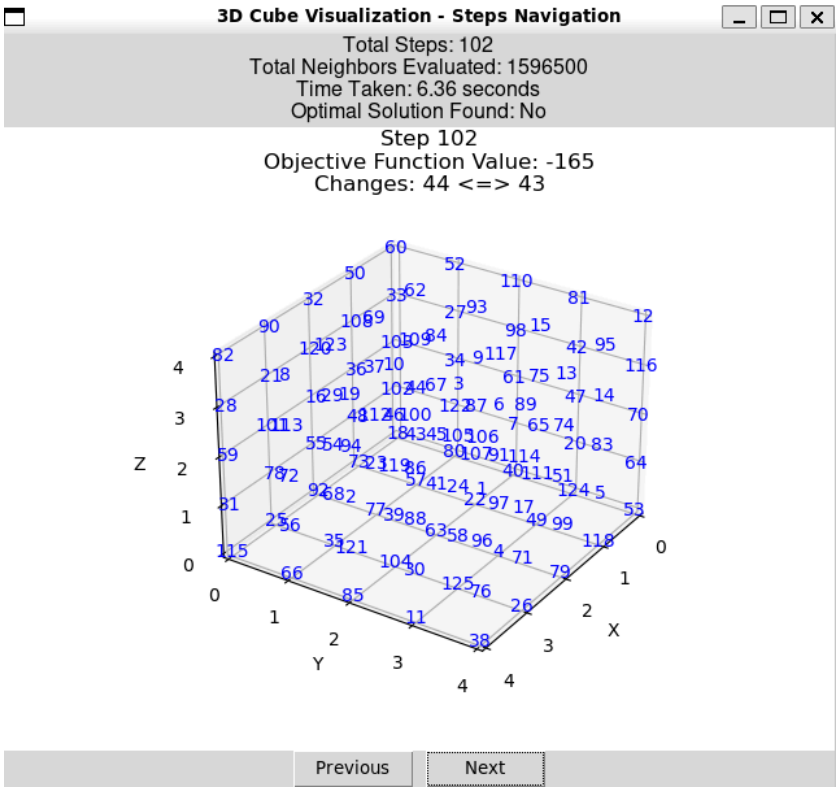


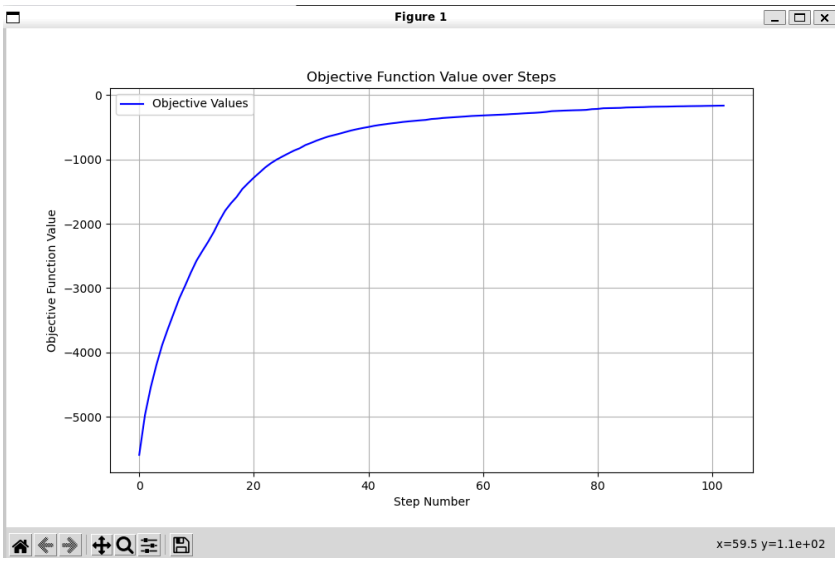
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
26 27 28 29 30
31 32 33 34 35
36 37 38 39 40
41 42 43 44 45
46 47 48 49 50
51 52 53 54 55
56 57 58 59 60
61 62 63 64 65
66 67 68 69 70
71 72 73 74 75
76 77 78 79 80
81 82 83 84 85
86 87 88 89 90
91 92 93 94 95
96 97 98 99 100
101 102 103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

	121 122 123 124 125
State Value	-9960
Cube Akhir	 83 78 20 11 121 110 2 116 57 33 87 112 97 14 6 17 22 23 119 125 18 101 60 111 25 9 82 98 113 13 120 96 37 34 28 40 35 88 36 115 24 58 43 81 109 122 44 49 51 50 48 107 39 67 55 56 66 75 59 52 72 65 63 53 62 68 69 64 54 61 71 8 73 79 84 76 41 80 103 15 26 46 86 70 85 10 91 38 89 90 104 92 93 31 4 99 45 19 27 124

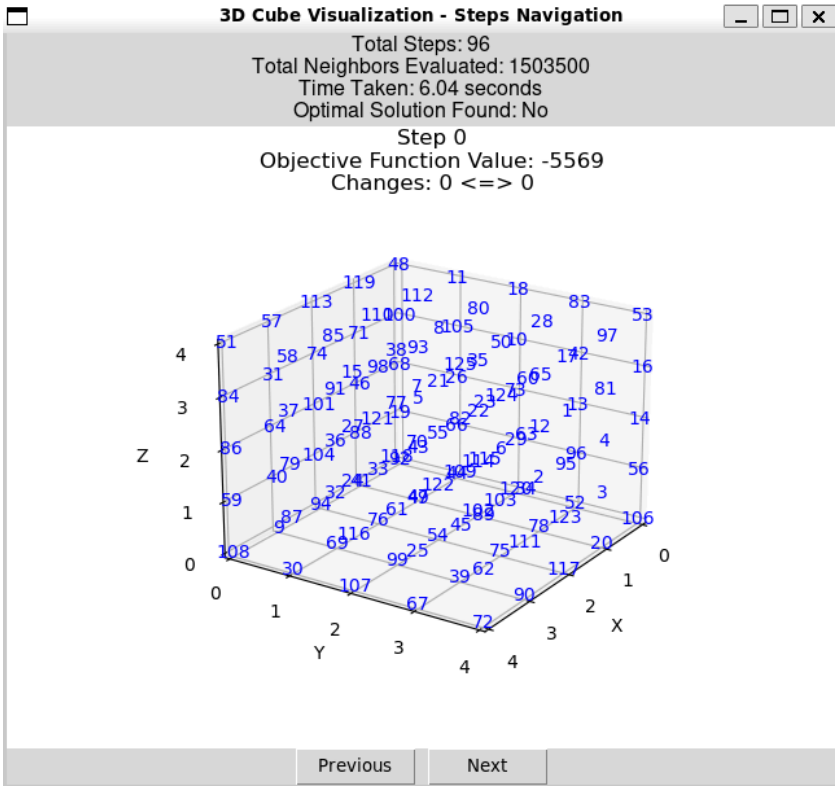
	100 7 77 21 108 3 105 1 95 117 106 12 30 123 42 102 74 94 29 16 5 118 114 47 32
State Value	-145
Graf Plot Objective Function	
Waktu	7.15 s

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

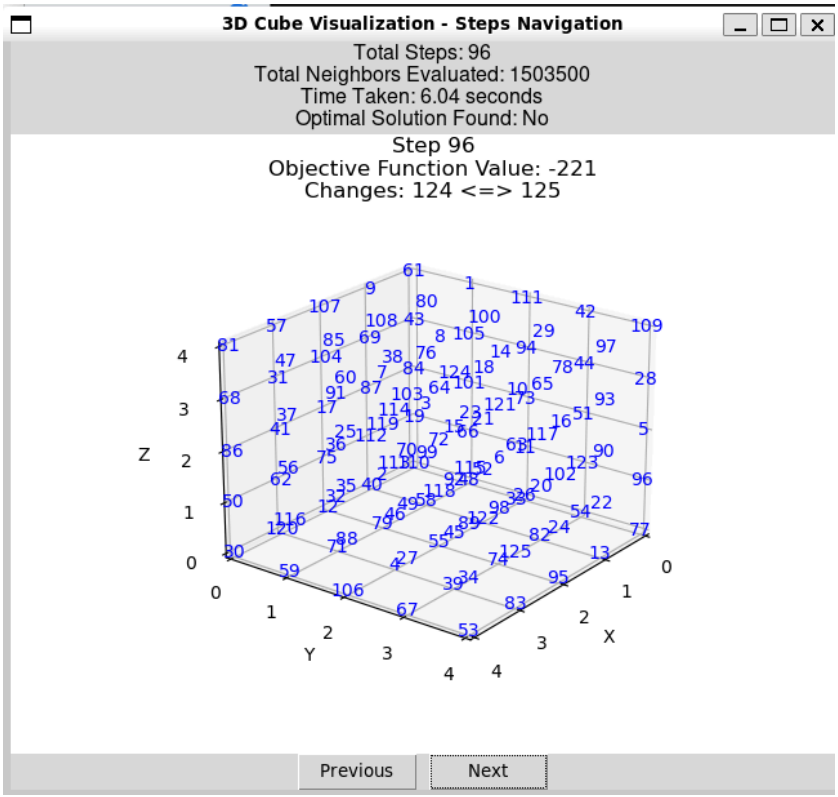
	83 8 9 87 111
State Value	-5592
Cube Akhir	 18 80 40 124 53 73 57 22 49 118 92 77 63 4 79 25 35 104 125 26 115 66 85 11 38 102 122 7 20 64 48 43 107 111 5 55 23 41 97 99 78 68 39 58 71 31 56 121 30 76 103 34 61 47 70 36 44 87 65 83 16 112 45 91 51 101 54 119 24 17 59 72 2 88 96 33 27 98 42 116 108 109 9 75 14 120 37 67 6 74 21 29 46 105 114

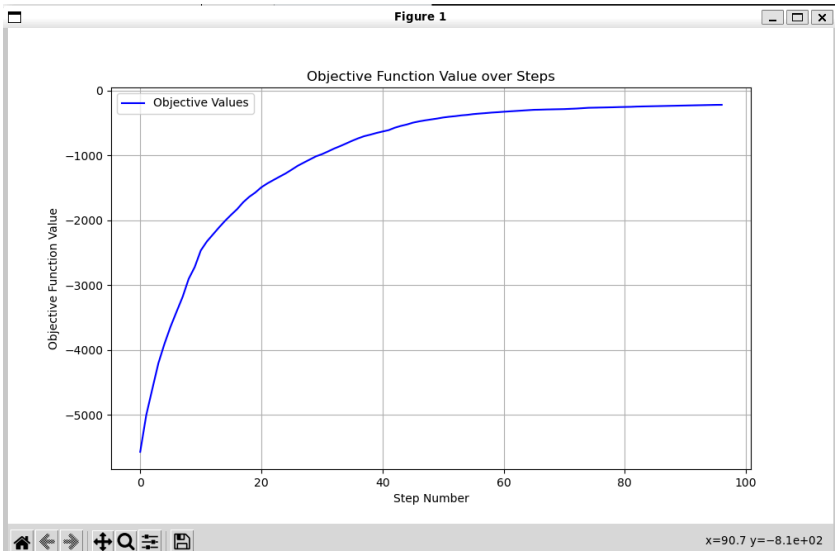
	28 113 94 86 1 60 52 110 81 12 50 62 93 15 95 32 69 84 117 13 90 123 10 3 89 82 8 19 100 106
State Value	-165
Graf Plot Objective Function	
Waktu	6.36 s

Cube Awal

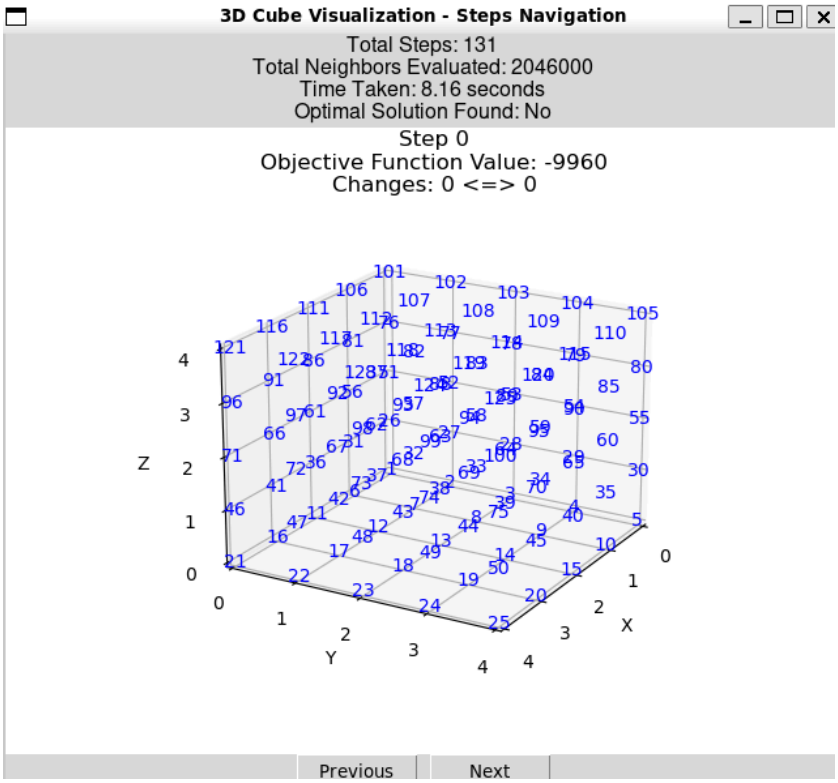


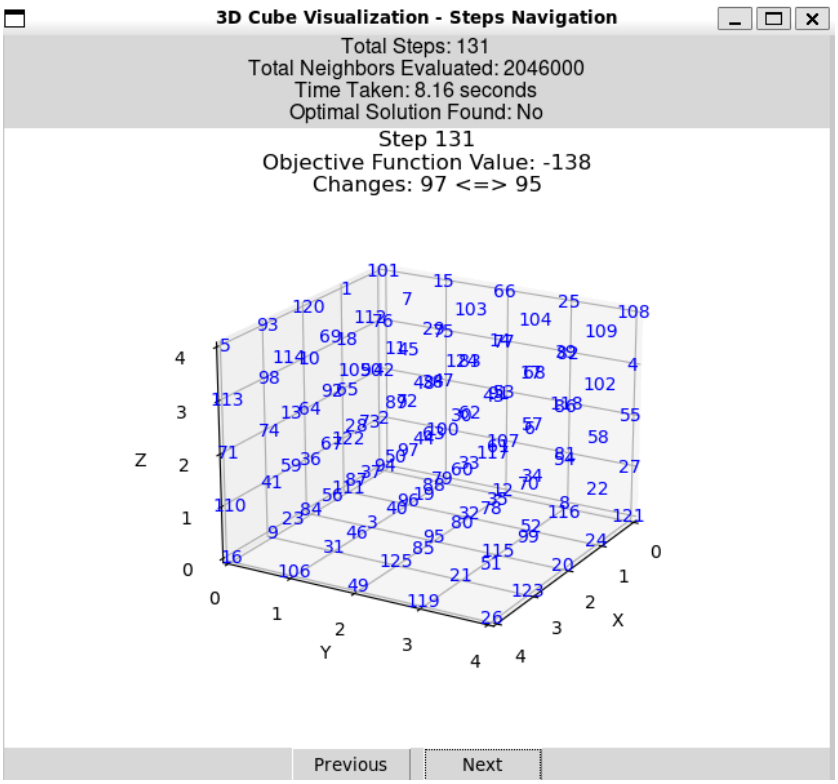
92 44 120 52 106
41 47 102 78 20
94 76 54 75 117
9 69 99 39 90
108 30 107 67 72
19 66 29 96 56
88 43 114 2 3
104 33 122 103 123
40 32 61 45 111
59 87 116 25 62
68 26 73 13 14
46 5 22 12 4
101 121 55 6 95
64 36 118 109 34
86 79 24 49 89
100 105 10 42 16
71 93 35 65 81
74 98 21 124 1
31 91 77 82 63
84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

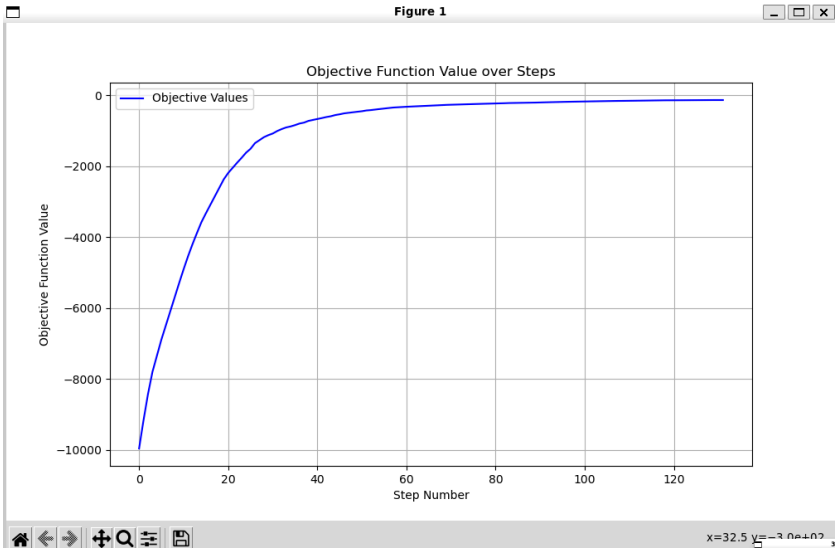
	51 58 15 7 23
State Value	-5569
Cube Akhir	 110 48 26 54 77 40 58 122 82 13 12 79 55 74 95 120 71 4 39 83 30 59 106 67 53 19 66 11 123 96 112 99 52 20 22 75 2 118 98 24 62 32 46 45 125 50 116 88 27 34 84 101 73 51 5 87 3 21 117 90 17 119 72 6 102 41 36 113 92 33 86 56 35 49 89 43 105 94 44 28 69 76 18 65 93 104 7 64 121 16

	31 91 114 15 63 68 37 25 70 115 61 1 111 42 109 9 80 100 29 97 107 108 8 14 78 57 85 38 124 10 81 47 60 103 23
State Value	-221
Graf Plot Objective Function	
Waktu	6.36 s

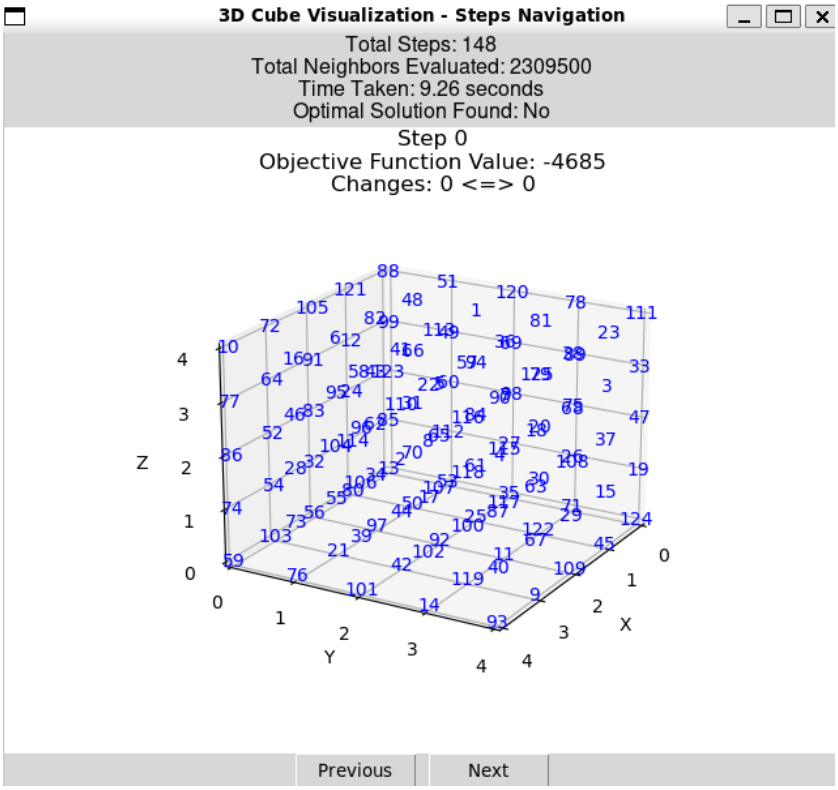
2. *Hill Climbing with Sideways Move*

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

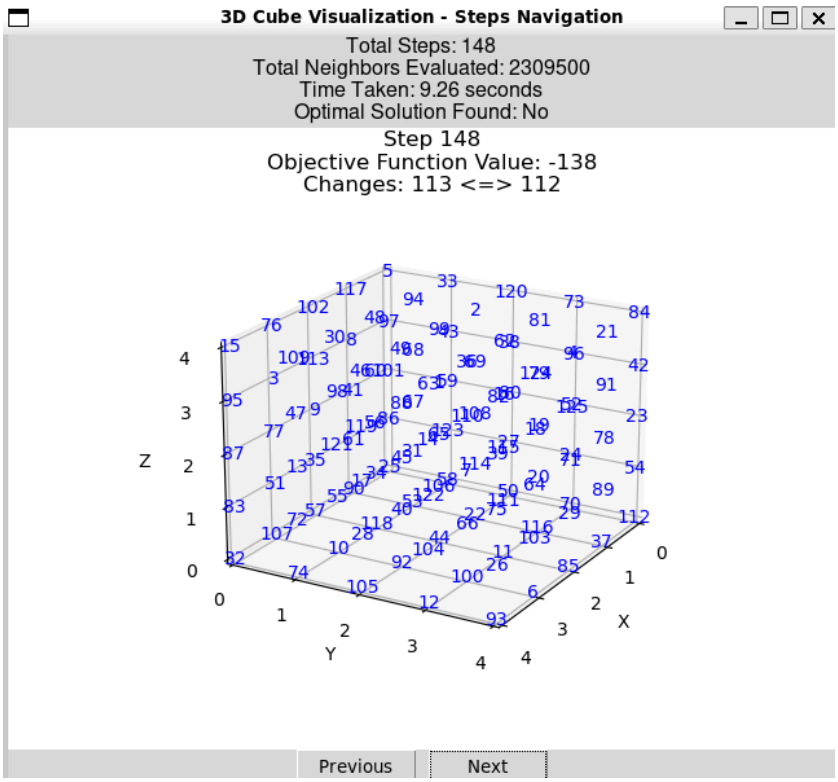
	121 122 123 124 125
State Value	-9960
Cube Akhir	 94 79 12 8 121 111 96 32 52 24 84 3 95 115 20 9 31 125 21 123 16 106 49 119 26 2 100 107 81 27 122 97 33 34 22 36 37 88 35 116 41 56 40 80 99 110 23 46 85 51 42 47 53 118 55 65 72 62 57 58 64 73 63 61 54 74 67 50 60 70 71 59 87 19 78 76 75 77 82 4 18 45 83 68 102 10 90 38 91 86 98 92 89 30 6

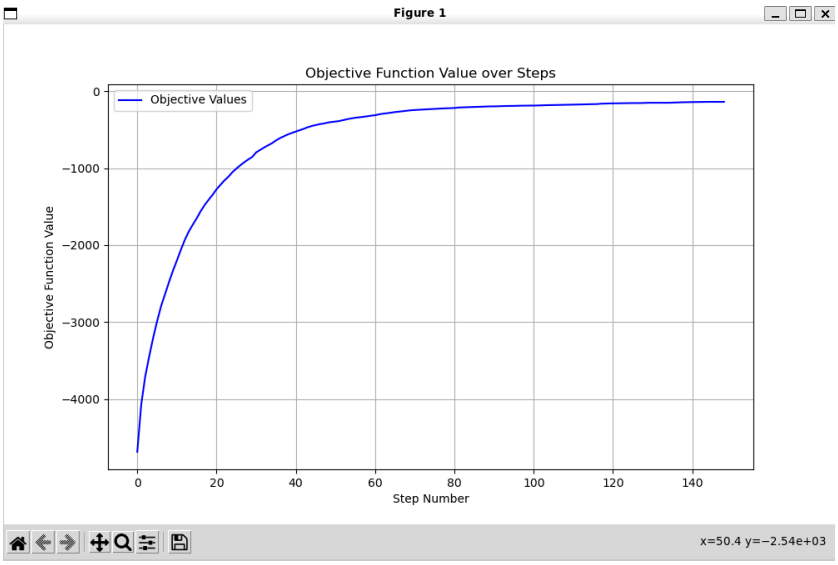
	113 13 28 44 117 101 15 66 25 108 1 7 103 104 109 120 112 29 14 39 93 69 11 124 17 5 114 105 48 43
State Value	-165
Graf Plot Objective Function	
Waktu	8.16 s s

Cube Awal

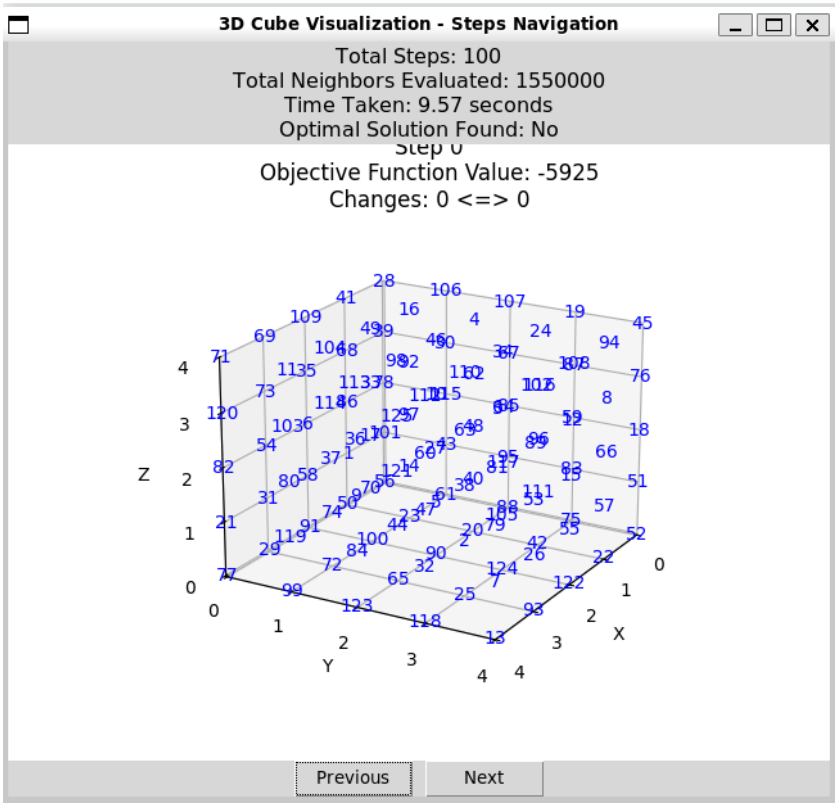


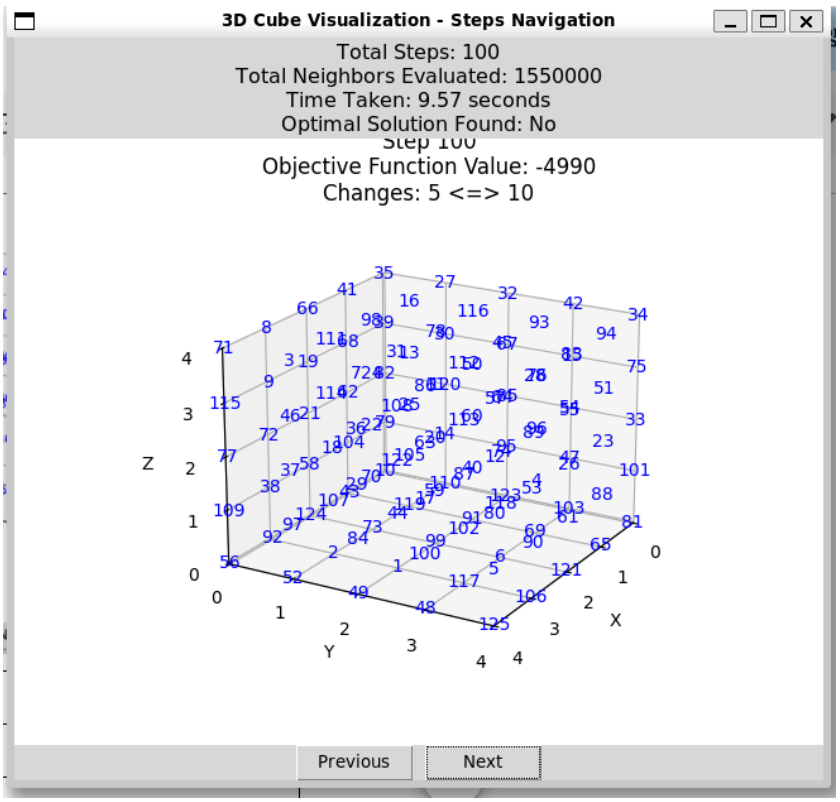
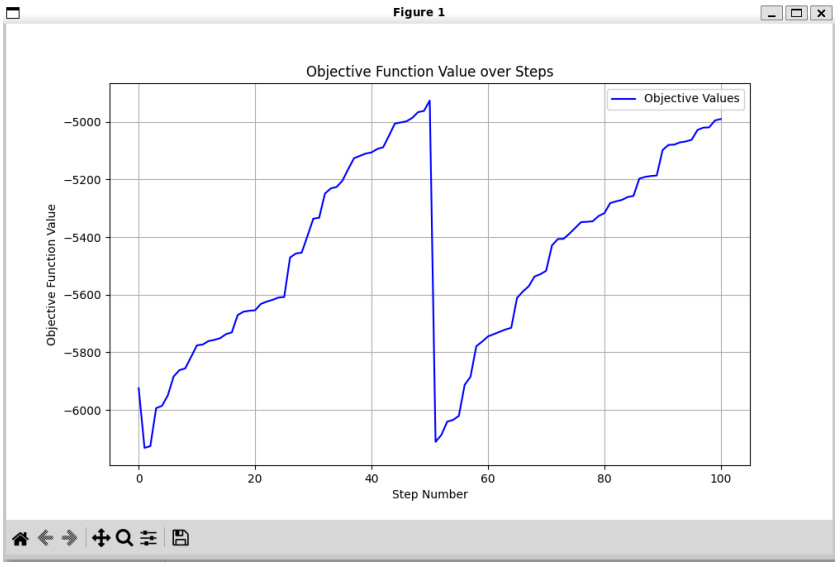
13 53 35 71 124
80 50 25 122 45
56 97 92 11 109
103 21 42 119 9
59 76 101 14 93
85 112 27 26 19
114 70 61 30 15
32 34 107 117 29
54 55 44 100 67
74 73 39 102 40
123 60 98 75 47
24 31 84 20 37
83 62 65 115 108
52 104 2 118 63
86 28 106 17 87
99 49 69 89 33
12 66 94 79 3
91 43 5 7 68
64 95 110 116 18
77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

	10 16 58 22 90
State Value	-4685
Cube Akhir	 25 58 50 70 112 90 53 22 116 37 57 118 44 11 85 107 10 92 100 6 32 74 105 12 93 86 123 27 24 54 61 31 114 20 89 35 34 106 111 29 51 55 40 66 103 83 72 28 104 26 101 59 80 52 23 41 67 108 19 78 9 56 65 115 71 77 121 45 7 64 87 13 17 122 75 97 43 38 96 42 8 68 69 79 91 113 60 1 16 125

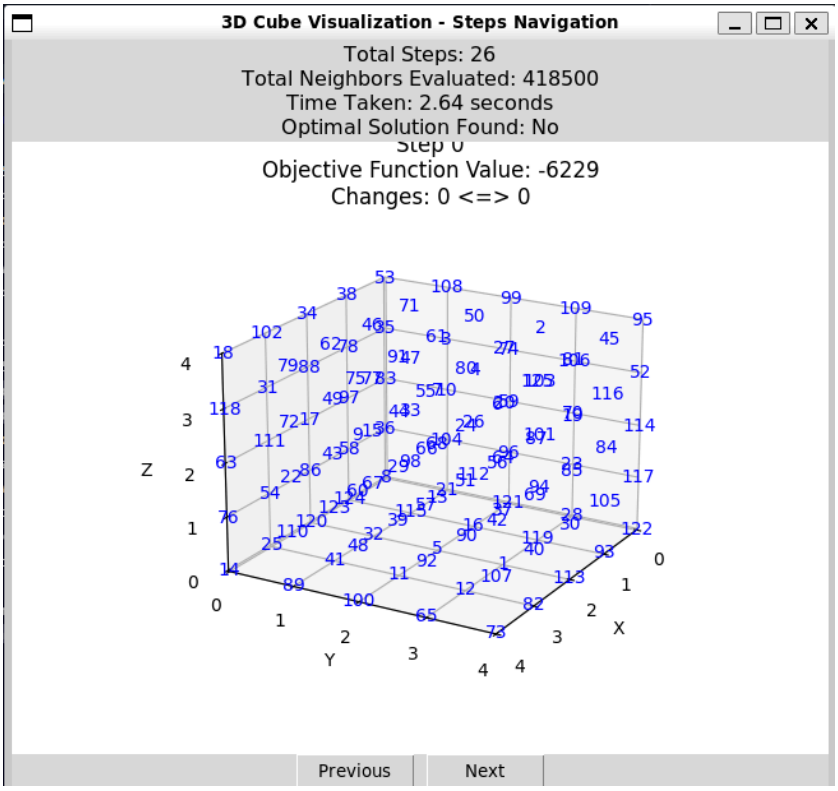
	3 98 88 110 18 95 47 119 14 39 5 33 120 73 84 117 94 2 81 21 102 48 99 62 4 76 30 49 36 124 15 109 46 63 82
State Value	-138
Graf Plot Objective Function	
Waktu	9.26 s s

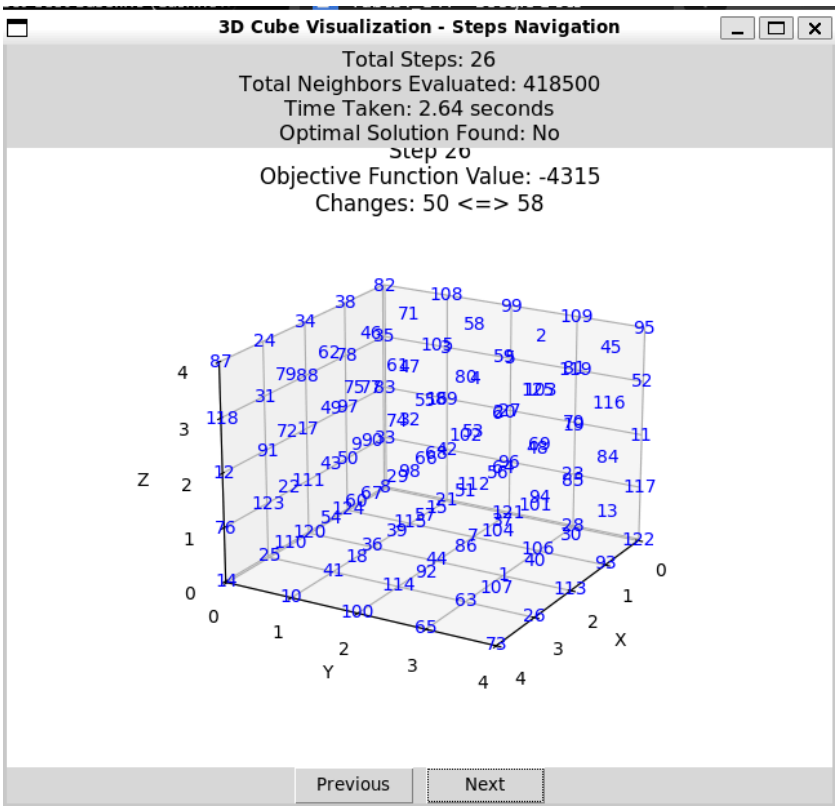
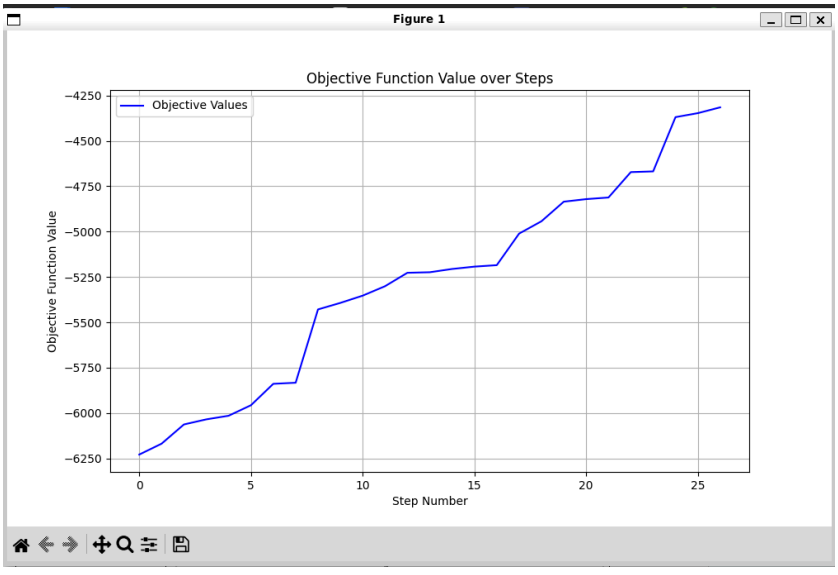
3. *Hill Climbing with Random Restart*

Cube Awal	
State Value	-5952

Cube Akhir	
State Value	-4990
Persentase ke Global Optimal	
Waktu	9.57s

4. *Stochastic Hill Climbing*

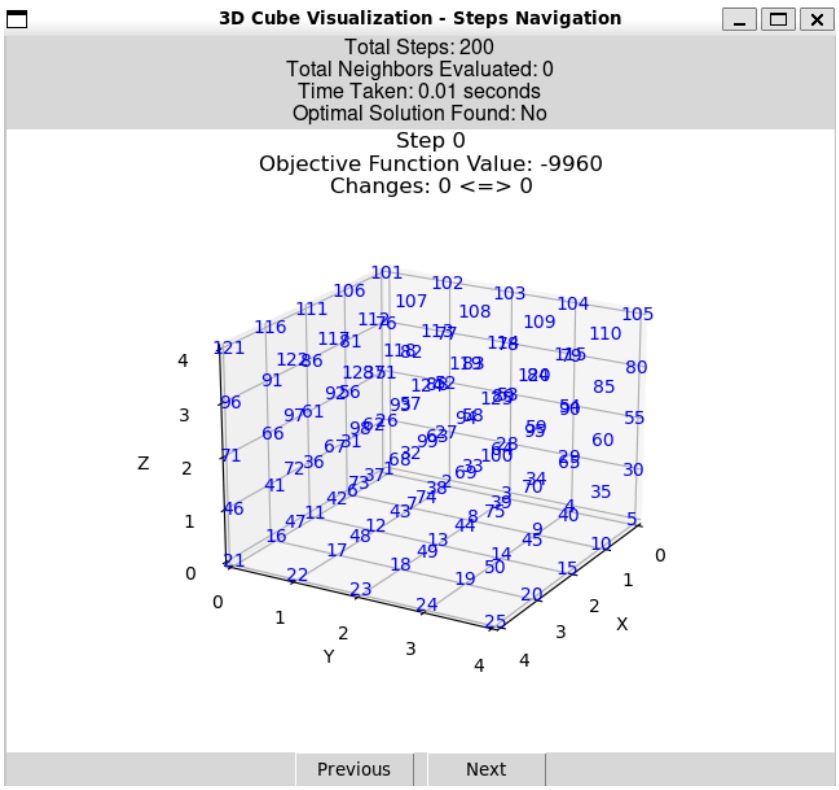
Cube Awal	
State Value	-6229

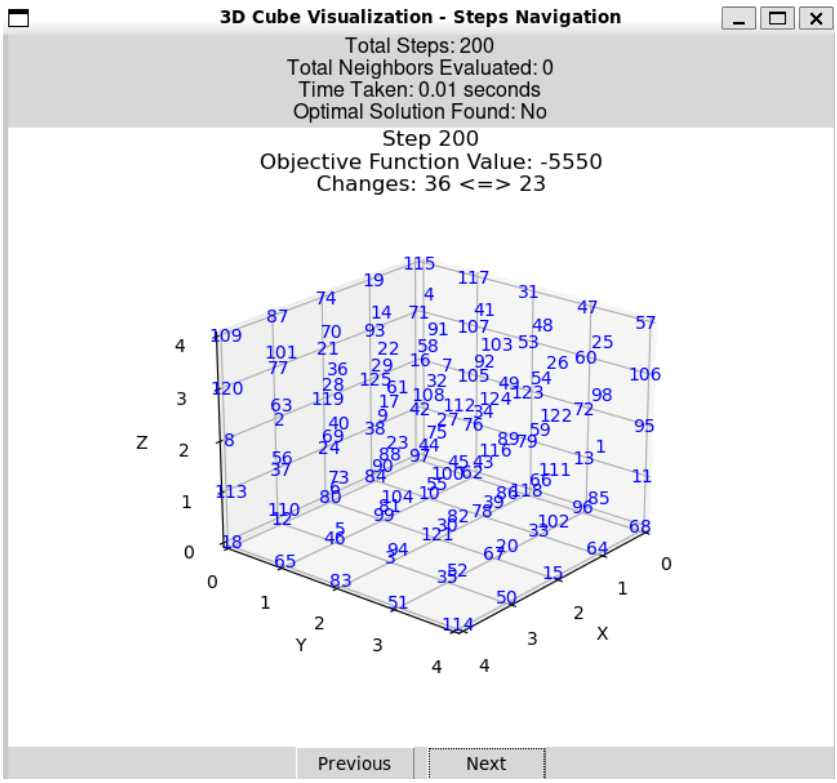
Cube Akhir	
State Value	-4315
Persentase ke Global Optimal	
Waktu	2.64s

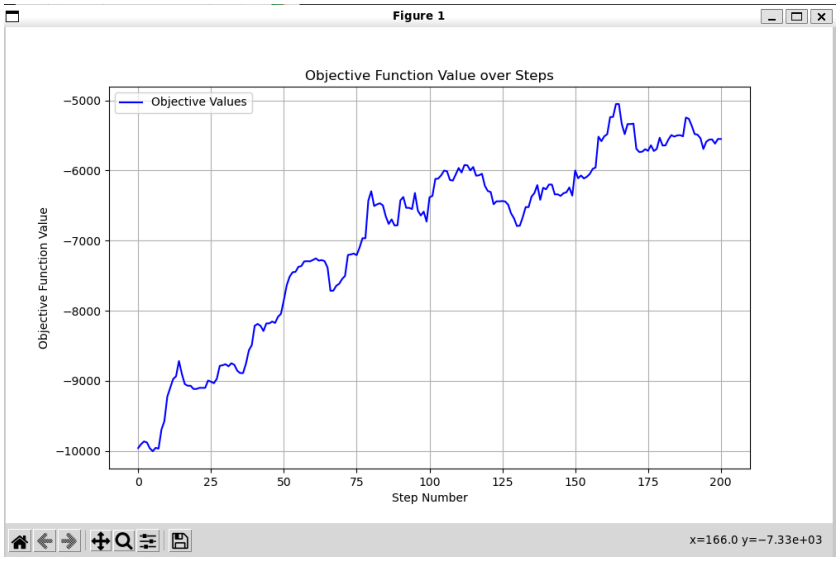
Cube Awal	Kasih Gambar Cube Awal disini

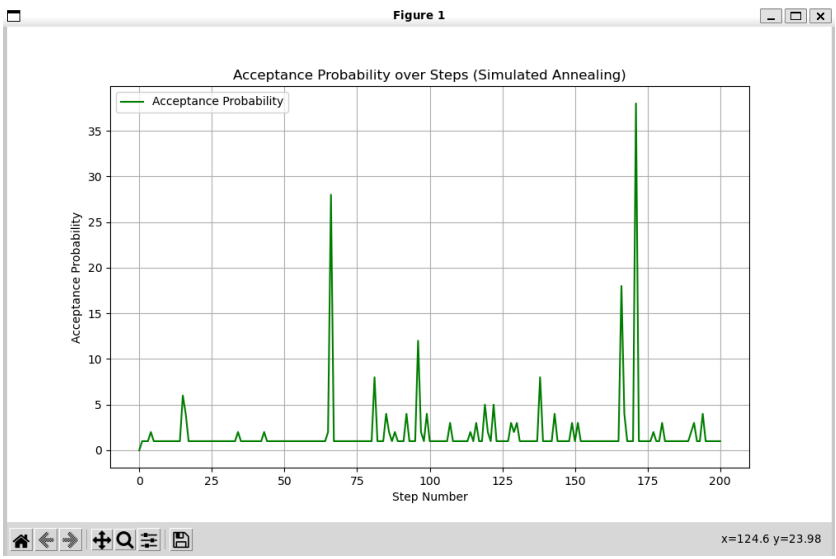
State Value	Kasih State Value nya disini
Cube Akhir	Kaish gambar cube Akhir disini
State Value	Kasih state value nya disini
Persentase ke Global Optimal	=StateValue cube akhir - Global optimal / Global optimal
Waktu	Berapa waktu untuk selesaiin

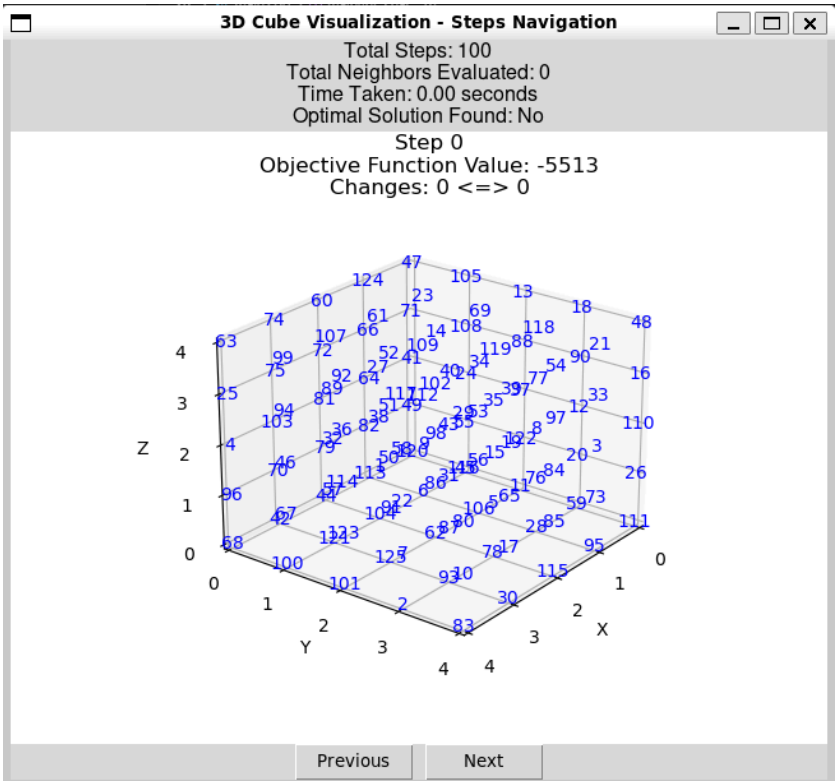
5. *Simulated Annealing*

Cube Awal	 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

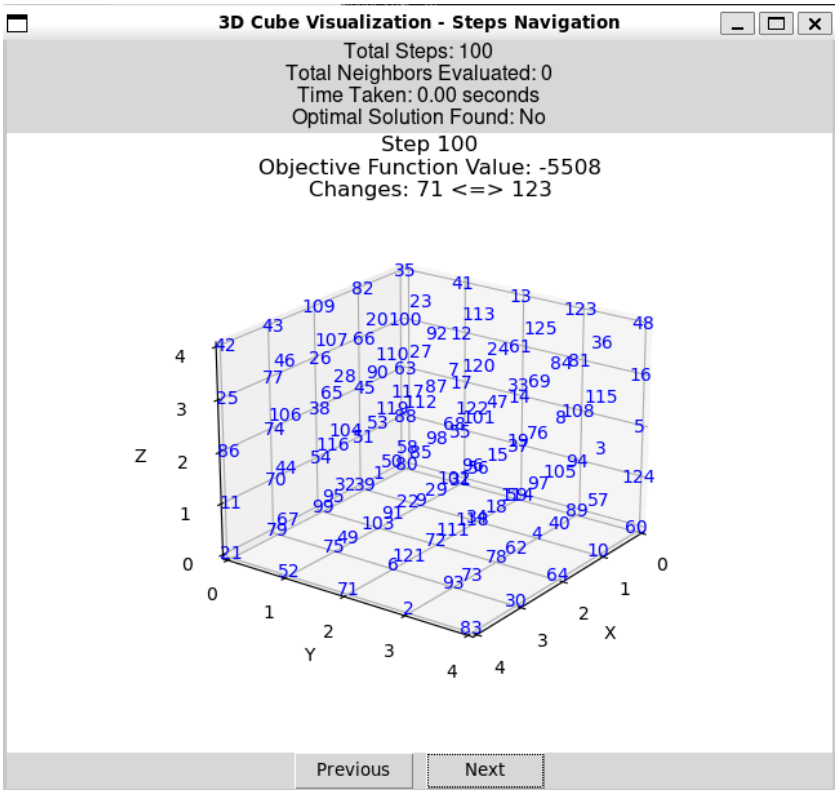
	56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
State Value	-9960
Cube Akhir	 97 62 118 96 68 84 10 78 33 64 80 99 121 67 15 12 46 3 35 50 18 65 83 51 114 42 76 79 13 11

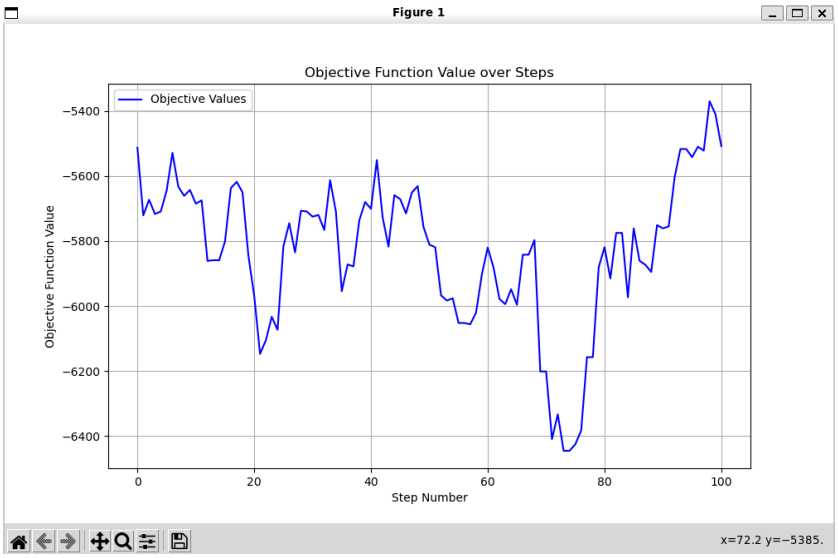
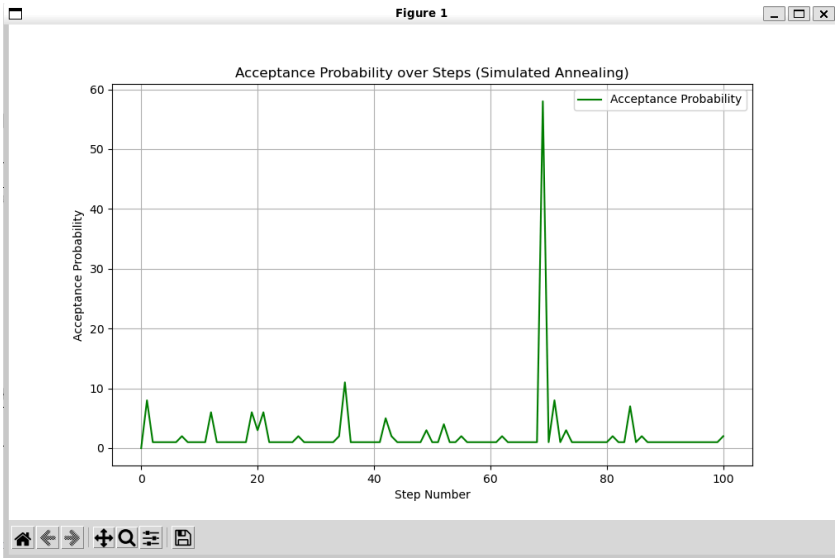
	38 44 43 66 85 24 90 55 39 102 37 6 81 30 20 113 110 5 94 52 16 105 123 72 95 125 108 34 59 1 119 9 75 116 111 2 69 88 100 86 8 56 73 104 82 71 107 53 60 106 93 58 92 54 98 21 29 32 124 122 77 28 17 27 89 120 63 40 23 45 115 117 31 47 57 19 4 41 48 25 74 14 91 103 26 87 70 22 7 49 109 101 36 61 112
State Value	-5550
Graf Plot Objective Function	

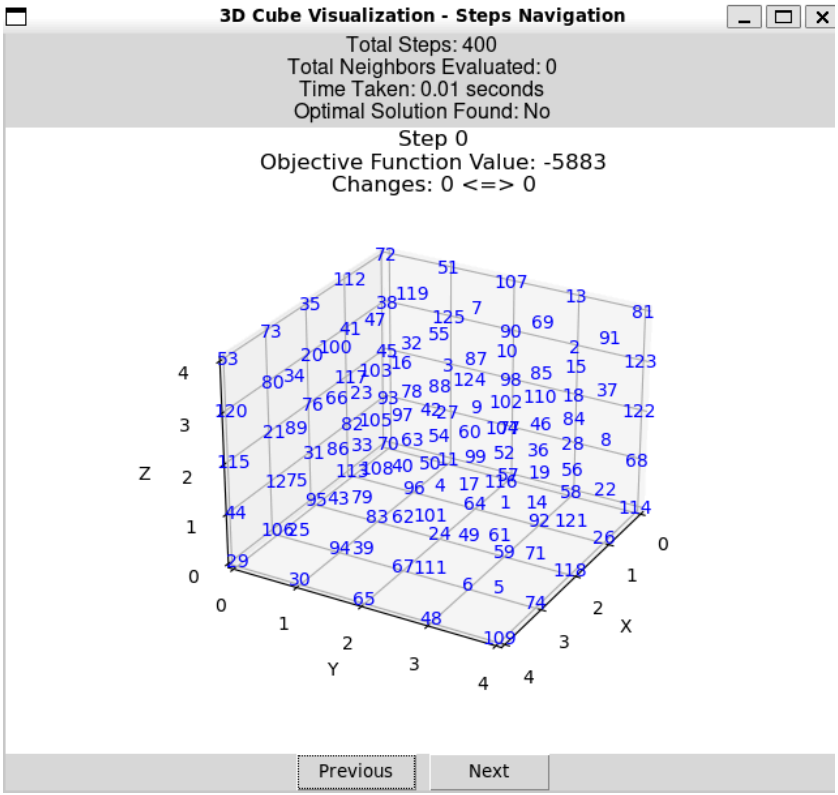
Graf Acceptance Probability	
Waktu	0.00612705 s

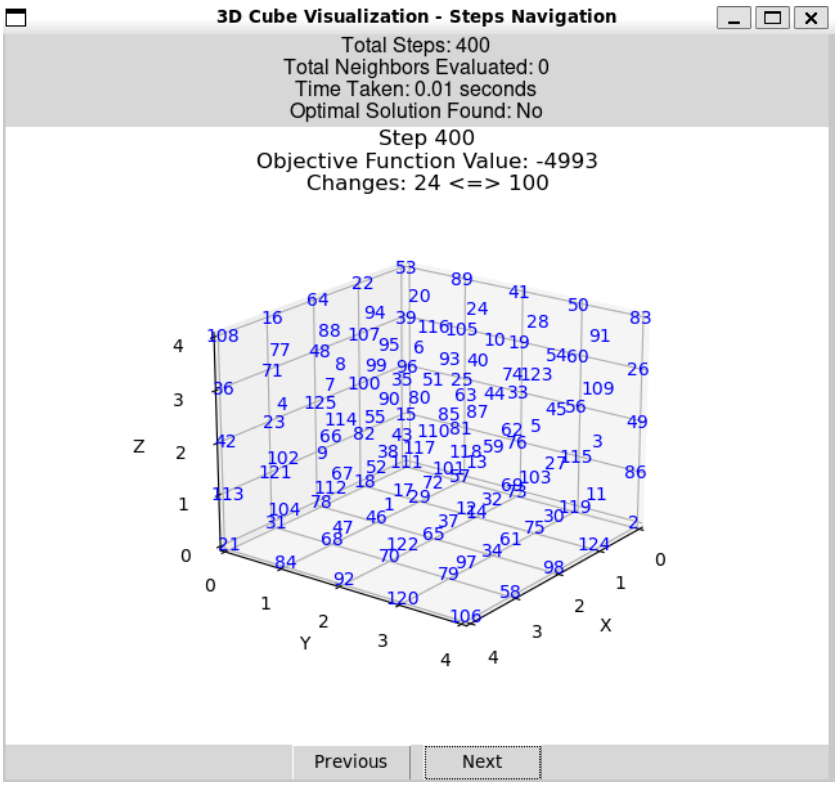
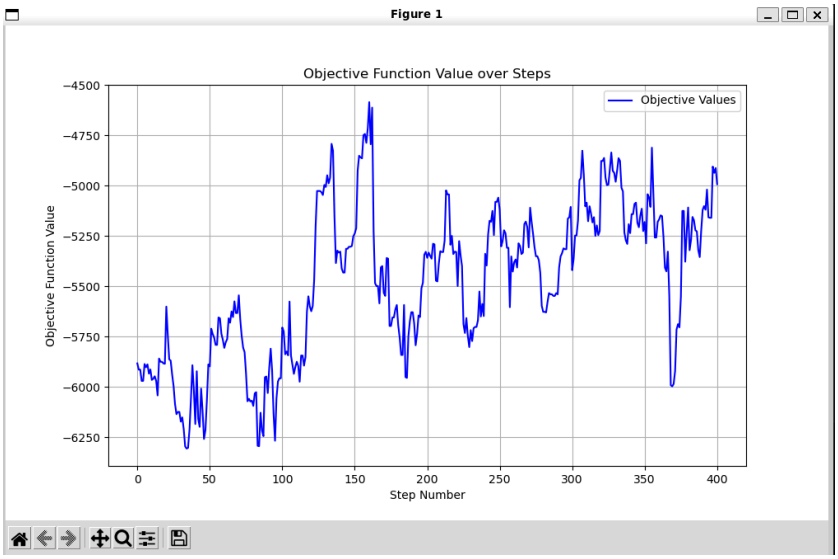
Cube Awal	 120 45 11 59 111 113 6 106 28 95 44 104 62 78 115 42 121 125 93 30 68 100 101 2 83

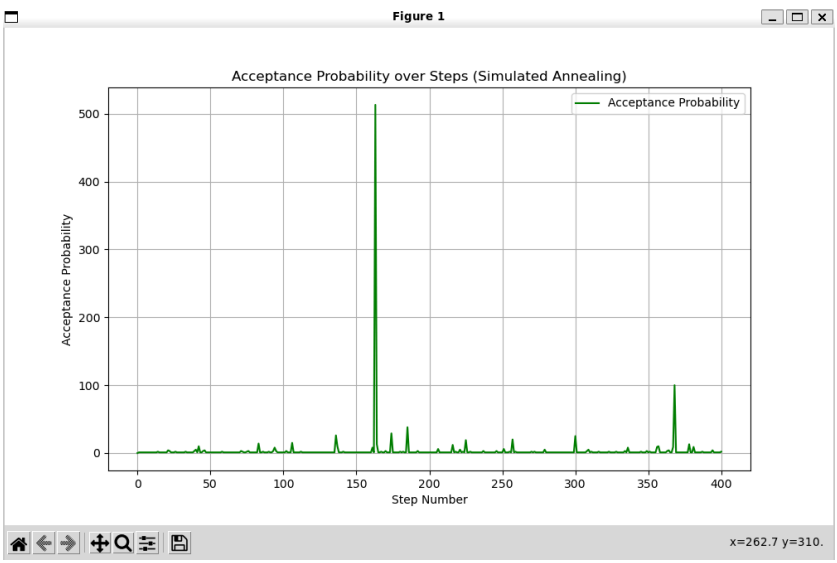
	49 55 122 20 26 82 9 56 76 73 79 1 86 5 85 70 57 91 87 17 96 67 123 7 10 41 24 37 12 110 64 112 53 8 3 81 38 98 15 84 103 32 50 31 65 4 46 114 22 80 71 108 88 90 16 66 109 34 77 33 72 27 102 35 97 75 89 51 43 19 25 94 36 58 116 47 105 13 18 48 124 23 69 118 21 60 61 14 119 54 74 107 52 40 39 63 99 92 117 29
State Value	-5513

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

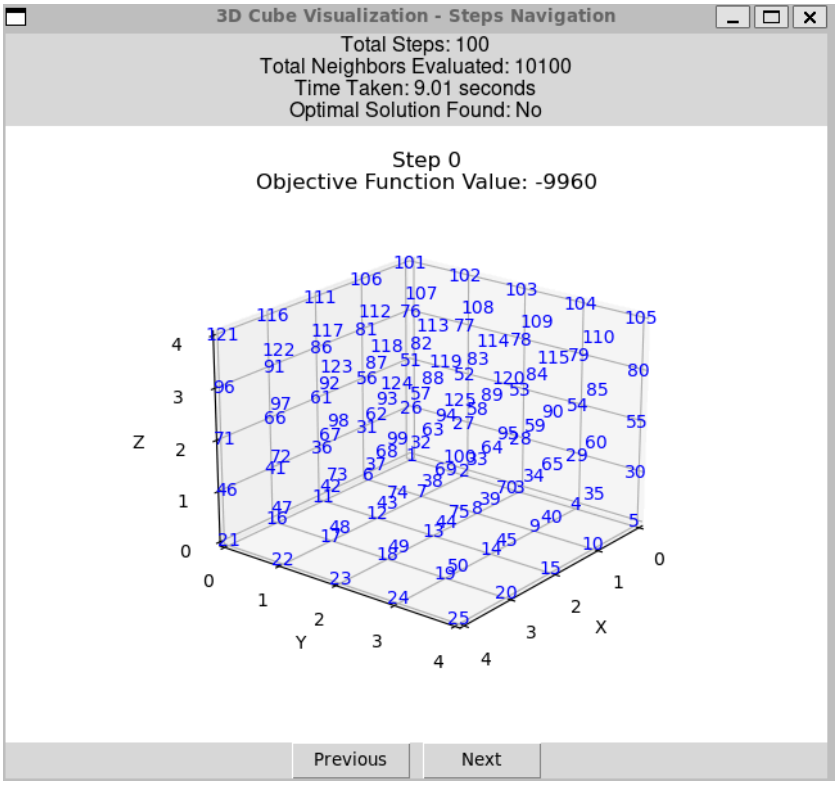
	42 46 28 117 122
State Value	-5508
Graf Plot Objective Function	
Graf Acceptance Probability	
Waktu	0.00310281 s

Cube Awal	
State Value	-5883

Cube Akhir	
State Value	-4493
Graf Plot Objective Function	

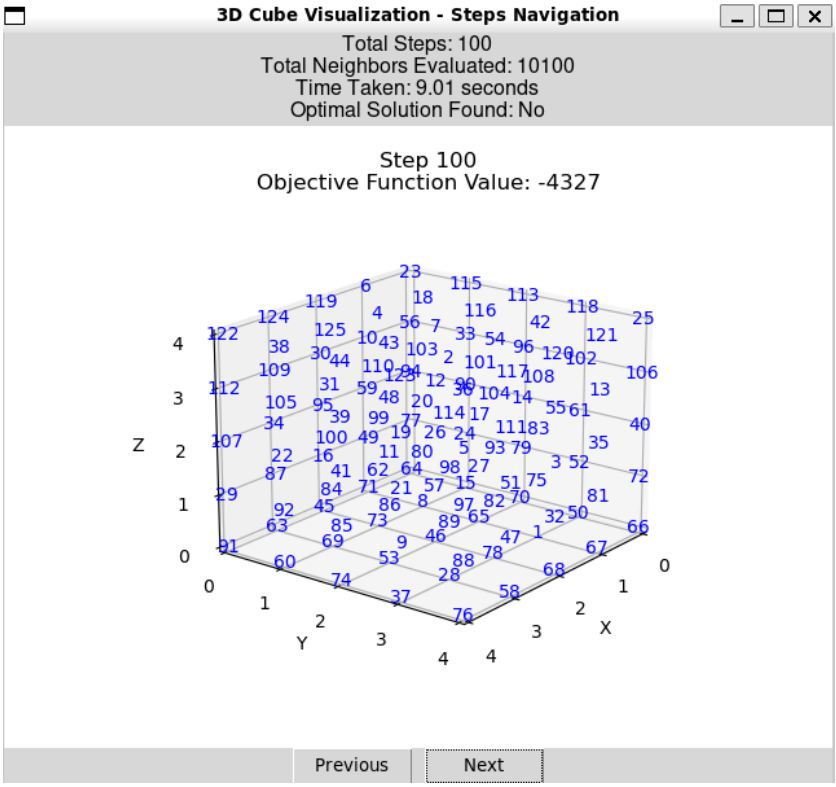
Graf Acceptance Probability	
Waktu	0.0127501 s

6. *Genetic Algorithm*

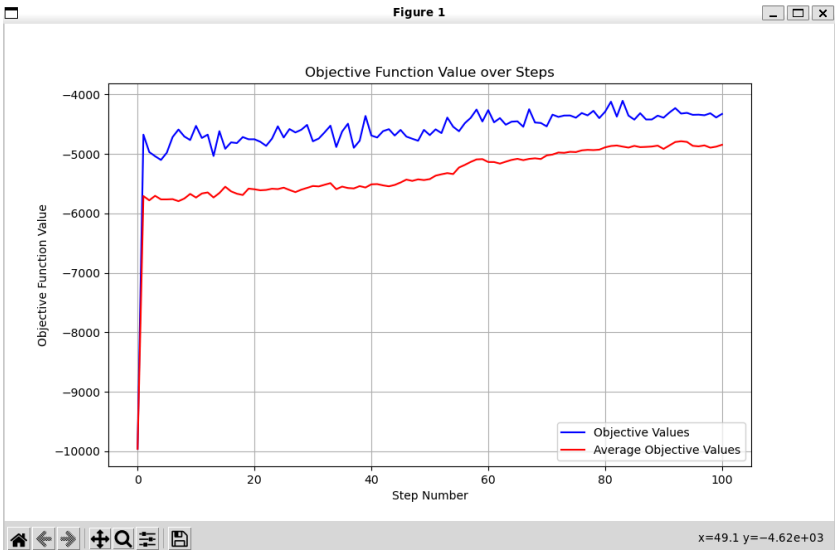
Jumlah Populasi	50
Maksimum Iterasi	50
Cube Awal	

	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
State Value	-9960

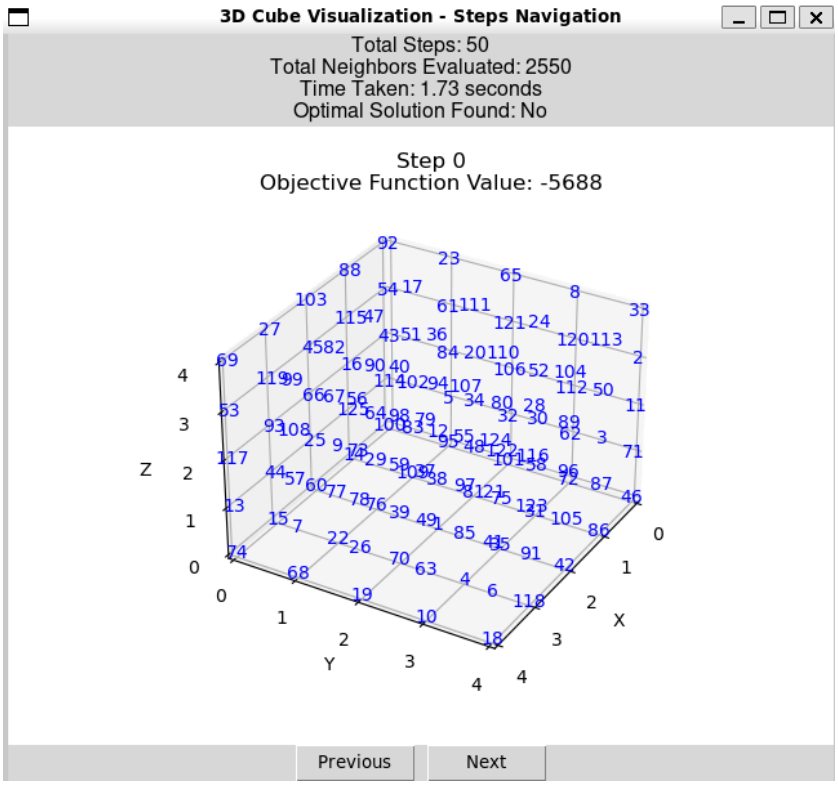
Cube Akhir

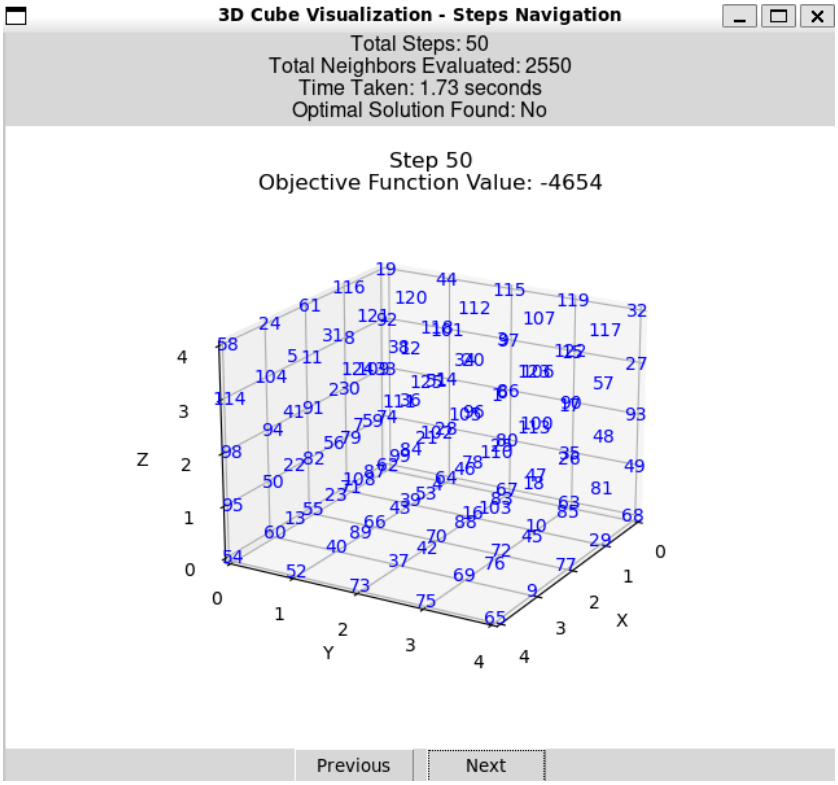


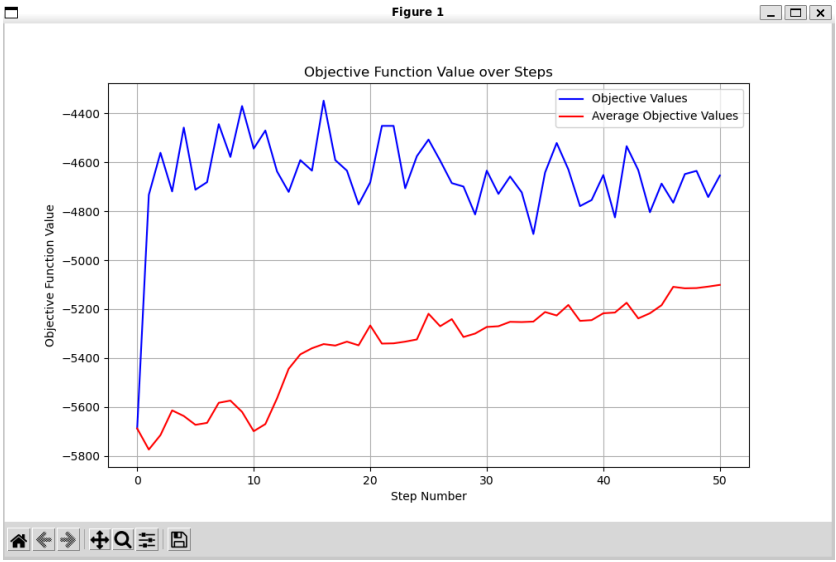
64 15 70 50 66
71 8 65 1 67
45 73 46 78 68
63 69 53 28 58
91 60 74 37 76
77 24 79 52 72
49 80 27 75 81
16 62 57 82 32
87 84 86 89 47
29 92 85 9 88
94 90 14 61 40
59 20 17 83 35
95 99 26 93 3
34 100 11 98 51
107 22 41 21 97
56 33 96 102 106
10 103 101 108 13
30 110 12 104 55
109 31 48 114 111
112 105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

	124 125 43 2 117 122 38 44 123 36
State Value	-4327
Graf Plot Objective Function / Average Objective Function	
Waktu	9.01 s

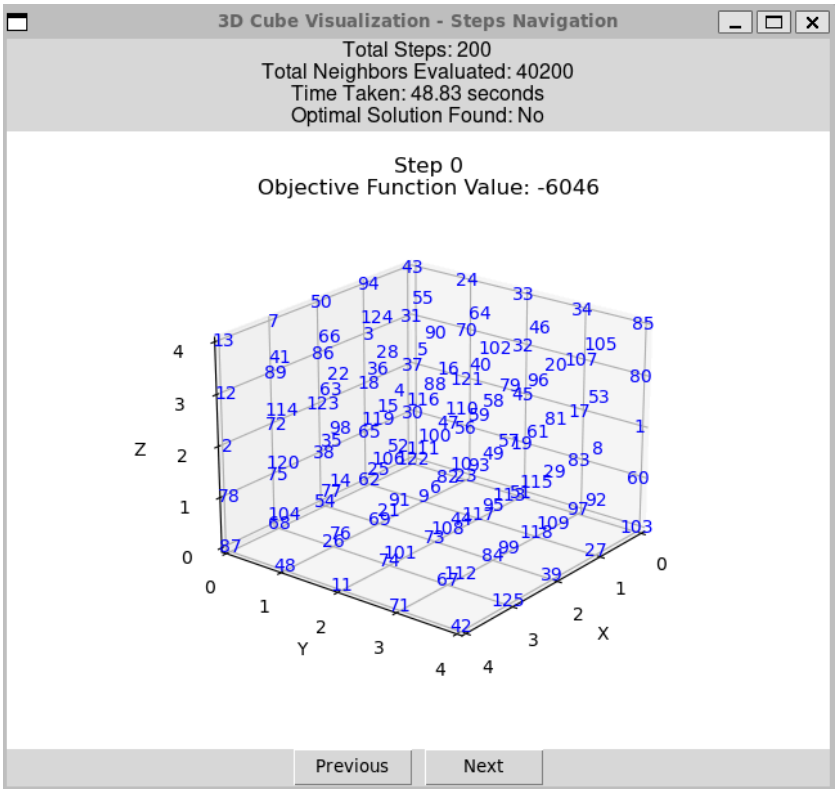
Jumlah Populasi	100
Maksimum Iterasi	100

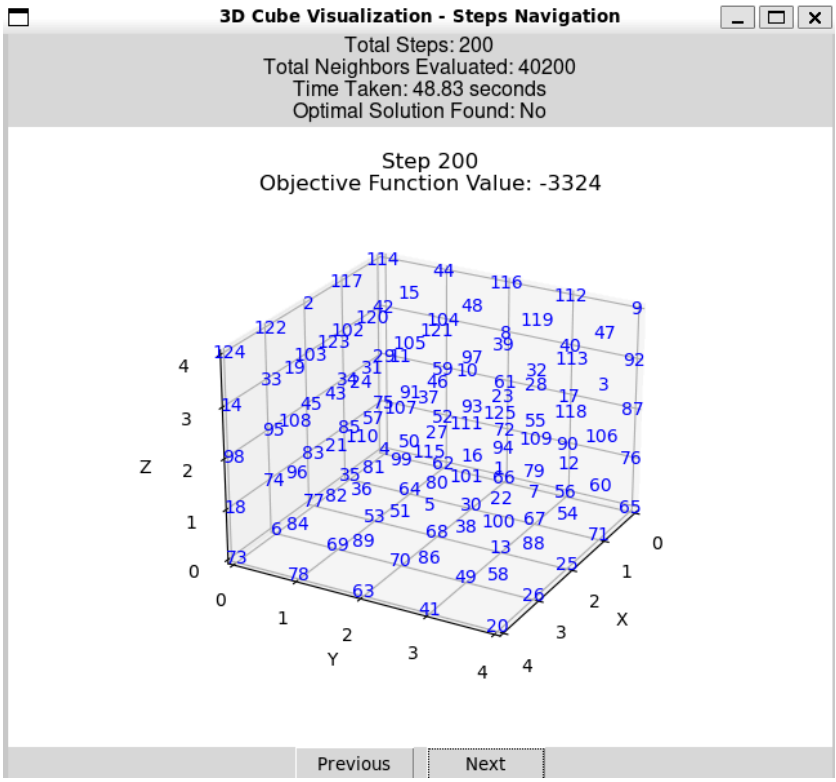
Cube Awal	
State Value	-5688

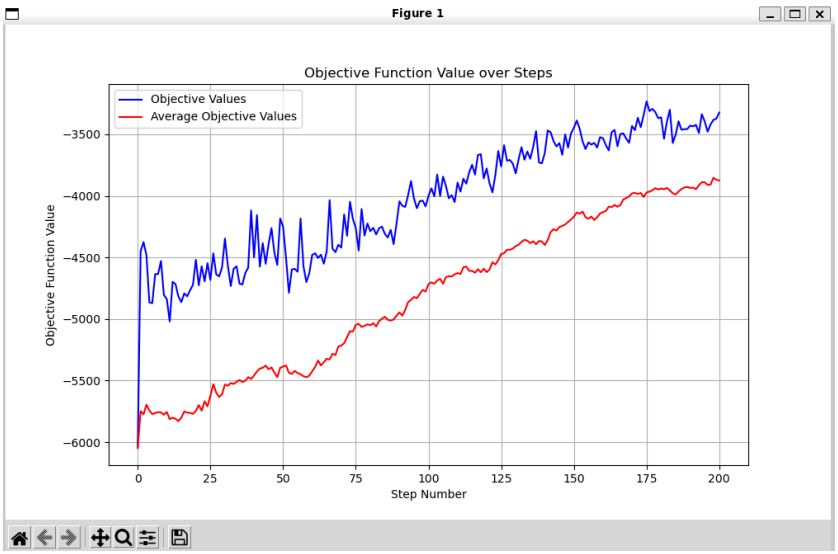
Cube Akhir	 100 95 101 72 46 14 109 81 31 86 60 76 1 35 42 15 22 70 4 118 74 68 19 10 18 114 5 32 62 71 125 83 48 58 87 25 29 38 75 105 44 77 39 85 91 13 7 26 63 6 43 84 106 112 11 16 102 34 30 3 66 64 12 122 96 93 9 59 97 123 117 57 78 49 41 54 61 121 120 2 115 51 20 52 50 45 90 94 80 89 119 67 98 55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

	69 99 56 79 124
State Value	-4654
Graf Plot Objective Function / Average Objective Function	 62 64 67 63 68 71 39 16 10 29 55 66 70 72 77 60 40 37 69 9 54 52 73 75 65 74 28 80 35 49 79 84 78 47 81 82 87 4 83 85 50 23 43 88 45 95 13 89 42 76 33 14 86 90 93 30 36 96 100 48 91 59 102 25 26 94 56 99 46 18 98 22 108 53 103 92 101 97 15 27 8 12 20 106 57 11 109 51 6 17 104 2 111 105 113 114 41 7 21 110 19 44 115 119 32 116 120 112 107 117 61 121 118 3 122 24 31 38 34 123

	58 5 124 125 1
Waktu	1.73 s

Jumlah Populasi	200
Maksimum Iterasi	200
Cube Awal	 122 23 51 97 103 62 9 117 118 27 54 69 73 84 39 68 26 74 67 125 87 48 11 71 42 30 56 19 83 60 65 111 93 115 92 38 25 6 95 109 75 77 21 108 99 78 104 76 101 112 37 121 45 17 1 18 116 59 61 8 123 119 100 49 29 72 35 106 82 113

	2 120 14 91 44 31 70 32 107 80 3 5 40 96 53 86 36 88 58 81 89 63 15 47 57 12 114 98 52 10 43 24 33 34 85 94 55 64 46 105 50 124 90 102 20 7 66 28 16 79 13 41 22 4 110
State Value	-6046
Cube Akhir	 4 62 66 56 65 35 64 30 67 71 77 53 68 13 25 6 69 70 49 26 73 78 63 41 20 75 52 72 90 76 85 50 16 79 60 83 81 80 22 54 74 82 51 38 88 18 84 89 86 58

	29 59 61 17 87 34 91 93 55 106 45 57 27 94 12 95 21 99 101 7 98 96 36 5 100 42 104 8 40 92 102 105 97 32 3 103 31 46 23 118 33 43 107 111 109 14 108 110 115 1 114 44 116 112 9 117 15 48 119 47 2 120 121 39 113 122 123 11 10 28 124 19 24 37 125
State Value	-3324
Graf Plot Objective Function / Average Objective Function	
Waktu	48.83 s

D. Analisis

Dari beberapa algoritma yang diimplementasi, terlihat bahwa algoritma XXX adalah algoritma yang paling dekat dengan global optimal.

Algoritma Steepest Ascent berjarak cukup dekat dengan global optimal, namun tidak mencapai global optimal. Hal ini dikarenakan algoritma ini selalu mengambil hasil yang paling baik, yang sangat memungkinkan bahwa pilihan yang diambil tersebut dapat menyebabkan terjebak pada local optima. Di sisi lain, algoritma ini termasuk salah satu algoritma yang dapat menemukan solusi yang sangat cepat, meskipun hasil yang diperoleh tidak optimal.

Algoritma Sideways move menyelesaikan permasaslahan dari algoritma steepest ascent yang seringkali terjebak di lokal optima dengan memberikan kesempatan untuk berpindah ke state dengan value yang sama. Hal ini ditunjukkan dengan hasil nya yang lebih baik jika dibandingkan dengan steepest ascent walau sedikit mengorbankan performa secara waktu.

Algoritma

Algoritma Random Restart masih berjarak cukup jauh dari global optimal. Hal ini dikarenakan algoritma ini melakukan algoritma steepest ascent berkali-kali, dan mengulang di saat sudah local optimal. Hal ini membuat algoritma ini memerlukan iterasi yang lebih banyak dan waktu yang lebih lama jika dibandingkan dengan algoritma steepest ascent biasa. Walaupun jika diberikan waktu dan iterasi lebih, algoritma ini besar peluang untuk mencapai global optimum karena tidak akan terjebak dalam lokal optimal, karena jika terjebak, maka algoritma ini akan mencari lagi jalan lain dari awal.

Algoritma Simulated Annealing memiliki pertumbuhan menuju solusi yang cukup lambat. Namun dengan jumlah iterasi yang sedikit, terlihat bahwa algoritma ini menuju ke solusi. Hal ini menunjukan dengan jumlah iterasi yang semakin banyak, algoritma ini bisa mendekati global optimal dengan baik. Algoritma ini tergolong lambat karena pemilihan neighbor yang dibangkitkan secara acak, dan dibolehkannya neighbor dengan state yang lebih buruk untuk dipilih. Selain itu, jumlah iterasi yang sedikit serta cutoff nya yang cukup longgar, yakni 0.5, juga berkontribusi untuk membuat algoritma ini kurang bisa bersinar.

Algoritma Genetic Algorithm dapat dibilang kurang efektif dalam melakukan pencarian menuju goal state, dilihat dari nilai objektif terbaik dan nilai objektif rata-rata tiap populasinya yang selalu naik-turun serta secara keseluruhan tren grafik nilai objektifnya cenderung datar. Hal ini dikarenakan algoritma Genetic Algorithm yang digunakan mengandung banyak ketidakpastian atau *randomness*, mulai dari pemilihan *parent* yang dilakukan secara probabilistik menggunakan *fitness function*, proses *crossover* yang memanfaatkan probabilitas untuk meletakkan angka yang konflik, hingga proses mutasi yang kejadiannya juga berdasarkan probabilitas. Genetic Algorithm juga berbeda dari algoritma Hill Climbing, dimana ia tetap pindah ke state baru walaupun state baru tersebut memiliki nilai objektif yang lebih buruk dari *current state*. Berdasarkan eksperimen, didapatkan bahwa semakin besar nilai jumlah populasi dan maksimum iterasi, maka final state yang diraih memiliki nilai obyektif yang semakin baik. Hal ini menunjukkan bahwa nilai jumlah populasi dan maksimum iterasi berbanding lurus dengan nilai obyektif final state, namun juga semakin lama waktu komputasinya.

KESIMPULAN DAN SARAN

A. Kesimpulan

Dari hasil eksperimen di atas, dapat disimpulkan bahwa Algoritma sideways move Hill Climbing merupakan algoritma yang paling baik untuk permasalahan *Diagonal Magic Cube*. Algoritma Steepest Ascent Hill Climbing bisa menyelesaikan dengan paling cepat, walaupun algoritma ini seringkali terjebak di lokal optimal. Selain itu, didapatkan juga bahwa genetic algorithm dan simulated annealing memiliki performa yang cukup buruk jika dibandingkan dengan algoritma steepest ascent hill climbing untuk jumlah iterasi yang sedikit.

B. Saran

Saran yang bisa didapatkan untuk kita adalah mencoba untuk membuat visualisasi dari seluruh tahap pencarian solusi dan menyajikannya dalam bentuk *video player*. Hal ini dapat lebih membantu pengguna untuk memvisualisasikan setiap perubahan yang ada. Selain itu, parameter-parameter yang ada juga dapat disesuaikan, seperti jumlah iterasi maksimum, temperatur awal dan laju penurunannya di algoritma *Simulated Annealing*, jumlah restart di algoritma *Random Restart Hill Climbing*, dll. Karena dengan parameter yang lain akan menghasilkan hasil yang lain juga.

Selain itu, terkait kinerja kami mendapat bahwa seharusnya kami mulai mengerjakan sejak jauh hari dan tidak deadliner supaya pengerjaan dapat dilakukan dengan pengujian secara benar dan tertata.

PEMBAGIAN KERJA

NIM	Pembagian Kerja
13522004	Random restart Hill Climbing, Sideways Hill Climbing, Stochastic Hill Climbing
13522007	Genetic Algorithm, Position
13522047	Utils, Visualisasi, Steepest Ascent Hill Climbing
13522051	Simulated Annealing

REFERENSI

[1] Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach 4th Edition* (Section 1.2, Chapter 4).

[2] Trump, W. (2005). *Perfect magic cubes*. Diakses 23 September 2024, dari https://www.trump.de/magic-squares/magic-cubes/cubes-1.html