Tugas Besar 1 IF3170 Inteligensi Artifisial

Pencarian Solusi Diagonal Magic Cube dengan Local Search

Dibuat oleh:

Akbar Al Fattah - 13522036

Devinzen - 13522064

Julian Caleb Simandjuntak - 13522099

Christopher Brian - 13522106

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika - Komputasi

Institut Teknologi Bandung

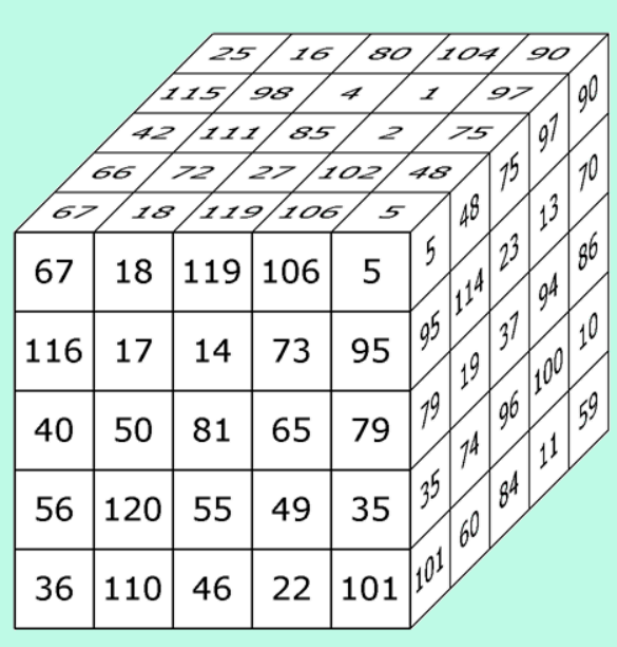
Daftar Isi

Deskripsi Persoalan

Diagonal magic cube merupakan sebuah kubus yang tersusun atas angka 1 hingga n^3 tanpa adanya pengulangan, dengan n adalah panjang sisi pada kubus tersebut. Angka-angka pada diagonal magic cube tersusun sedemikian rupa sehingga:

- Terdapat satu angka yang merupakan magic number dari kubus tersebut
- 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

Contoh diagonal magic cube dengan ukuran 5x5x5 adalah sebagai berikut:



Pada Tugas Besar 1, mahasiswa diminta untuk menyelesaikan permasalahan diagonal magic cube berukuran 5x5x5 dengan menggunakan local search dan initial state yang berupa susunan angka 1 hingga 5^3 secara acak. Local search yang digunakan meliputi Steepest Ascent Hill-climbing, Hill-climbing

with Sideways Move, Random Restart Hill-climbing, Stochastic Hill-climbing, Simulated Annealing, dan Genetic Algorithm. Langkah yang dapat dilakukan untuk setiap iterasinya adalah menukar 2 angka pada kubus. Khusus untuk Genetic Algorithm, boleh dilakukan penukaran posisi lebih dari 2 angka sekaligus dalam satu iterasi. Penyelesaian diagonal magic cube harus dapat divisualisasikan.

Pemilihan Objective Function

Objective function yang kami gunakan adalah total banyak baris, kolom, pilar, diagonal sisi, dan diagonal ruang pada kubus yang jumlahnya sama dengan *magic constant* dari orde kubus tersebut. Secara matematis, persamaan untuk menghitung *objective function* tersebut adalah sebagai berikut

$$f(cube) = \sum_i 1[baris(i) = M_3(n)] + \sum_i 1[kolom(i) = M_3(n)] + \sum_i 1[pilar(i) = M_3(n)] +$$

$$\sum_i 1[diagonalsisi(i) = M_3(n)] + \sum_i 1[diagonalruang(i) = M_3(n)]$$

dengan 1[...] sebagai fungsi indikator yang bernilai 1 jika kondisi di dalamnya bernilai benar (jumlah sama dengan *magic constant*) dan 0 untuk sebaliknya. *Magic constant* dari sebuah kubus dapat dihitung menggunakan persamaan berikut.

$$M_3(n) = \frac{n(n^3+1)}{2}$$

dengan n menandakan orde (panjang sisi) kubus. Untuk kubus berorder 5, nilai *magic constant*-nya sama dengan 109.

Fungsi di atas kami pilih sebagai *objective function* dikarenakan beberapa alasan. Fungsi yang terdiri atas keluaran biner menyederhanakan kalkulasi (hanya perlu mengecek jumlah, tidak perlu per elemen kubus), sehingga mempercepat pembaruan nilai fungsi setelah setiap eksplorasi tetangga baru. Keluaran biner juga mempermudah implementasi fungsi dalam program komputer.

Keluaran biner juga memungkinkan pembaruan yang efisien pada proses pencarian lokal, karena hanya perlu dilakukan komputasi ulang pada bagian yang diubah, bukan keseluruhan kubus. Komputasi ulang menjadi lebih cepat karena tidak perlu dilakukan evaluasi terhadap setiap baris, kolom, pilar, atau diagonal setelah setiap langkah.

Penjelasan Implementasi Algoritma Local Search

Untuk menyelesaikan persoalan diatas, akan digunakan enam buah algoritma local search, yaitu Steepest Ascent Hill-Climbing, Hill-Climbing with Sideways Move, Random Restart Hill-Climbing, Stochastic Hill-Climbing, Simulated Annealing, dan Genetic Algorithm.

Magic Cube

Kelas utama MagicCubeClass digunakan untuk merepresentasikan sebuah magic cube. Kelas ini memiliki atribut cubeState, value (nilai objective function), dan sumConstraint. Konstruktor akan menginisialisasi kelas magic cube dengan value 0, cubeState berupa kubus berukuran 5 kali 5 kali 5 yang semua elemennya bernilai 0, dan sumConstraint bernilai 315. Kelas ini memiliki getter dan setter untuk cubeState dan value, serta getter untuk sumConstraint. Selain itu, terdapat beberapa fungsi private untuk menghitung sumConstraint, menghitung jumlah baris yang total nilainya sama dengan magic constant, menghitung jumlah kolom yang total nilainya sama dengan magic constant, menghitung jumlah pilar yang total nilainya sama dengan magic constant, menghitung jumlah diagonal bidang yang total nilainya sama dengan magic constant, dan menghitung jumlah diagonal ruang yang total nilainya sama dengan magic constant. Terdapat pula beberapa fungsi publik untuk mendapatkan state awal acak, menghitung nilai objective function dari sebuah state, mendapatkan suksesor acak, mendapatkan suksesor terbaik (state dengan nilai objective function terbesar), dan menyalin sebuah objek dari kelas magic cube.

Berikut adalah source code untuk kelas utama magic cube.

```
export class MagicCubeClass {
    private cubeState: number[][][] = []; //representasi magic cube dalam bentuk array 3d
    private value: number; //nilai objective function dari magic cube saat ini
    private sumConstraint: number;
    public getCurrentValue() : number{ //method mendapatkan objective function dari magic cube saat ini
        return this.value;
    }
```

```
    public setCurrentValue(value: number){ //set value dari suatu state dengan nilai yang baru
        this.value = value;
    }
    public getCurrentState(): number[][][]{ //method untuk mendapatkan state cube saat ini
        return this.cubeState;
    }
    public setCurrentState(state: number[][][]):void{ //set state magic cube
        this.cubeState = state;
    }
    public getSumConstraintVal(): number{ // mendapatkan syarat jumlah agar bisa menjadi magic cube
        return this.sumConstraint;
    }
    constructor() { //konstruktor yang akan inisialisasi objective function = 0 dan nilai cubenya 0 semua
        this.value = 0;
        this.cubeState = Array.from({length:5},()=>Array.from({length:5},()=>Array(5).fill(0)));
        this.sumConstraint = this.getSumConstraint(5);
    }

    private getSumConstraint(n:number):number{
        //menghitung nilai dari hasil penjumlahan di setiap bagian magic cube yang diminta yang menjadi syarat suatu
kubus
        //bisa dianggap sebagai magic cube (untuk kasus ini, n=5, maka sum constraintnya adalah 315)
        //artinya, penjumlahan semua bagian magic cube yang diperlukan harus sama dengan 315
        return (n * (n**3+1) )/2;
    }

    public getRandomInitState():number[][][]{ //buat acak state cube yang bakal jadi initial state
        //buat array yang isinya angka dari 1 sampai 125
        const cubeNumbers = Array.from({length:125}, (_,i) => i+1);

        //acak angka dari 1 sampai 125
        for(let i = cubeNumbers.length - 1; i >= 0; i--){
            const j = Math.floor(Math.random()*(i+1));
            [cubeNumbers[i],cubeNumbers[j]] = [cubeNumbers[j],cubeNumbers[i]];
        }

        const initState: number[][][] = Array.from({length:5},()=>Array.from({length:5},()=>Array(5).fill(0)));
        let idx = 0;
        for(let x =0; x<5; x++){
            for(let y =0; y <5; y++){
                for(let z =0; z<5; z++){
                    initState[x][y][z] = cubeNumbers[idx++];
                }
            }
        }

        return initState;
    }

    private row315(state: number[][][]): number {
        //hitung banyaknya baris yang jumlahnya sama dengan sumConstraint (n = 5, sumConstraint = 315)
        let cnt = 0;
        for (let z = 0; z < 5; z++) {
            for (let y = 0; y < 5; y++) {
                const sum = state[z][y].reduce((acc, val) => acc + val, 0);
                if (sum === this.sumConstraint) {
                    cnt++;
                }
```

```
            }
        }
        return cnt;
    }

    private column315(state: number[][][]): number {
        //hitung banyaknya kolom yang jumlahnya sama dengan sumConstraint (n = 5, sumConstraint = 315)
        let cnt = 0;
        for (let z = 0; z < 5; z++) {
            for (let x = 0; x < 5; x++) {
                const sum = state[z].reduce((acc, row) => acc + row[x], 0);
                if (sum === this.sumConstraint) {
                    cnt++;
                }
            }
        }
        return cnt;
    }

    private pillar315(state: number[][][]): number {
        //hitung banyaknya tiang yang jumlahnya sama dengan sumConstraint (n = 5, sumConstraint = 315)
        let cnt = 0;
        for (let x = 0; x < 5; x++) {
            for (let y = 0; y < 5; y++) {
                let sum = 0;
                for (let z = 0; z < 5; z++) {
                    sum += state[z][y][x];
                }
                if (sum === this.sumConstraint) {
                    cnt++;
                }
            }
        }
        return cnt;
    }
    private planeDiag315(state: number[][][]): number {
        //hitung banyaknya diagonal bidang yang jumlahnya sama dengan sumConstraint (n = 5, sumConstraint = 315)
        let cnt = 0;

        // hitung semua diagonal bidang di bidang z
        for (let z = 0; z < 5; z++) {
            // periksa diagonal bidang dari kiri bawah ke kanan atas
            let sum1 = 0;
            for (let i = 0; i < 5; i++) {
                sum1 += state[z][i][i];
            }
            if (sum1 === this.sumConstraint) {
                cnt++;
            }

            // periksa diagonal bidang dari kiri atas ke kanan bawah
            let sum2 = 0;
            for (let i = 0; i < 5; i++) {
                sum2 += state[z][i][4 - i];
            }
            if (sum2 === this.sumConstraint) {
                cnt++;
            }
```

```
    }

    // hitung semua diagonal bidang di bidang y
    for (let y = 0; y < 5; y++) {
        // periksa diagonal bidang dari kiri bawah ke kanan atas
        let sum1 = 0;
        for (let i = 0; i < 5; i++) {
            sum1 += state[i][y][i];
        }
        if (sum1 === this.sumConstraint) {
            cnt++;
        }

        // periksa diagonal bidang dari kiri atas ke kanan bawah
        let sum2 = 0;
        for (let i = 0; i < 5; i++) {
            sum2 += state[i][y][4 - i];
        }
        if (sum2 === this.sumConstraint) {
            cnt++;
        }
    }

    // hitung semua diagonal bidang di bidang x
    for (let x = 0; x < 5; x++) {
        // periksa diagonal bidang dari kiri bawah ke kanan atas
        let sum1 = 0;
        for (let i = 0; i < 5; i++) {
            sum1 += state[i][i][x];
        }
        if (sum1 === this.sumConstraint) {
            cnt++;
        }

        // periksa diagonal bidang dari kiri atas ke kanan bawah
        let sum2 = 0;
        for (let i = 0; i < 5; i++) {
            sum2 += state[i][4 - i][x];
        }
        if (sum2 === this.sumConstraint) {
            cnt++;
        }
    }

    return cnt;
}
private spaceDiag315(state: number[][][]): number {
    //hitung banyaknya diagonal ruang yang jumlahnya sama dengan sumConstraint (n = 5, sumConstraint = 315)
    let cnt = 0;

    let sum1 = 0;
    for (let i = 0; i < 5; i++) {
        sum1 += state[i][i][i];
    }
    if (sum1 === this.sumConstraint) {
        cnt++;
    }
```

```
        let sum2 = 0;
        for (let i = 0; i < 5; i++) {
            sum2 += state[i][i][4 - i];
        }
        if (sum2 === this.sumConstraint) {
            cnt++;
        }

        let sum3 = 0;
        for (let i = 0; i < 5; i++) {
            sum3 += state[i][4 - i][i];
        }
        if (sum3 === this.sumConstraint) {
            cnt++;
        }

        let sum4 = 0;
        for (let i = 0; i < 5; i++) {
            sum4 += state[i][4 - i][4 - i];
        }
        if (sum4 === this.sumConstraint) {
            cnt++;
        }

        return cnt;
    }

    public objectiveFunction(state: number[][][]): number {
        //hitung nilai Objective Function dari state saat ini
        const rowCnt = this.row315(state);
        const colCnt = this.column315(state);
        const pillarCnt = this.pillar315(state);
        const planeDiagCnt = this.planeDiag315(state);
        const spaceDiagCnt = this.spaceDiag315(state);
        return rowCnt + colCnt + pillarCnt + pillarCnt + planeDiagCnt+spaceDiagCnt;
    }

    public getRandomSuccessor(state:number[][][]): number[][][]{ // method untuk mencari state tetangga random

        const newState = state.map(layer => layer.map(row => [...row]));
        //pilih posisi pertama acak
        const p1 ={
            x: Math.floor(Math.random() * 5),
            y: Math.floor(Math.random() * 5),
            z: Math.floor(Math.random() * 5)
        };
        //pilih posisi kedua acak
        const p2 ={
            x: Math.floor(Math.random() * 5),
            y: Math.floor(Math.random() * 5),
            z: Math.floor(Math.random() * 5)
        }
        //kalo posisinya kebetulan sama, acak lagi p2 sampai beda posisi dengan p1
        while (p1.x === p2.x && p1.y === p2.y && p1.z === p2.z){
            p2.x = Math.floor(Math.random() * 5);
            p2.y = Math.floor(Math.random() * 5);
            p2.z = Math.floor(Math.random() * 5);
        }
```

```
        //tukar 2 angka di posisi yang udah dipilih
        const temp = newState[p1.x][p1.y][p1.z];
        newState[p1.x][p1.y][p1.z] = newState[p2.x][p2.y][p2.z];
        newState[p2.x][p2.y][p2.z] = temp;

        return newState;
    }

    public getBestSuccessor(state:number[][][]): number[][][]{
        //mendapatkan successor dengan value objective function tertinggi
        let bestState = state;
        let bestValue = this.objectiveFunction(bestState);
        for (let x1 = 0; x1 < 5; x1++) {
            for (let y1 = 0; y1 < 5; y1++) {
                for (let z1 = 0; z1 < 5; z1++) {
                    for (let x2 = 0; x2 < 5; x2++) {
                        for (let y2 = 0; y2 < 5; y2++) {
                            for (let z2 = 0; z2 < 5; z2++) {
                                if (x1 === x2 && y1 === y2 && z1 === z2) {
                                    continue;
                                }
                                const neighbor = state.map(layer => layer.map(row => [...row]));

                                const temp = neighbor[x1][y1][z1];
                                neighbor[x1][y1][z1] = neighbor[x2][y2][z2];
                                neighbor[x2][y2][z2] = temp;

                                const neighborVal = this.objectiveFunction(neighbor);

                                if (neighborVal > bestValue) {
                                    bestState = neighbor;
                                    bestValue = neighborVal;
                                }
                            }
                        }
                    }
                }
            }
        }

        return bestState;
    }

    public clone(): MagicCubeClass { // Fungsi untuk membuat duplikasi dari magic cube
        const newCube = new MagicCubeClass();
        newCube.setCurrentState(this.cubeState.map(layer => layer.map(row => [...row])));
        newCube.setCurrentValue(this.value);
        return newCube;
    }
}
```

Steepest Ascent Hill-Climbing

Steepest Ascent Hill-Climbing adalah algoritma local search yang dimulai dengan sebuah randomly-generated initial state, yang pada kasus ini adalah

sebuah kubus 5x5x5 yang seluruhnya telah diisi dengan angka 1 - 125 secara acak dan unik (tidak ada duplikat), lalu dilakukan iterasi yang membangkitkan seluruh suksesor yang berupa kubus dengan pertukaran antar 2 angka bebas pada kubus dan dipilih satu suksesor dengan objective function value yang paling besar (berdasarkan objective function), lebih besar daripada objective function value dari kubus initial state. Iterasi dilakukan lagi terhadap suksesor tersebut dan diulangi hingga mencapai puncaknya, yaitu ketika tidak ada suksesor dengan objective function value yang lebih besar. Steepest Ascent Hill-Climbing belum tentu mengembalikan global maksimum, bisa jadi stuck di local maksimum.

Berikut adalah source code untuk kelas steepest ascent hill climbing.

```
import {MagicCubeClass} from "../magic-cube-class/magic-cube-class";

export class hcSteepestAscent {
    public static search(magicCube : MagicCubeClass): Array<MagicCubeClass>{
        // Instantiate resultnya, yaitu state-state yang dikunjungi
        const result : Array<MagicCubeClass> = [];

        // Membuat objek MagicCubeClass
        const currentMagicCube : MagicCubeClass = magicCube.clone();

        // Masukkan initial state ke dalam result
        result.push(currentMagicCube.clone());

        // Looping hingga menemukan local maximum atau global maximum
        while (true) {

            // console.log(currentMagicCube.getCurrentValue());

            // Mengambil best successor
            const bestSuccessorState = currentMagicCube.getBestSuccessor(currentMagicCube.getCurrentState());
            const bestSuccessorValue = currentMagicCube.objectiveFunction(bestSuccessorState);

            // Jika best successor value <= current magic cube value, break karena mencapai
            // local maximum atau global maximum
            if (bestSuccessorValue <= currentMagicCube.getCurrentValue()) {
                break
            } else {
                // Jika best successor value > current magic cube value, pindah ke successor tersebut
                currentMagicCube.setCurrentState(bestSuccessorState);
                currentMagicCube.setCurrentValue(bestSuccessorValue);

                // Masukkan state ke result
                result.push(currentMagicCube.clone());
            }
```

```
        }

        // console.log(result);

        return result;
    }
}
```

Hill-Climbing with Sideways Move

Hill-Climbing with Sideways Move adalah algoritma local search yang memiliki prosedur mirip dengan Steepest Ascent Hill-Climbing namun dengan konsiderasi suksesor yang diambil bisa sama dengan current objective function value dari kubus, tidak harus lebih besar. Hill-Climbing with Sideways Move juga belum tentu mengembalikan global maksimum, bisa jadi stuck di local maksimum.

Berikut adalah source code untuk kelas sideways move hill climbing.

```
import {MagicCubeClass} from "../magic-cube-class/magic-cube-class";

export class hcSidewaysMove{
    public static search(magicCube : MagicCubeClass, sidewaysMoveMax : number): Array<MagicCubeClass>{
        // Instantiate resultnya, yaitu state-state yang dikunjungi
        const result : Array<MagicCubeClass> = [];

        // Jumlah maksimum berapa kali sideways move
        // const sidewaysMoveMax : number = 5;
        let sidewaysMoveTimes : number = 0;

        // Membuat objek MagicCubeClass
        const currentMagicCube : MagicCubeClass = magicCube.clone();

        // Masukkan initial state ke dalam result
        result.push(currentMagicCube.clone());

        // Looping hingga menemukan local maximum (yang tidak memiliki neighbor best successor
        // dengan value yang sama) atau global maximum
        while (true) {

            // console.log(currentMagicCube.getCurrentValue());

            // Mengambil best successor
            const bestSuccessorState = currentMagicCube.getBestSuccessor(currentMagicCube.getCurrentState());
            const bestSuccessorValue = currentMagicCube.objectiveFunction(bestSuccessorState);

            // Jika best successor value < current magic cube value, break karena mencapai
            // local maximum atau global maximum
            if (bestSuccessorValue < currentMagicCube.getCurrentValue() || sidewaysMoveTimes == sidewaysMoveMax) {
```

```
                break
            } else {

                // Kalau melakukan sideways move, dapat dilakukan sejumlah maksimum kali
                if (bestSuccessorValue == currentMagicCube.getCurrentValue()) {
                    sidewaysMoveTimes++;
                } else {
                    sidewaysMoveTimes = 0;
                }

                // Jika best successor value >= current magic cube value, pindah ke successor tersebut
                currentMagicCube.setCurrentState(bestSuccessorState);
                currentMagicCube.setCurrentValue(bestSuccessorValue);

                // Masukkan state ke result
                result.push(currentMagicCube.clone());
            }
        }

        // console.log(result);

        return result;
    }
}
```

Random Restart Hill-Climbing

Random Restart Hill-Climbing adalah algoritma local search yang memiliki prosedur mirip dengan Steepest Ascent Hill-Climbing namun dengan pengulangan algoritma dilakukan sebanyak jumlah tertentu, dengan initial state yang berbeda-beda (random) dan diambil kubus dengan objective function value (local maksimum) yang paling tinggi, dengan harapan objective function value tersebut adalah global maksimum (yang berarti belum tentu mencapai global maksimum).

Berikut adalah source code untuk kelas random restart hill climbing.

```
import { MagicCubeClass } from "../magic-cube-class/magic-cube-class";
import { hcSteepestAscent } from "./hc-steepest-ascent";

export class hcRandomRestart {
    public static search(magicCube : MagicCubeClass, maxRestarts: number): Array<MagicCubeClass> {
        //Variabel untuk menyimpan steps dan value local optima terbaik dari semua restart
        let bestLocalOptimaStates: Array<MagicCubeClass> = [];
        let bestLocalOptimaValue = -1;

        for (let i = 0; i < maxRestarts; i++) {
            // Lakukan steepest ascent
            const currentLocalOptimaSteps = hcSteepestAscent.search(magicCube);
```

```
            const currentLocalOptimaCube = currentLocalOptimaSteps[currentLocalOptimaSteps.length - 1];
            const currentLocalOptimaValue =
currentLocalOptimaCube.objectiveFunction(currentLocalOptimaCube.getCurrentState());

            //Bandingkan hasil local optima sekarang dengan local optima terbaik sejauh ini
            if (currentLocalOptimaValue > bestLocalOptimaValue) {
                bestLocalOptimaStates = currentLocalOptimaSteps;
                bestLocalOptimaValue = currentLocalOptimaValue;
            }

            //Break jika global optima tercapai
            if (currentLocalOptimaValue === 109) {
                break;
            }
        }
        return bestLocalOptimaStates;
    }
}
```

Stochastic Hill Climbing

Stochastic Hill Climbing adalah algoritma local search yang dimulai dengan random initial state sama seperti algoritma-algoritma sebelumnya, namun iterasi dilakukan secara terbatas (jumlah yang telah ditentukan sebelumnya) dan untuk setiap iterasi, akan diambil suksesor secara random, dan jika suksesor mempunyai objective function value yang lebih besar, maka suksesor akan menjadi current dan digunakan untuk iterasi berikutnya. Stochastic Hill Climbing juga tidak menjamin global maksimum.

Berikut adalah source code untuk kelas stochastic hill climbing.

```
import {MagicCubeClass} from "../magic-cube-class/magic-cube-class";

export class hcStochastic {
    public static search(magicCube : MagicCubeClass): Array<MagicCubeClass> {
        //Inisialisasi current cube dan array result
        const currentCube : MagicCubeClass = magicCube.clone();
        const result : Array<MagicCubeClass> = [];

        //Masukkan current cube ke array result sebagai state pertama
        result.push(currentCube.clone());

        //Untuk sekarang, dilakukan 500 iterasi
        for (let i = 0; i < 500; i++) {
            const randomSuccessorState = currentCube.getRandomSuccessor(currentCube.getCurrentState());
            const randomSuccessorValue = currentCube.objectiveFunction(randomSuccessorState);

            //Bandingkan random successor dengan current cube
            if (randomSuccessorValue > currentCube.getCurrentValue()) {
```

```
                currentCube.setCurrentState(randomSuccessorState);
                currentCube.setCurrentValue(randomSuccessorValue);
            }

            //Tambahkan random successor ke dalam array result
            currentCube.setCurrentState(randomSuccessorState);
            currentCube.setCurrentValue(randomSuccessorValue);
            result.push(currentCube.clone());

            //Break jika global optima tercapai
            if (randomSuccessorValue === 109) {
                break;
            }
        }

        //Kembalikan array result
        return result;
    }
}
```

Simulated Annealing

Simulated Annealing adalah algoritma local search yang menyerupai Stochastic Hill-Climbing, namun adanya tambahan pertimbangan bagi kubus dengan objective function value yang lebih rendah, yaitu dengan menggunakan fungsi probabilitas $e^{\Delta E/T}$ dengan T adalah temperatur annealing yang berkurang secara berkala (misalnya dengan menggunakan cooling rate), ΔE adalah objective function value suksesor dikurangi objective function value current. Jika dari awal suksesor mempunyai ΔE yang positif, maka langsung suksesor menjadi current, jika tidak, akan dihitung menggunakan fungsi probabilitas, dan jika melebihi suatu threshold tertentu, neighbor akan menjadi current, jika tidak, akan di skip, lanjut ke iterasi berikutnya.

Pada implementasi algoritma simulated annealing, temperatur annealing berkurang secara linear menggunakan fungsi schedule yang menghitung temperatur berdasarkan cooling rate dan waktu yang sudah dilewati.

Berikut adalah source code untuk kelas simulated annealing.

```
import {MagicCubeClass} from "../magic-cube-class/magic-cube-class";
```

```
export class simulatedAnnealing{

    private static schedule(initTemperature:number, coolingRate:number, time:number):number{
        //IMPLEMENTASI FUNGSI SCHEDULE DI SINI
        // cooling ratenya pakai yg linear
        return (initTemperature - coolingRate * time);
    }

    private static calculateMoveProbability(deltaE:number, temperature:number):number{
        //IMPLEMENTASI FUNGSI PELUANG PINDAH DI SINI
        if (deltaE >= 0){return 1;}
        if (temperature <= 0){return 0;}
        return Math.exp(deltaE/temperature);
    }

    public static search(magicCube : MagicCubeClass): {
        cubeStates: Array<MagicCubeClass>,
        tValues: Array<number>,
        probabilities: Array<number>
    } {
        //IMPLEMENTASI ALGORITMA SIMULATED ANNEALING DI SINI

        // bisa diganti angkanya kalau diperlukan
        const initTemperature = 1;
        const coolingRate = 10;

        // variables or something
        const currentMagicCube : MagicCubeClass = magicCube.clone();
        const startTime = performance.now(); // start time in ms
        let time = 0;
        let temperature = initTemperature;
        const results : Array<MagicCubeClass> = [];
        const tValuesArray : Array<number> = [];
        const probabilitiesArray : Array<number> = [];

        // loop
        while (temperature > 0){
            time = (performance.now() - startTime) / 1000;
            temperature = simulatedAnnealing.schedule(initTemperature, coolingRate, time);

            const randomSuccessorState = currentMagicCube.getRandomSuccessor(currentMagicCube.getCurrentState());
            const randomSuccessorValue = currentMagicCube.objectiveFunction(randomSuccessorState);

            const probability = simulatedAnnealing.calculateMoveProbability(randomSuccessorValue -
currentMagicCube.getCurrentValue(), temperature);

            if (randomSuccessorValue >= currentMagicCube.getCurrentValue()){
                currentMagicCube.setCurrentState(randomSuccessorState);
                currentMagicCube.setCurrentValue(randomSuccessorValue);
            } else {
                // valuenya lebih kecil, jadi di random
                const randomNumber = Math.random();
                if (randomNumber < probability){
                    currentMagicCube.setCurrentState(randomSuccessorState);
                    currentMagicCube.setCurrentValue(randomSuccessorValue);
                }
            }
```

```
            // tambahin hasil iterasi ini ke array
            results.push(currentMagicCube.clone());
            tValuesArray.push(time);
            probabilitiesArray.push(probability);
        }

        return {
            cubeStates: results,
            tValues: tValuesArray,
            probabilities: probabilitiesArray
        };
    }
}
```

Genetic Algorithm

Genetic Algorithm adalah algoritma local search, yang dimulai dengan pengambilan k buah randomized initial state, yaitu kubus yang sudah diisi angka 1 - 125 secara acak, kemudian dihitung fitness valuenya, yang pada kasus ini adalah objective function valuenya. Berdasarkan fitness value yang dijadikan angka probabilitas, dilakukan pengambilan beberapa “parent state” secara acak. Kemudian, setiap 2 parent, akan dilakukan crossover, yaitu penukaran sebagian dari kubus berdasarkan suatu crossover point tertentu. Setelah dilakukan crossover, akan dilakukan mutasi, yaitu perubahan elemen dari kubus secara acak, menjadi angka yang acak pula. Seluruh state hasil mutasi akan dihitung fitness valuenya lagi lalu akan kembali melalui proses genetic algorithm, diulang sebanyak jumlah iterasi tertentu. Genetic Algorithm tidak menjamin global maksimum.

Berikut adalah source code untuk kelas genetic algorithm.

```
import {MagicCubeClass} from "../magic-cube-class/magic-cube-class";
import {Genome} from "../genome/genome";

export class geneticAlgorithm {
    public static search(magicCube : MagicCubeClass): Array<MagicCubeClass> {
        const genomeAmt = 50; //banyak genome
        const mutationProb = 0.8; //peluang mutasi
        const maxIteration = 1000;
        const results : Array<MagicCubeClass> = [];
        let it = 1;
        //1. isi daftar genome dengan genome acak
        const genomeList : Array<Genome> = Array.from({length: genomeAmt}, () => {
            magicCube.setCurrentState(magicCube.getRandomInitState());
```

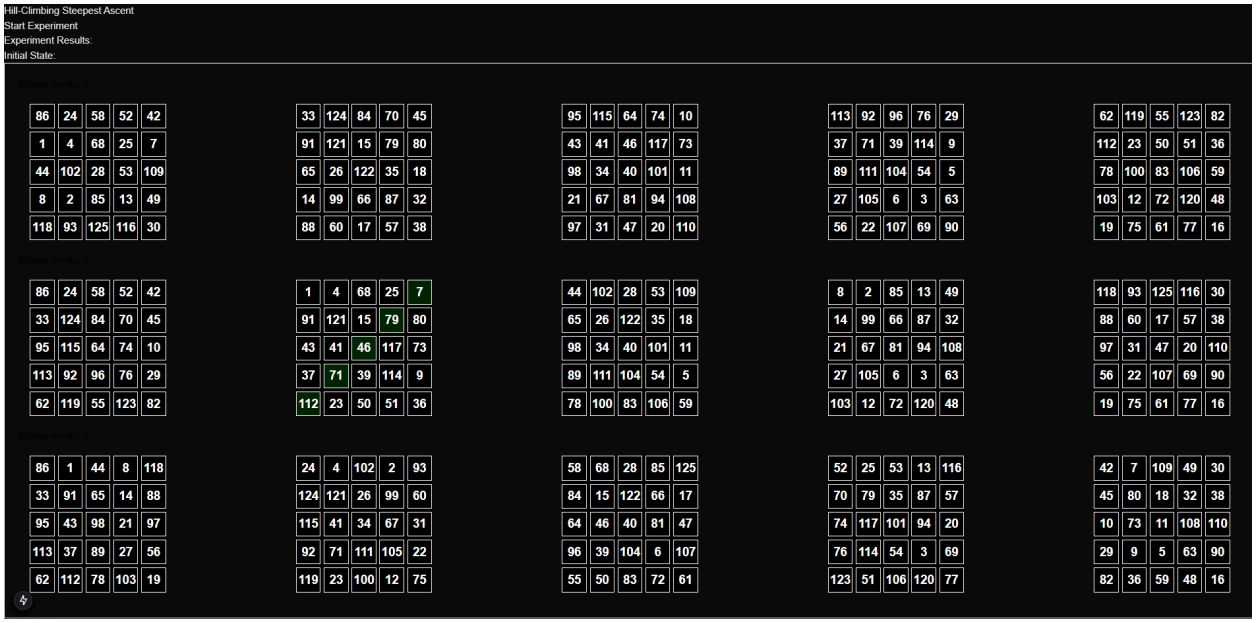
```
            return new Genome(magicCube);
        });

        //Ulangi sampai solusinya ketemu
        while(true){
            const fitnessVals = genomeList.map((genome, index) => {
                const cube = genome.genomeToCube(genome.getGenomeStrand());
                const fitness = magicCube.objectiveFunction(cube);
                return fitness;
            });

            const fitnessMax = Math.max(...fitnessVals);
            const bestGenomeIdx = fitnessVals.indexOf(fitnessMax);
            const bestGenome = genomeList[bestGenomeIdx];
            const bestCube = new MagicCubeClass();
            bestCube.setCurrentState(bestGenome.genomeToCube(bestGenome.getGenomeStrand()));
            bestCube.setCurrentValue(magicCube.objectiveFunction(bestCube.getCurrentState()));
            results.push(bestCube);
            if(fitnessMax === 109 ||it>=maxIteration){

                return results;
            }

            const chosenGenomes = this.selectGenomes(genomeList, fitnessVals);
            const newGenomeList = this.createNextGen(chosenGenomes,mutationProb);

            genomeList.length = 0;
            genomeList.push(...newGenomeList);
            it++;
        }
    }
    private static selectGenomes(genomeList : Array<Genome>, fitnessScores: Array<number>):Array<Genome>  {
        //method untuk pilih daftar genome menggunakan roulette wheel
           const fitnessSum = fitnessScores.reduce((sum:number, fitness) => sum + fitness, 0); //jumlah seluruh fitness
value
        const probList = fitnessScores.map(scores => scores/fitnessSum); //daftar probabilitas setiap genome

        const chosenGenomes : Array<Genome> = [];
        for(let i = 0; i < genomeList.length; i++) {
            const chosen = this.probabilityWheelSelect(genomeList,probList); //pilih genome menggunakan probability wheel
            chosenGenomes.push(chosen);
        }

        return chosenGenomes;
    }
    private static probabilityWheelSelect(genomeList:Genome[], probList:number[]):Genome{
        //method probability menggunakan roulette wheel
        const randomVal = Math.random(); //pilih angka random di antara 0 dan 1
        let cumulativeProb = 0; //probabilitas kumulatif

        for(let i = 0; i<genomeList.length; i++) {

            cumulativeProb += probList[i]; //tambahkan probabilitas ke i

            if(randomVal < cumulativeProb){ //jika nilai random yang terpilih di bawah probabilitas kumulatif saat ini
                return genomeList[i]; //pilih genome ke i
            }
        }
```

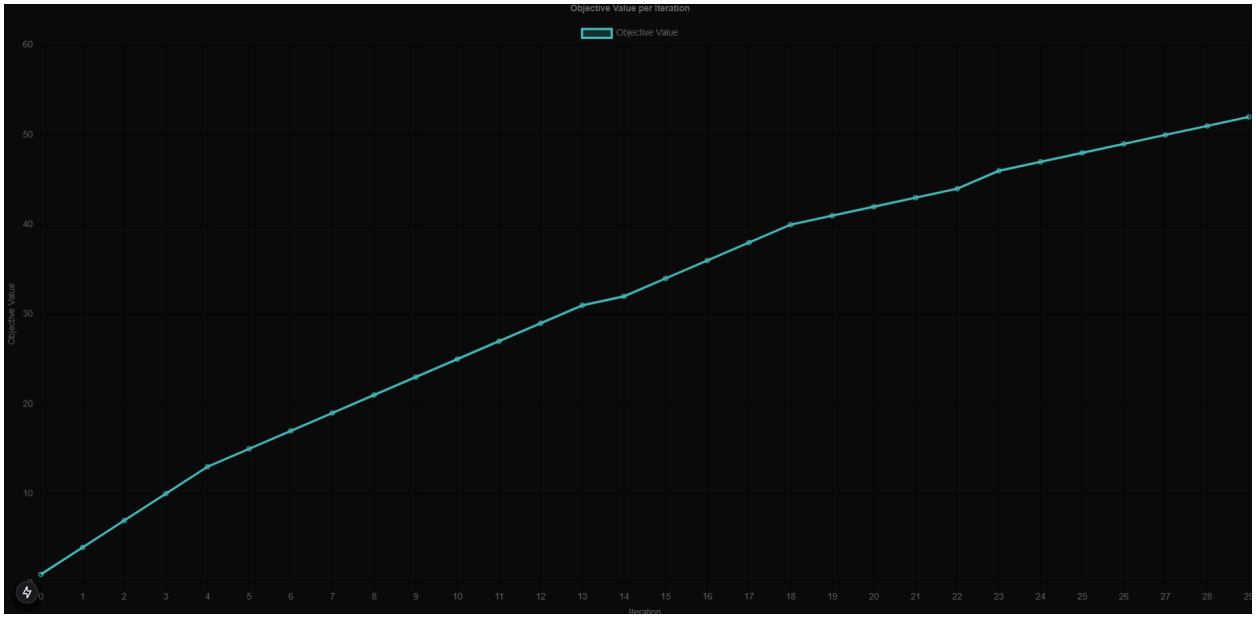
```
        return genomeList[genomeList.length-1]; //pilih genome terakhir jika probabilitas kumulatif akhir tidak bulat menjadi 1
    }

    private static createNextGen(chosenGenomes : Genome[], mutationProb:number):Genome[]{
        //crossover dan mutasi semua genome saat ini
        const nextGen: Genome[] = [];
        for(let i = 0; i<chosenGenomes.length; i+=2){
            //pilih 1 pasangan genome
            const par1 = chosenGenomes[i];
            const par2 = chosenGenomes[i+1] || chosenGenomes[0]; //pilih genome pertama lagi jika banyak genomenya ganjil

            const [child1, child2] = this.crossover(par1,par2); //crossover pasangan genome tadi
            this.mutate(child1,mutationProb); //mutasi anak pertama
            this.mutate(child2,mutationProb); //mutasi anak kedua

            nextGen.push(child1,child2); //masukkan daftar anak baru ke generasi berikutnya
        }
        return nextGen;
    }
    private static crossover(par1:Genome, par2:Genome):[Genome,Genome]{
        //kawin silang antara dua buah gen
        const crossPoint = Math.floor(Math.random() * 125); //pilih titik crossover acak
        const gen1 = par1.getGenomeStrand();
        const gen2 = par2.getGenomeStrand();

        for(let i = crossPoint; i<125; i++){
            //lakukan kawin silang dimulai dari titik crossover sampai akhir dari kedua genome
            [gen1[i], gen2[i]] = [gen2[i], gen1[i]];
        }
        let newgen1 = new Genome();
        newgen1.setGenomeStrand(gen1);
        let newgen2 = new Genome();
        newgen2.setGenomeStrand(gen2);
        return [newgen1, newgen2];
    }

    private static mutate(gen:Genome, mutationProb:number):void{
        const genomeStrand = gen.getGenomeStrand();
        for(let i = 0; i<genomeStrand.length; i++){
            if(Math.random() <mutationProb){
                const swapIdx = Math.floor(Math.random() * 125);
                [genomeStrand[i],genomeStrand[swapIdx]] = [genomeStrand[swapIdx],genomeStrand[i]];
            }
        }
        gen.setGenomeStrand(genomeStrand);
    }
}
```

Hasil Eksperimen dan Analisis

Steepest Ascent Hill-Climbing

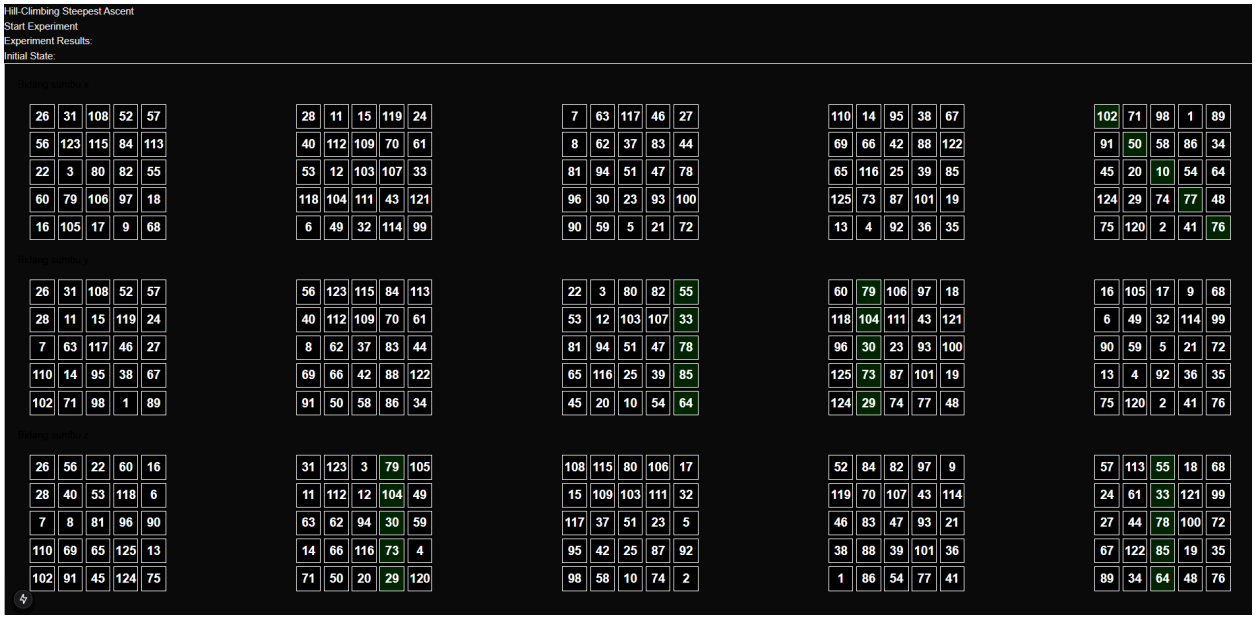
Percobaan 1:

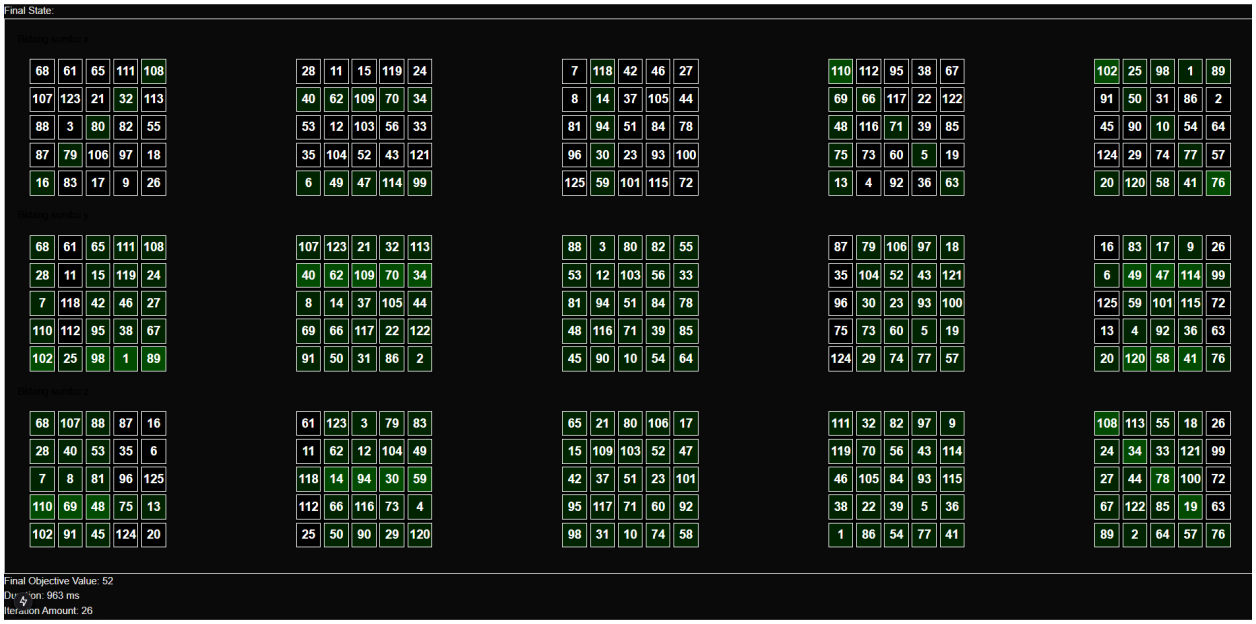


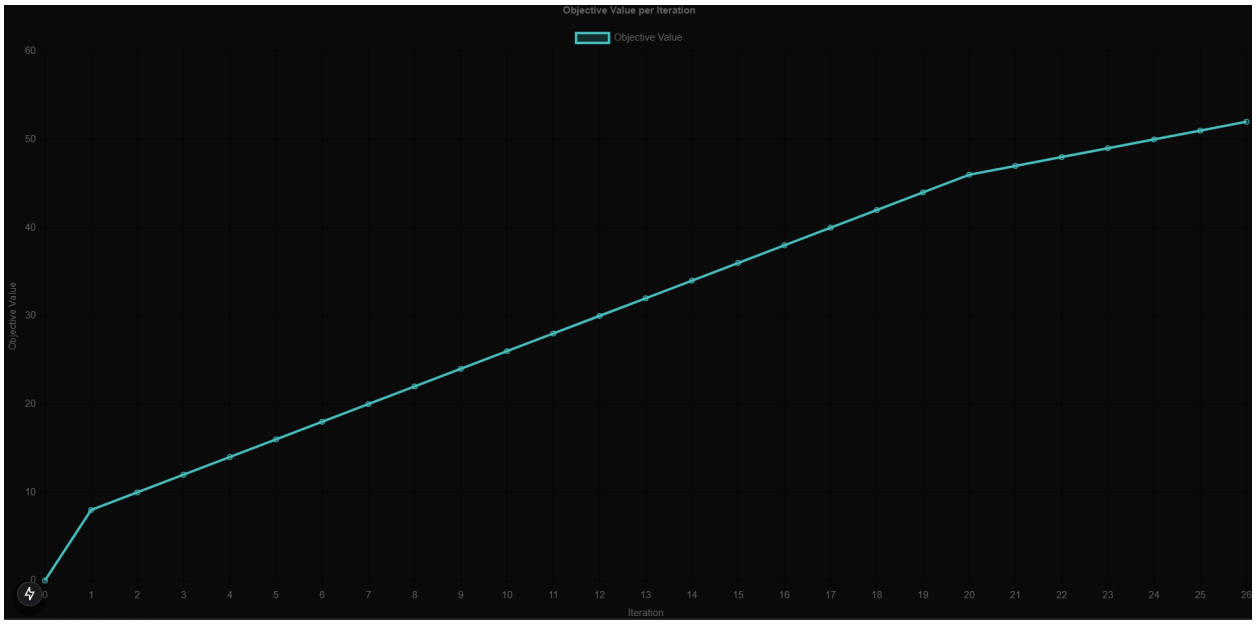


Dalam percobaan ini, didapat hasil yang cukup baik, yaitu state akhir dengan objective value sebesar 52. Hasil ini juga didapat dalam waktu yang cukup cepat, yaitu sekitar 1 detik. Terdapat sebanyak 29 iterasi. Local optima berhasil dicapai.

Percobaan 2:

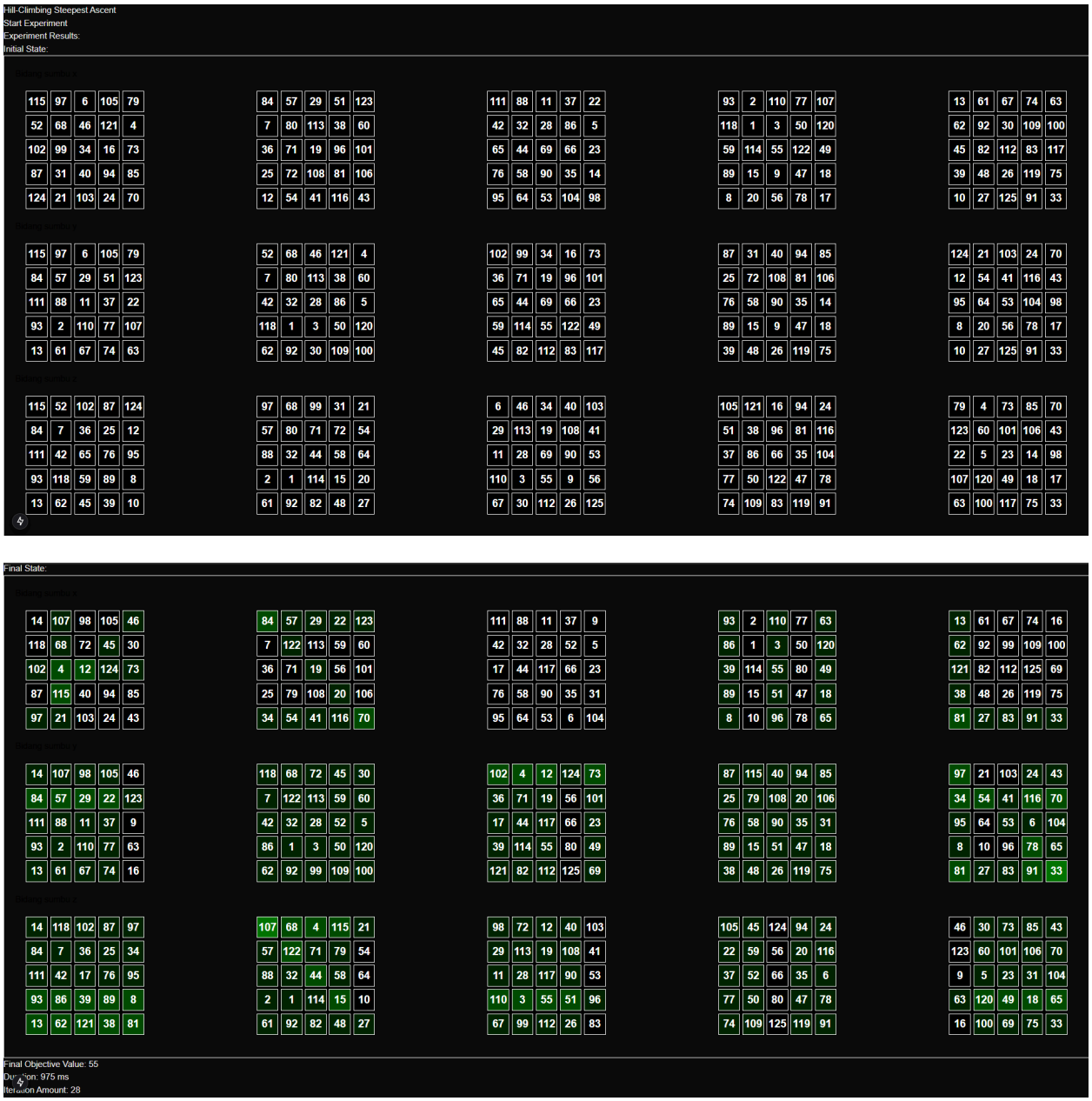


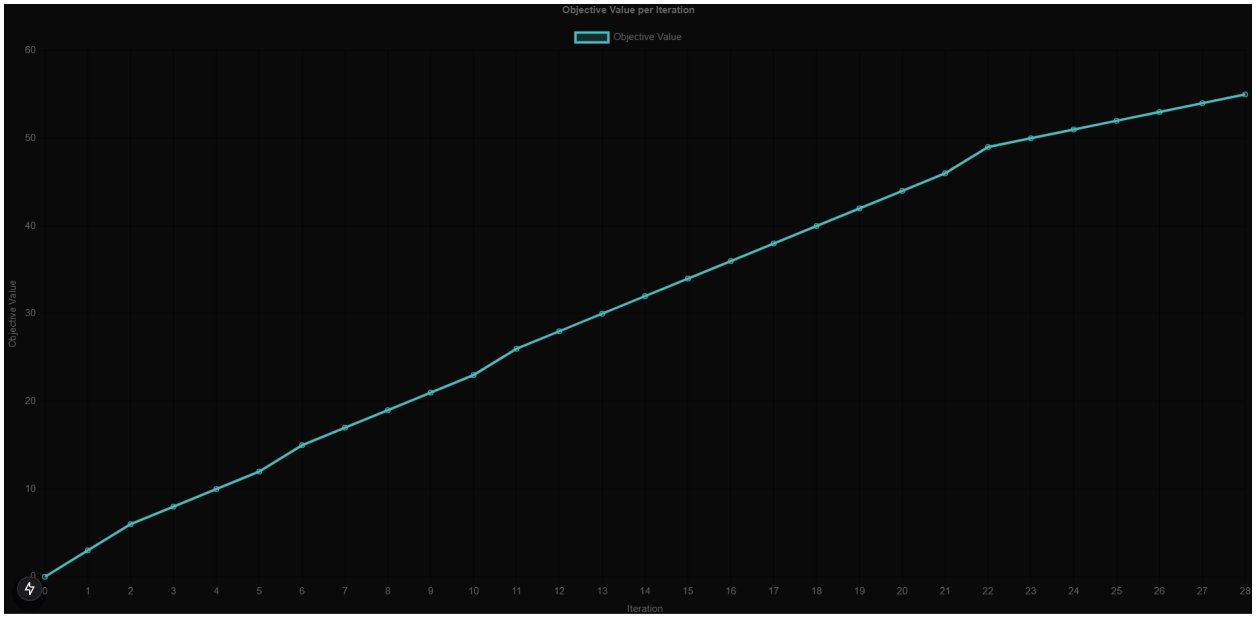




Dalam percobaan ini, didapat hasil yang cukup baik, yaitu state akhir dengan objective value sebesar 52. Hasil ini juga didapat dalam waktu yang cukup cepat, yaitu sekitar 1 detik. Terdapat sebanyak 26 iterasi. Local optima berhasil dicapai.

Percobaan 3:


Hill-Climbing Steepest Ascent
Start Experiment
Experiment Results:
Initial State:
Solving Sudoku 4
115 97 6 105 79
52 68 46 121 4
102 99 34 16 73
87 31 40 94 85
124 21 103 24 70
84 57 29 51 123
7 80 113 38 60
36 71 19 96 101
25 72 108 81 106
12 54 41 116 43
111 88 11 37 22
42 32 28 86 5
65 44 69 66 23
76 58 90 35 14
95 64 53 104 98
93 2 110 77 107
118 1 3 50 120
59 114 55 122 49
89 15 9 47 18
8 20 56 78 17
13 61 67 74 63
62 92 30 109 100
45 82 112 83 117
39 48 26 119 75
10 27 125 91 33
Solving Sudoku 3
115 97 6 105 79
84 57 29 51 123
111 88 11 37 22
93 2 110 77 107
13 61 67 74 63
52 68 46 121 4
7 80 113 38 60
42 32 28 86 5
118 1 3 50 120
62 92 30 109 100
102 99 34 16 73
36 71 19 96 101
65 44 69 66 23
59 114 55 122 49
45 82 112 83 117
87 31 40 94 85
25 72 108 81 106
76 58 90 35 14
89 15 9 47 18
39 48 26 119 75
124 21 103 24 70
12 54 41 116 43
95 64 53 104 98
8 20 56 78 17
10 27 125 91 33
Solving Sudoku 2
115 52 102 87 124
84 7 36 25 12
111 42 65 76 95
93 118 59 89 8
13 62 45 39 10
97 68 99 31 21
57 80 71 72 54
88 32 44 58 64
2 1 114 15 20
61 92 82 48 27
6 46 34 40 103
29 113 19 108 41
11 28 69 90 53
110 3 55 9 56
67 30 112 26 125
105 121 16 94 24
51 38 96 81 116
37 86 66 35 104
77 50 122 47 78
74 109 83 119 91
79 4 73 85 70
123 60 101 106 43
22 5 23 14 98
107 120 49 18 17
63 100 117 75 33
Final State:
Solving Sudoku 4
14 107 98 105 46
118 68 72 45 30
102 4 12 124 73
87 115 40 94 85
97 21 103 24 43
84 57 29 22 123
7 122 113 59 60
36 71 19 56 101
25 79 108 20 106
34 54 41 116 70
111 88 11 37 9
42 32 28 52 5
17 44 117 66 23
76 58 90 35 31
95 64 53 6 104
93 2 110 77 63
86 1 3 50 120
39 114 55 80 49
89 15 51 47 18
8 10 96 78 65
13 61 67 74 16
62 92 99 109 100
121 82 112 125 69
38 48 26 119 75
81 27 83 91 33
Solving Sudoku 3
14 107 98 105 46
84 57 29 22 123
111 88 11 37 9
93 2 110 77 63
13 61 67 74 16
118 68 72 45 30
7 122 113 59 60
42 32 28 52 5
86 1 3 50 120
62 92 99 109 100
102 4 12 124 73
36 71 19 56 101
17 44 117 66 23
39 114 55 80 49
121 82 112 125 69
87 115 40 94 85
25 79 108 20 106
76 58 90 35 31
89 15 51 47 18
38 48 26 119 75
97 21 103 24 43
34 54 41 116 70
95 64 53 6 104
8 10 96 78 65
81 27 83 91 33
Solving Sudoku 2
14 118 102 87 97
84 7 36 25 34
111 42 17 76 95
93 86 39 89 8
13 62 121 38 81
107 68 4 115 21
57 122 71 79 54
88 32 44 58 64
2 1 114 15 10
61 92 82 48 27
98 72 12 40 103
29 113 19 108 41
11 28 117 90 53
110 3 55 51 96
67 99 112 26 83
105 45 124 94 24
22 59 56 20 116
37 52 66 35 6
77 50 80 47 78
74 109 125 119 91
46 30 73 85 43
123 60 101 106 70
9 5 23 31 104
63 120 49 18 65
16 100 69 75 33
Final Objective Value: 55
Duration: 975 ms
Iteration Amount: 28



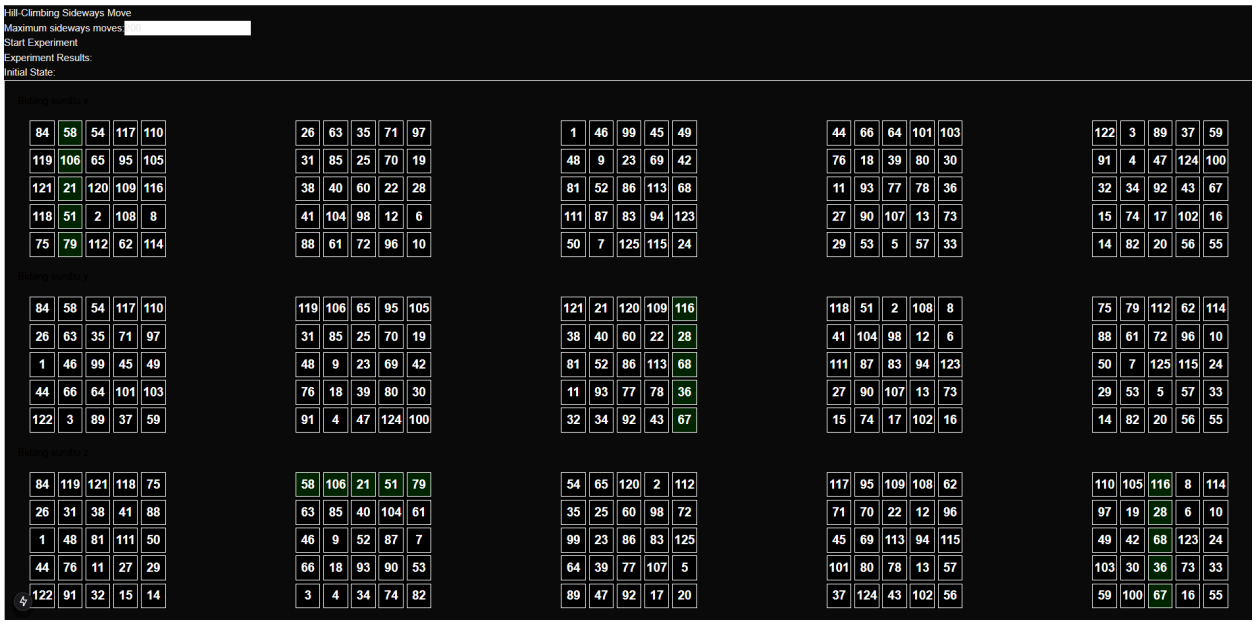
Dalam percobaan ini, didapat hasil yang cukup baik, yaitu state akhir dengan objective value sebesar 55. Hasil ini juga didapat dalam waktu yang cukup cepat, yaitu sekitar 1 detik. Terdapat sebanyak 28 iterasi. Local optima berhasil dicapai.

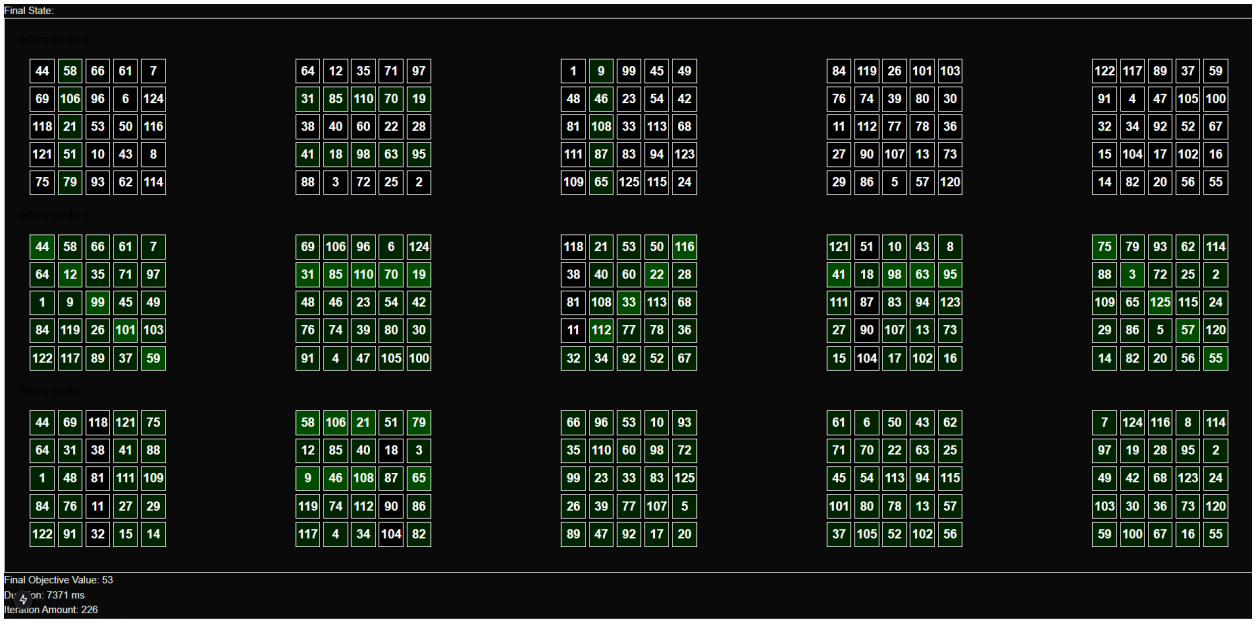
Hill-Climbing with Sideways Move

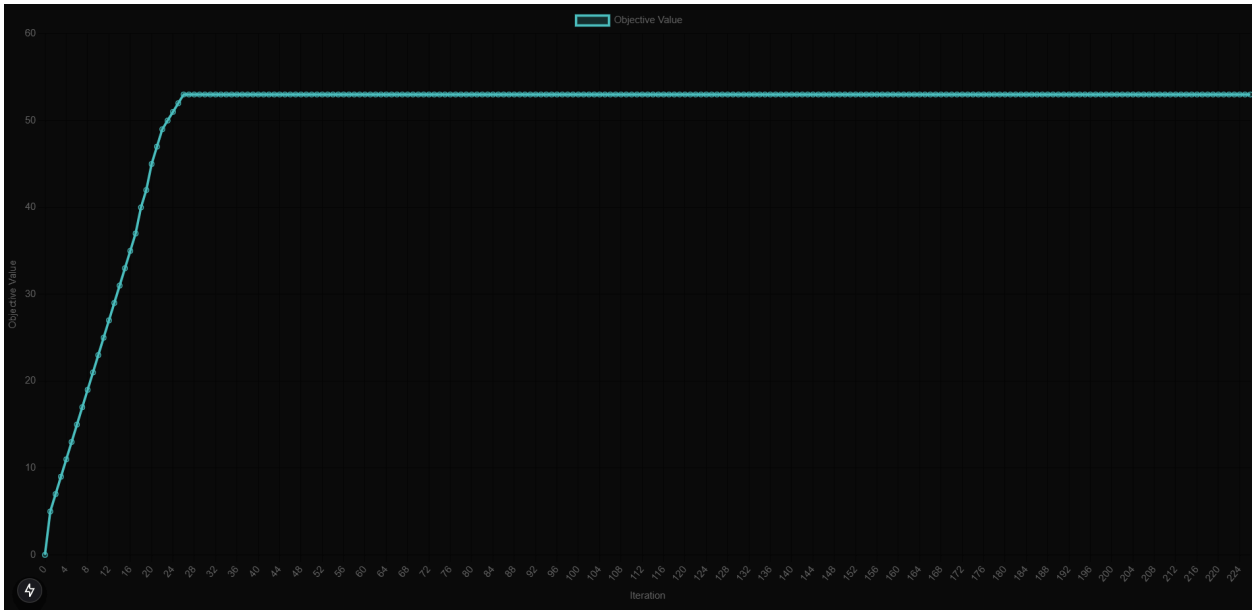
Percobaan 1:

Dalam percobaan ini, didapat hasil yang cukup baik, yaitu state akhir dengan objective value sebesar 51. Hasil ini juga didapat dalam waktu yang agak cepat, yaitu sekitar 5 detik. Terdapat sebanyak 126 iterasi. Local optima diraih pada iterasi ke-26. Meski demikian, local optima tidak bisa ditinggalkan sampai iterasi terakhir.

Percobaan 2:

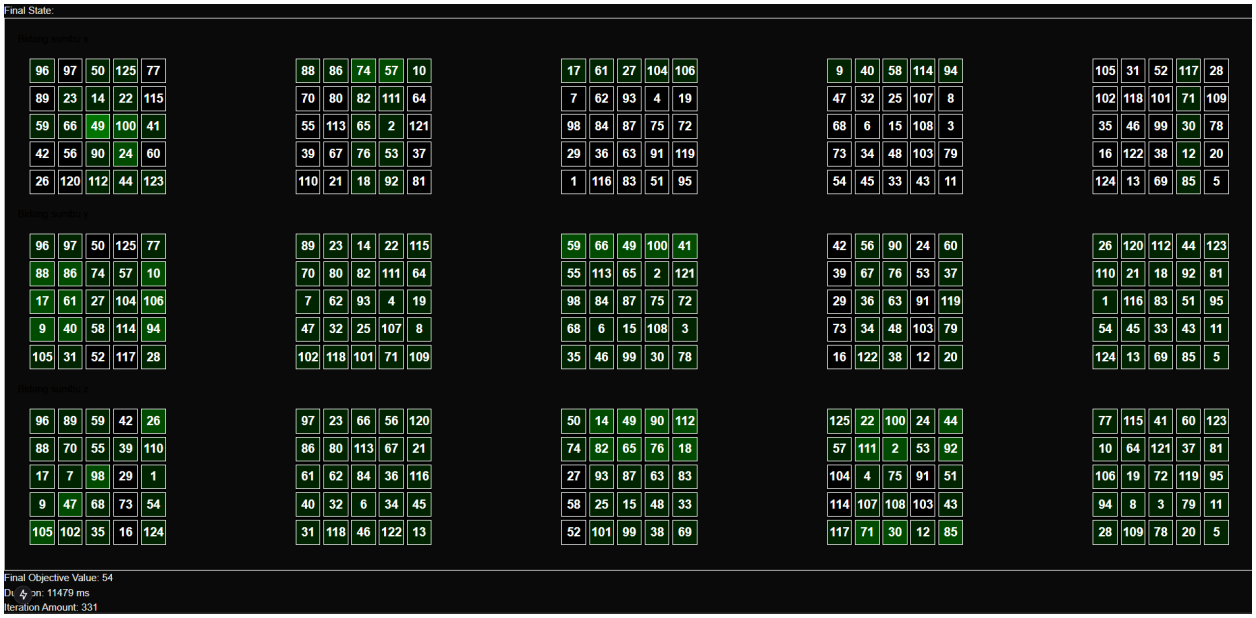

Hill-Climbing Sideways Move
Maximum sideways moves:
Start Experiment
Experiment Results:
Initial State:
Solving sudoku 1
84 58 54 117 110
119 106 65 95 105
121 21 120 109 116
118 51 2 108 8
75 79 112 62 114
26 63 35 71 97
31 85 25 70 19
38 40 60 22 28
41 104 98 12 6
88 61 72 96 10
1 46 99 45 49
48 9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


Final State:
Solving sudoku 1
44 58 66 61 7
69 106 96 6 124
118 21 53 50 116
121 51 10 43 8
75 79 93 62 114
64 12 35 71 97
31 85 110 70 19
38 40 60 22 28
41 18 98 63 95
88 3 72 25 2
1 9 99 45 49
48 46 23 54 42
81 108 33 113 68
111 87 83 94 123
109 65 125 115 24
84 119 26 101 103
76 74 39 80 30
11 112 77 78 36
27 90 107 13 73
29 86 5 57 120
122 117 89 37 59
91 4 47 105 100
32 34 92 52 67
15 104 17 102 16
14 82 20 56 55
Solving sudoku 2
44 58 66 61 7
64 12 35 71 97
1 9 99 45 49
84 119 26 101 103
122 117 89 37 59
69 106 96 6 124
31 85 110 70 19
48 46 23 54 42
76 74 39 80 30
91 4 47 105 100
118 21 53 50 116
38 40 60 22 28
81 108 33 113 68
11 112 77 78 36
32 34 92 52 67
121 51 10 43 8
41 18 98 63 95
111 87 83 94 123
27 90 107 13 73
15 104 17 102 16
75 79 93 62 114
88 3 72 25 2
109 65 125 115 24
29 86 5 57 120
14 82 20 56 55
Solving sudoku 3
44 69 118 121 75
64 31 38 41 88
1 48 81 111 109
84 76 11 27 29
122 91 32 15 14
58 106 21 51 79
12 85 40 18 3
9 46 108 87 65
119 74 112 90 86
117 4 34 104 82
66 96 53 10 93
35 110 60 98 72
99 23 33 83 125
26 39 77 107 5
89 47 92 17 20
61 6 50 43 62
71 70 22 63 25
45 54 113 94 115
101 80 78 13 57
37 105 52 102 56
7 124 116 8 114
97 19 28 95 2
49 42 68 123 24
103 30 36 73 120
59 100 67 16 55
Final Objective Value: 53
Duration: 7371 ms
Iteration Amount: 226



Dalam percobaan ini, didapat hasil yang cukup baik, yaitu state akhir dengan objective value sebesar 53. Hasil ini juga didapat dalam waktu yang agak cepat, yaitu sekitar 7 detik. Terdapat sebanyak 226 iterasi. Local optima diraih pada iterasi ke-26. Meski demikian, local optima tidak bisa ditinggalkan sampai iterasi terakhir.

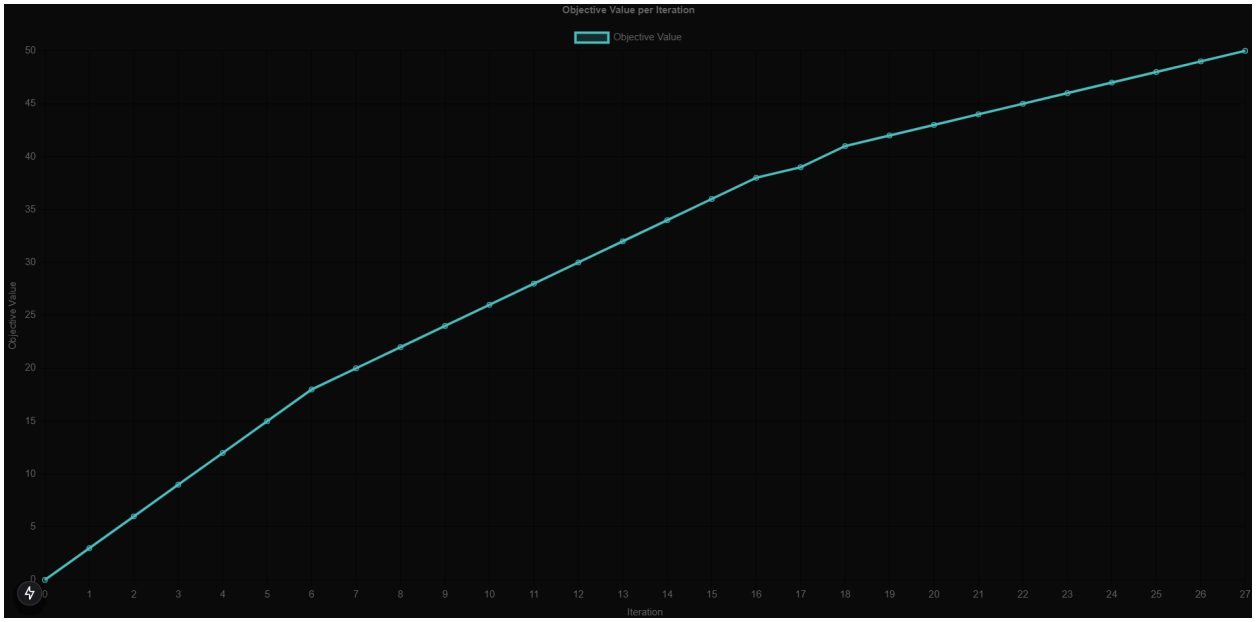
Percobaan 3:

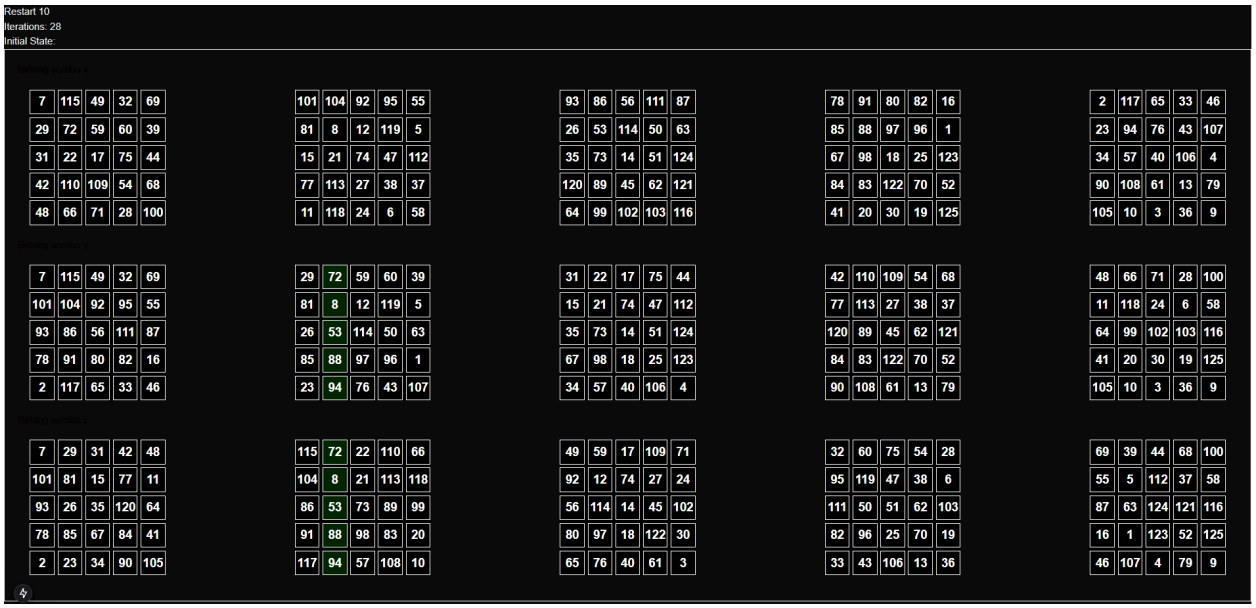


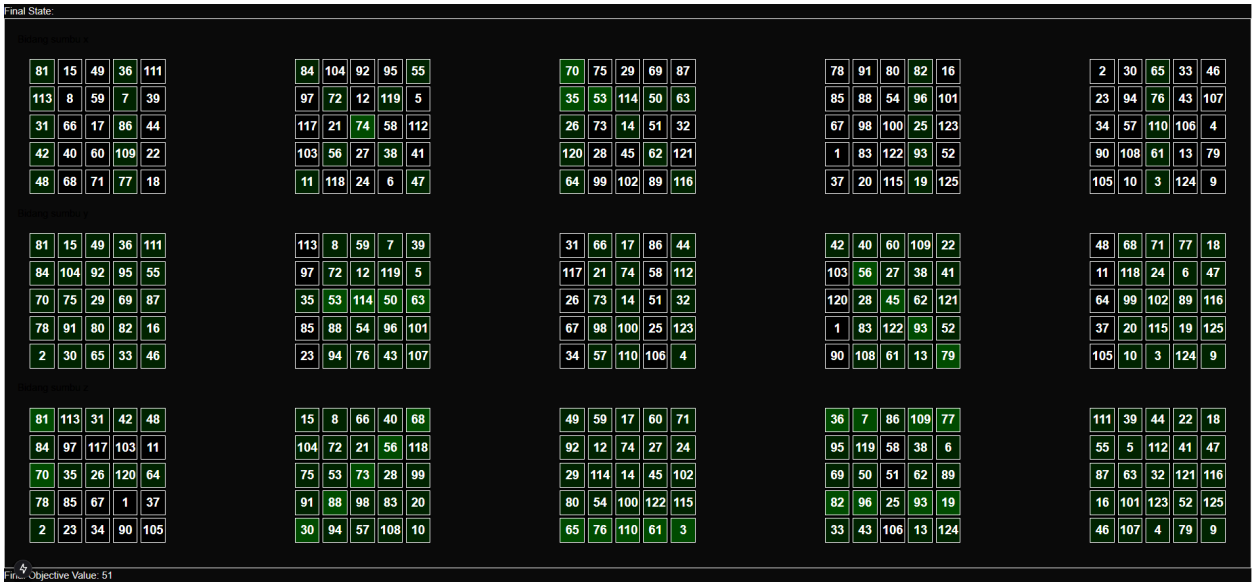
Dalam percobaan ini, didapat hasil yang cukup baik, yaitu state akhir dengan objective value sebesar 54. Hasil ini juga didapat dalam waktu yang agak lama, yaitu sekitar 11 detik. Terdapat sebanyak 331 iterasi. Local optima diraih pada iterasi ke-30. Meski demikian, local optima tidak bisa ditinggalkan sampai iterasi terakhir.

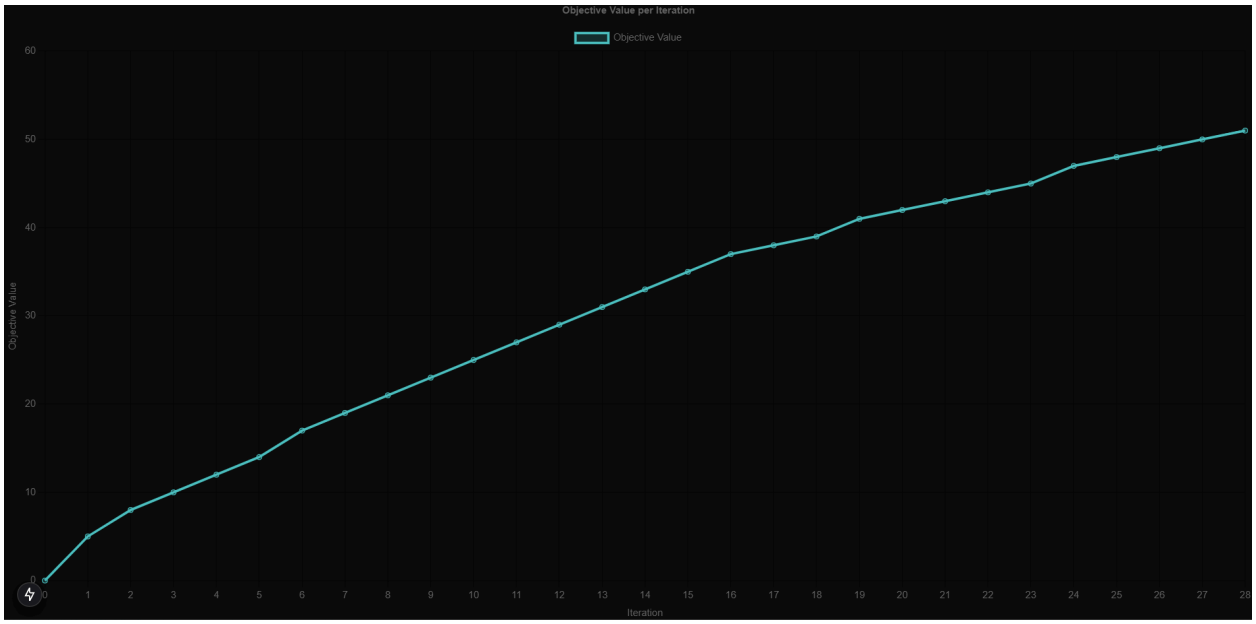
Random Restart Hill-Climbing

Percobaan 1:


Objective Value per Iteration
Objective Value
Objective Value
Iteration

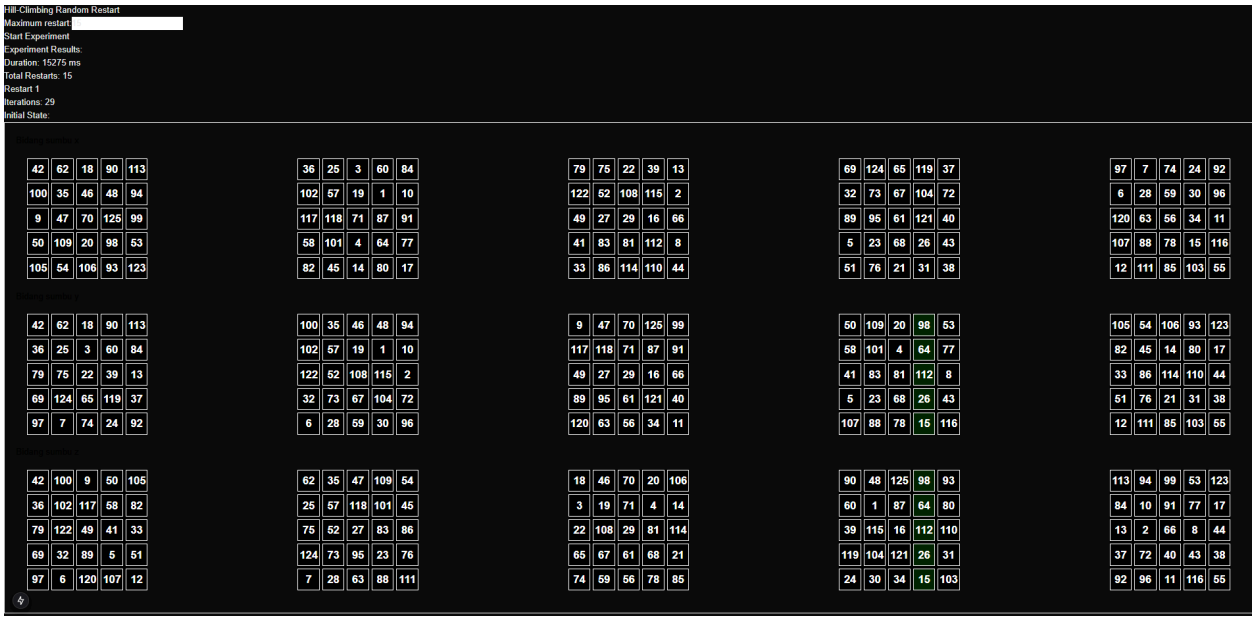

Restart 10
Iterations: 28
Initial State:
7 115 49 32 69
29 72 59 60 39
31 22 17 75 44
42 110 109 54 68
48 66 71 28 100
101 104 92 95 55
81 8 12 119 5
15 21 74 47 112
77 113 27 38 37
11 118 24 6 58
93 86 56 111 87
26 53 114 50 63
35 73 14 51 124
120 89 45 62 121
64 99 102 103 116
78 91 80 82 16
85 88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

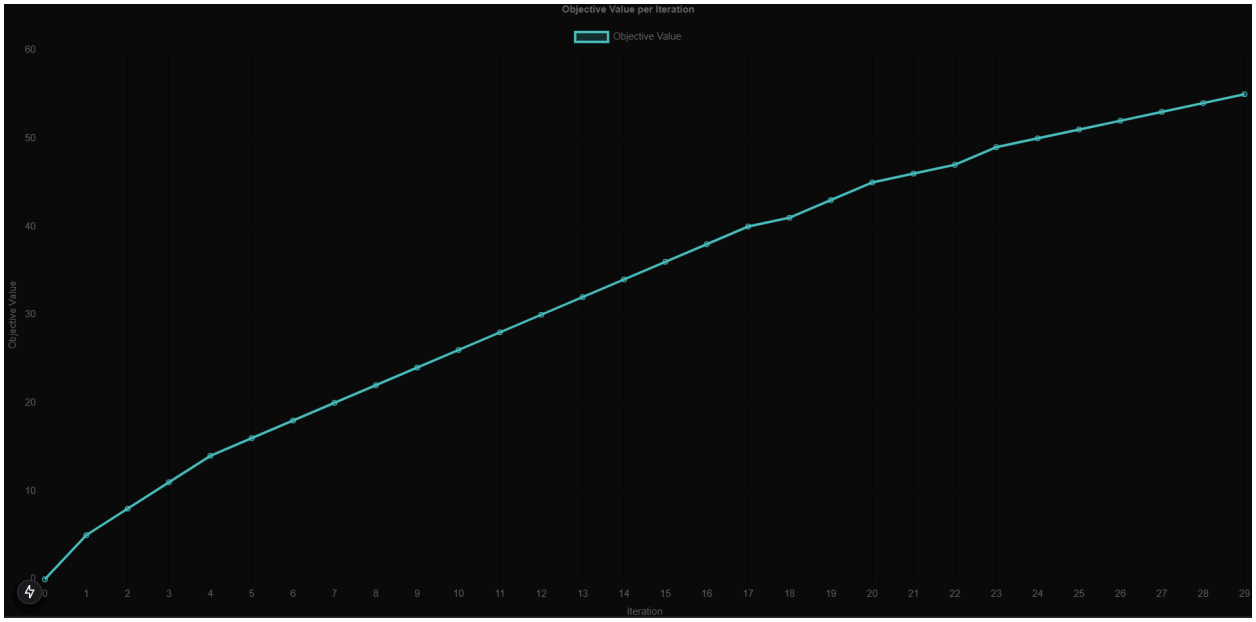




Dalam percobaan ini, dilakukan 10 kali restart, didapat hasil yang cukup baik, yaitu state akhir dengan objective value kurang lebih 50 untuk setiap restart. Hasil ini juga didapat dalam waktu yang cukup cepat, yaitu sekitar 10 detik total, atau 1 detik per restart. Terdapat sebanyak kurang lebih 20-30 iterasi per restart. Local optima berhasil diraih.

Percobaan 2:


Hill-Climbing Random Restart
Maximum restart:
Start Experiment
Experiment Results:
Duration: 15275 ms
Total Restarts: 15
Restart 1
Iterations: 29
Initial State:
42 62 18 90 113
100 35 46 48 94
9 47 70 125 99
50 109 20 98 53
105 54 106 93 123
36 25 3 60 84
102 57 19 1 10
117 118 71 87 91
58 101 4 64 77
82 45 14 80 17
79 75 22 39 13
122 52 108 115 2
49 27 29 16 66
41 83 81 112 8
33 86 114 110 44
69 124 65 119 37
32 73 67 104 72
89 95 61 121 40
5 23 68 26 43
51 76 21 31 38
97 7 74 24 92
6 28 59 30 96
120 63 56 34 11
107 88 78 15 116
12 111 85 103 55
42 62 18 90 113
36 25 3 60 84
79 75 22 39 13
69 124 65 119 37
97 7 74 24 92
100 35 46 48 94
102 57 19 1 10
122 52 108 116 2
32 73 67 104 72
6 28 59 30 96
9 47 70 125 99
117 118 71 87 91
49 27 29 16 66
89 95 61 121 40
120 63 56 34 11
50 109 20 98 53
58 101 4 64 77
41 83 81 112 8
5 23 68 26 43
107 88 78 15 116
105 54 106 93 123
82 45 14 80 17
33 86 114 110 44
51 76 21 31 38
12 111 85 103 55
42 100 9 50 105
36 102 117 58 82
79 122 49 41 33
69 32 89 5 51
97 6 120 107 12
62 35 47 109 54
25 57 118 101 45
75 52 27 83 86
124 73 95 23 76
7 28 63 88 111
18 46 70 20 106
3 19 71 4 14
22 108 29 81 114
65 67 61 68 21
74 59 56 78 85
90 48 125 98 93
60 1 87 64 80
39 115 16 112 110
119 104 121 26 31
24 30 34 15 103
113 94 99 53 123
84 10 91 77 17
13 2 66 8 44
37 72 40 43 38
92 96 11 116 55
Final State:
34 84 86 90 92
100 53 46 115 73
9 12 70 125 99
104 20 59 98 52
18 54 81 102 123
36 124 3 43 60
55 105 19 1 2
117 118 71 110 91
58 101 4 64 77
82 45 14 80 17
79 75 87 39 13
122 35 108 40 10
49 27 57 16 66
41 83 106 112 8
33 29 114 121 44
69 25 65 119 37
32 94 67 50 72
89 95 61 22 48
5 23 68 26 62
120 111 21 31 38
97 7 74 24 113
6 28 30 109 96
51 63 56 42 11
107 88 78 15 116
47 76 85 103 93
34 84 86 90 92
36 124 3 43 60
79 75 87 39 13
69 25 65 119 37
97 7 74 24 113
100 53 46 115 73
55 105 19 1 2
122 35 108 40 10
32 94 67 50 72
6 28 30 109 96
9 12 70 125 99
117 118 71 110 91
49 27 57 16 66
89 95 61 22 48
51 63 56 42 11
104 20 59 98 52
58 101 4 64 77
41 83 106 112 8
5 23 68 26 62
107 88 78 15 116
18 54 81 102 123
82 45 14 80 17
33 29 114 121 44
120 111 21 31 38
47 76 85 103 93
34 100 9 104 18
36 55 117 58 82
79 122 49 41 33
69 32 89 5 120
97 6 51 107 47
84 53 12 20 54
124 105 118 101 45
75 35 27 83 29
25 94 95 23 111
7 28 63 88 76
86 46 70 59 81
3 19 71 4 14
87 108 57 106 114
65 67 61 68 21
74 30 56 78 85
90 115 125 98 102
43 1 110 64 80
39 40 16 112 121
119 50 22 26 31
24 109 42 15 103
92 73 99 52 123
60 2 91 77 17
13 10 66 8 44
37 72 48 62 38
113 96 11 116 93
Final Objective Value: 55



Restart 15
Iterations: 28
Initial State:

Solved state x

103	43	94	108	70
56	46	124	65	2
71	23	61	95	32
35	49	92	105	72
100	30	14	123	52

88	110	6	5	80
77	34	53	3	111
73	11	93	125	102
84	75	9	106	45
82	59	51	36	104

60	37	99	15	13
107	114	121	85	24
117	101	79	74	67
47	54	8	120	87
116	44	17	39	33

20	69	50	58	10
109	48	55	12	83
16	26	66	86	98
4	25	113	91	115
28	7	118	41	38

97	78	63	31	96
112	119	68	27	76
40	18	1	89	81
64	19	90	122	62
29	42	57	21	22

Solved state y

103	43	94	108	70
88	110	6	5	80
60	37	99	15	13
20	69	50	58	10
97	78	63	31	96

56	46	124	65	2
77	34	53	3	111
107	114	121	85	24
109	48	55	12	83
112	119	68	27	76

71	23	61	95	32
73	11	93	125	102
117	101	79	74	67
16	26	66	86	98
40	18	1	89	81

35	49	92	105	72
84	75	9	106	45
47	54	8	120	87
4	25	113	91	115
64	19	90	122	62

100	30	14	123	52
82	59	51	36	104
116	44	17	39	33
28	7	118	41	38
29	42	57	21	22

Solved state z

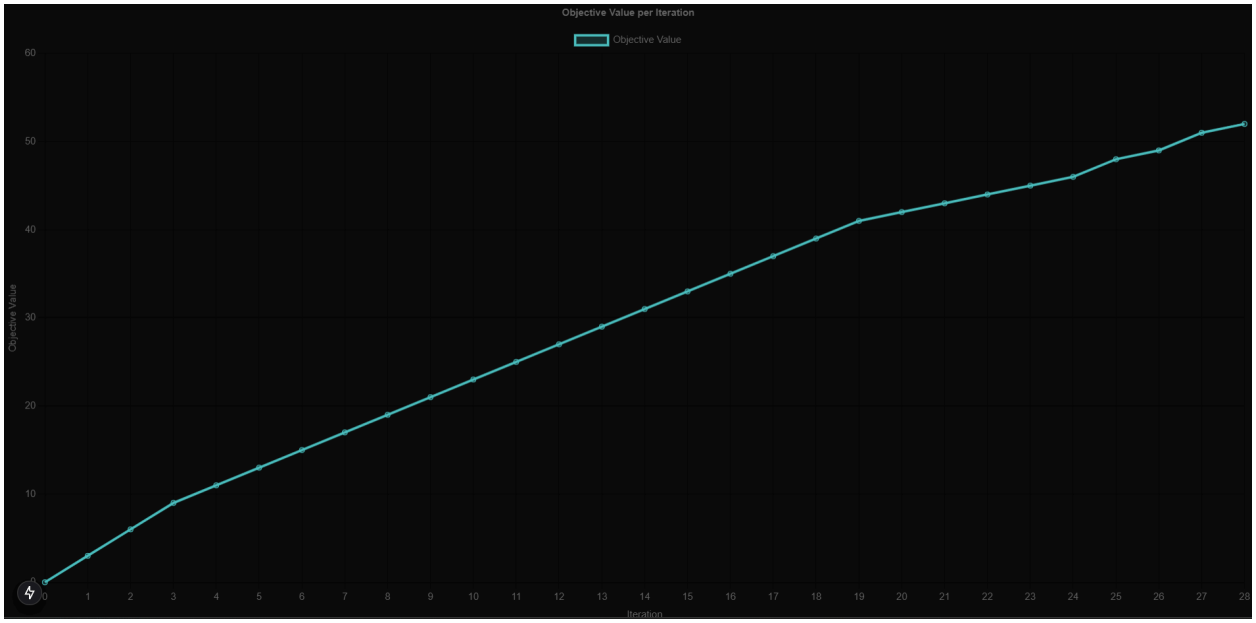
103	56	71	35	100
88	77	73	84	82
60	107	117	47	116
20	109	16	4	28
97	112	40	64	29

43	46	23	49	30
110	34	11	75	59
37	114	101	54	44
69	48	26	25	7
78	119	18	19	42

94	124	61	92	14
6	53	93	9	51
99	121	79	8	17
50	55	66	113	118
63	68	1	90	57

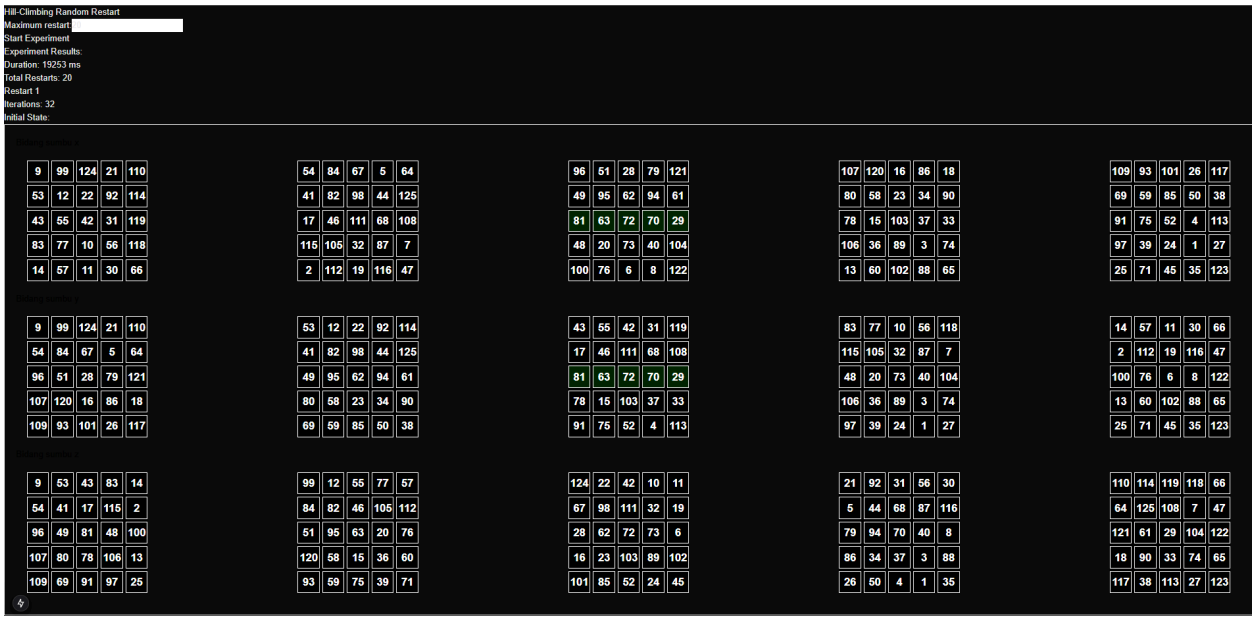
108	65	95	105	123
5	3	125	106	36
15	85	74	120	39
58	12	86	91	41
31	27	89	122	21

70	2	32	72	52
80	111	102	45	104
13	24	67	87	33
10	83	98	115	38
96	76	81	62	22

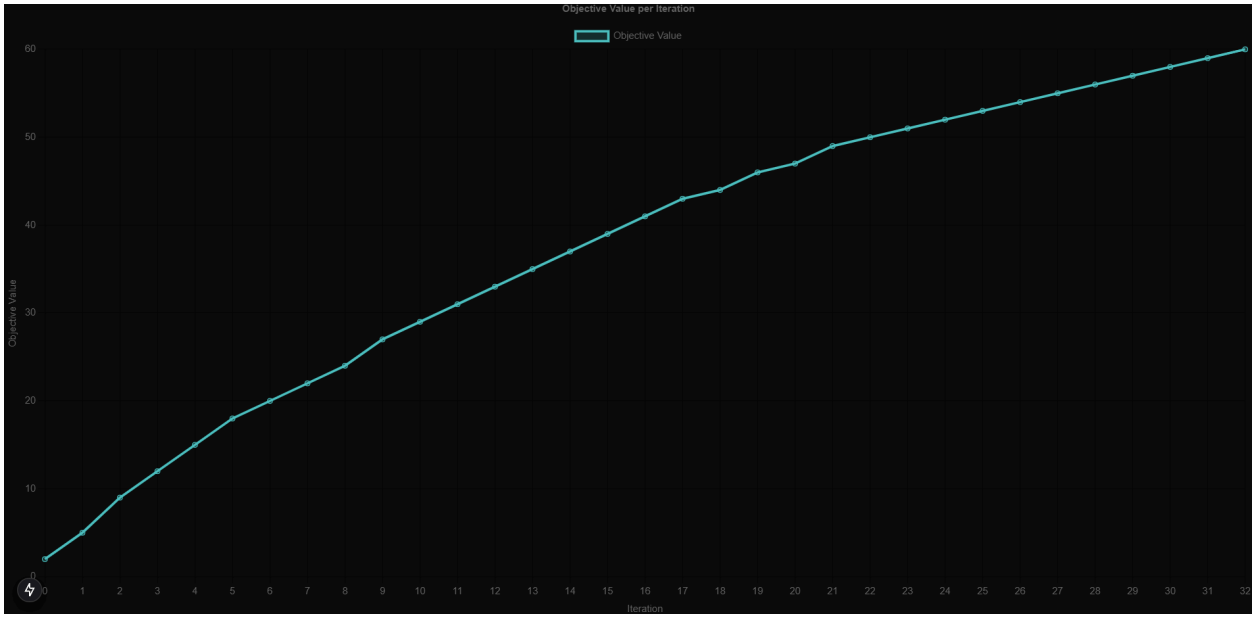


Dalam percobaan ini, dilakukan 15 kali restart, didapat hasil yang cukup baik, yaitu state akhir dengan objective value kurang lebih 50 untuk setiap restart. Hasil ini juga didapat dalam waktu yang cukup cepat, yaitu sekitar 15 detik total, atau 1 detik per restart. Terdapat sebanyak kurang lebih 20-30 iterasi per restart. Local optima berhasil diraih.

Percobaan 3:


Hill-Climbing Random Restart
Maximum restart:
Start Experiment
Experiment Results:
Duration: 19253 ms
Total Restarts: 20
Restart 1
Iterations: 32
Initial State:
9 99 124 21 110
53 12 22 92 114
43 55 42 31 119
83 77 10 56 118
14 57 11 30 66
54 84 67 5 64
41 82 98 44 125
17 46 111 68 108
115 105 32 87 7
2 112 19 116 47
96 51 28 79 121
49 95 62 94 61
81 63 72 70 29
48 20 73 40 104
100 76 6 8 122
107 120 16 86 18
80 58 23 34 90
78 15 103 37 33
106 36 89 3 74
13 60 102 88 65
109 93 101 26 117
69 59 85 50 38
91 75 52 4 113
97 39 24 1 27
25 71 45 35 123
9 99 124 21 110
54 84 67 5 64
96 51 28 79 121
107 120 16 86 18
109 93 101 26 117
53 12 22 92 114
41 82 98 44 125
49 95 62 94 61
80 58 23 34 90
69 59 85 50 38
43 55 42 31 119
17 46 111 68 108
81 63 72 70 29
78 15 103 37 33
91 75 52 4 113
83 77 10 56 118
115 105 32 87 7
48 20 73 40 104
106 36 89 3 74
97 39 24 1 27
14 57 11 30 66
2 112 19 116 47
100 76 6 8 122
13 60 102 88 65
25 71 45 35 123
9 53 43 83 14
54 41 17 115 2
96 49 81 48 100
107 80 78 106 13
109 69 91 97 25
99 12 55 77 57
84 82 46 105 112
51 95 63 20 76
120 58 15 36 60
93 59 75 39 71
124 22 42 10 11
67 98 111 32 19
28 62 72 73 6
16 23 103 89 102
101 85 52 24 45
21 92 31 56 30
5 44 68 87 116
79 94 70 40 8
86 34 37 3 88
26 50 4 1 35
110 114 119 118 66
64 125 108 7 47
121 61 29 104 122
18 90 33 74 65
117 38 113 27 123


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



Restart 20
Iterations: 28
Initial State:

Initial Solution

48	124	14	82	83
2	5	79	27	40
16	30	62	34	111
15	43	13	47	1
106	19	68	75	100

65	108	112	49	35
57	42	95	45	53
73	80	125	56	117
123	103	98	74	51
97	67	77	10	32

105	8	110	46	21
119	18	76	17	92
55	91	86	33	71
70	93	54	28	102
109	113	59	31	66

72	63	120	12	11
122	87	121	90	69
64	104	85	58	118
22	50	99	61	52
44	7	84	29	89

116	9	78	38	23
60	6	81	20	88
115	3	25	114	96
94	4	41	36	101
37	107	39	24	26

Initial Solution

48	124	14	82	83
65	108	112	49	35
105	8	110	46	21
72	63	120	12	11
116	9	78	38	23

2	5	79	27	40
57	42	95	45	53
119	18	76	17	92
122	87	121	90	69
60	6	81	20	88

16	30	62	34	111
73	80	125	56	117
55	91	86	33	71
64	104	85	58	118
115	3	25	114	96

15	43	13	47	1
123	103	98	74	51
70	93	54	28	102
22	50	99	61	52
94	4	41	36	101

106	19	68	75	100
97	67	77	10	32
109	113	59	31	66
44	7	84	29	89
37	107	39	24	26

Initial Solution

48	2	16	15	106
65	57	73	123	97
105	119	55	70	109
72	122	64	22	44
116	60	115	94	37

124	5	30	43	19
108	42	80	103	67
8	18	91	93	113
63	87	104	50	7
9	6	3	4	107

14	79	62	13	68
112	95	125	98	77
110	76	86	54	59
120	121	85	99	84
78	81	25	41	39

82	27	34	47	75
49	45	56	74	10
46	17	33	28	31
12	90	58	61	29
38	20	114	36	24

83	40	111	1	100
35	53	117	51	32
21	92	71	102	66
11	69	118	52	89
23	88	96	101	26

Final State:

Initial Solution

48	20	36	101	110
2	5	79	39	40
24	30	62	34	87
15	11	22	47	1
106	21	68	16	99

65	108	112	118	32
125	69	95	45	53
120	80	86	76	117
123	103	98	74	78
19	67	77	10	35

14	115	43	46	97
6	18	56	17	92
55	111	57	33	71
70	93	54	28	102
109	113	59	31	66

72	63	73	12	83
122	104	4	90	42
64	91	85	58	49
13	50	100	61	52
44	7	84	29	89

116	9	51	38	23
60	119	81	124	88
8	3	25	114	96
94	121	41	105	82
37	107	27	75	26

Initial Solution

48	20	36	101	110
65	108	112	118	32
14	115	43	46	97
72	63	73	12	83
116	9	51	38	23

2	5	79	39	40
125	69	95	45	53
6	18	56	17	92
122	104	4	90	42
60	119	81	124	88

24	30	62	34	87
120	80	86	76	117
55	111	57	33	71
64	91	85	58	49
8	3	25	114	96

15	11	22	47	1
123	103	98	74	78
70	93	54	28	102
13	50	100	61	52
94	121	41	105	82

106	21	68	16	99
19	67	77	10	35
109	113	69	31	66
44	7	84	29	89
37	107	27	75	26

Initial Solution

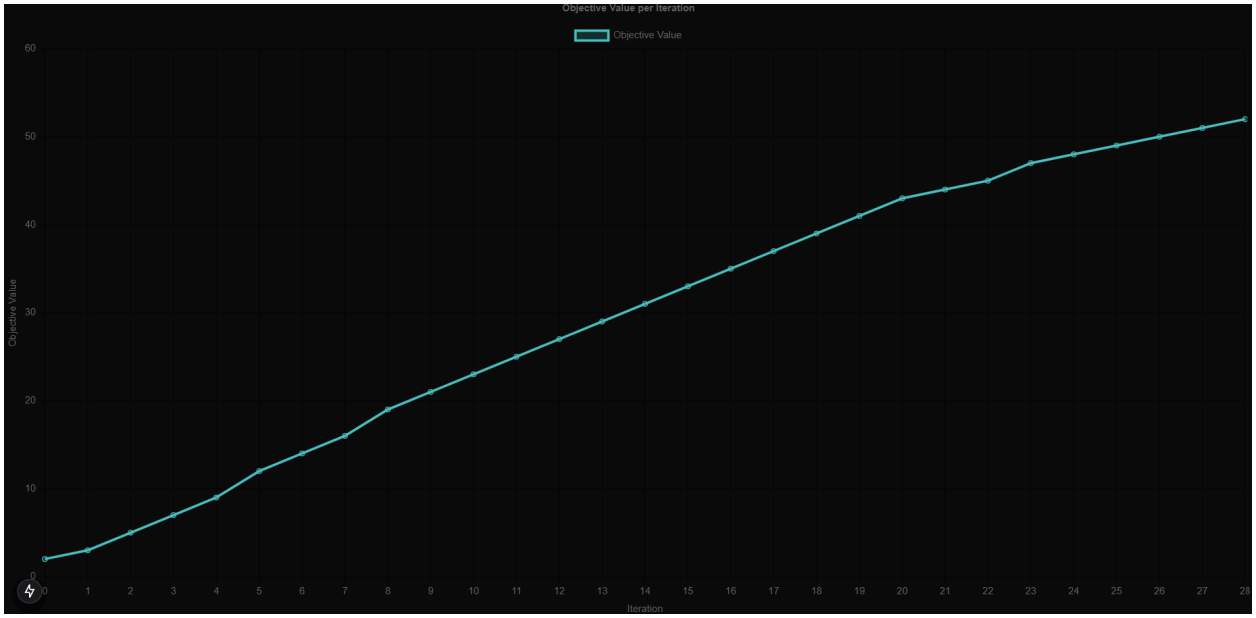
48	2	24	15	106
65	125	120	123	19
14	6	55	70	109
72	122	64	13	44
116	60	8	94	37

20	5	30	11	21
108	69	80	103	67
115	18	111	93	113
63	104	91	50	7
9	119	3	121	107

36	79	62	22	68
112	95	86	98	77
43	56	57	54	59
73	4	85	100	84
51	81	25	41	27

101	39	34	47	16
118	45	76	74	10
46	17	33	28	31
12	90	58	61	29
38	124	114	105	75

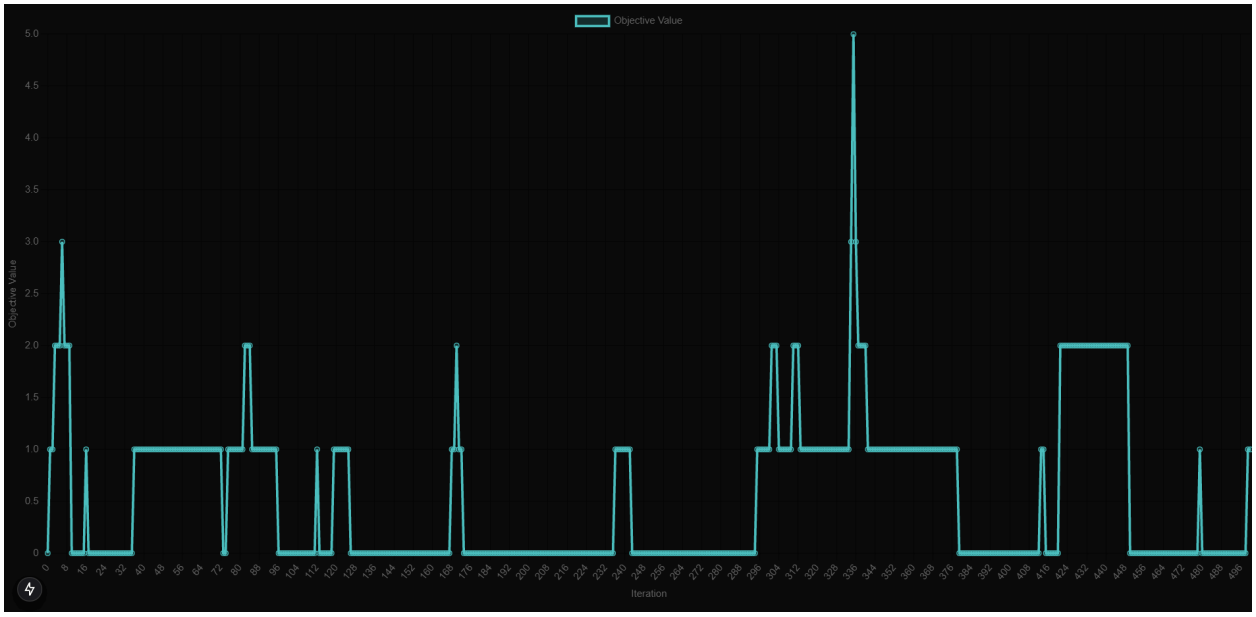
110	40	87	1	99
32	53	117	78	35
97	92	71	102	66
83	42	49	52	89
23	88	96	82	26



Dalam percobaan ini, dilakukan 20 kali restart, didapat hasil yang cukup baik, yaitu state akhir dengan objective value kurang lebih 50 untuk setiap restart. Hasil ini juga didapat dalam waktu yang cukup cepat, yaitu sekitar 20 detik total, atau 1 detik per restart. Terdapat sebanyak kurang lebih 15-30 iterasi per restart. Local optima berhasil diraih.

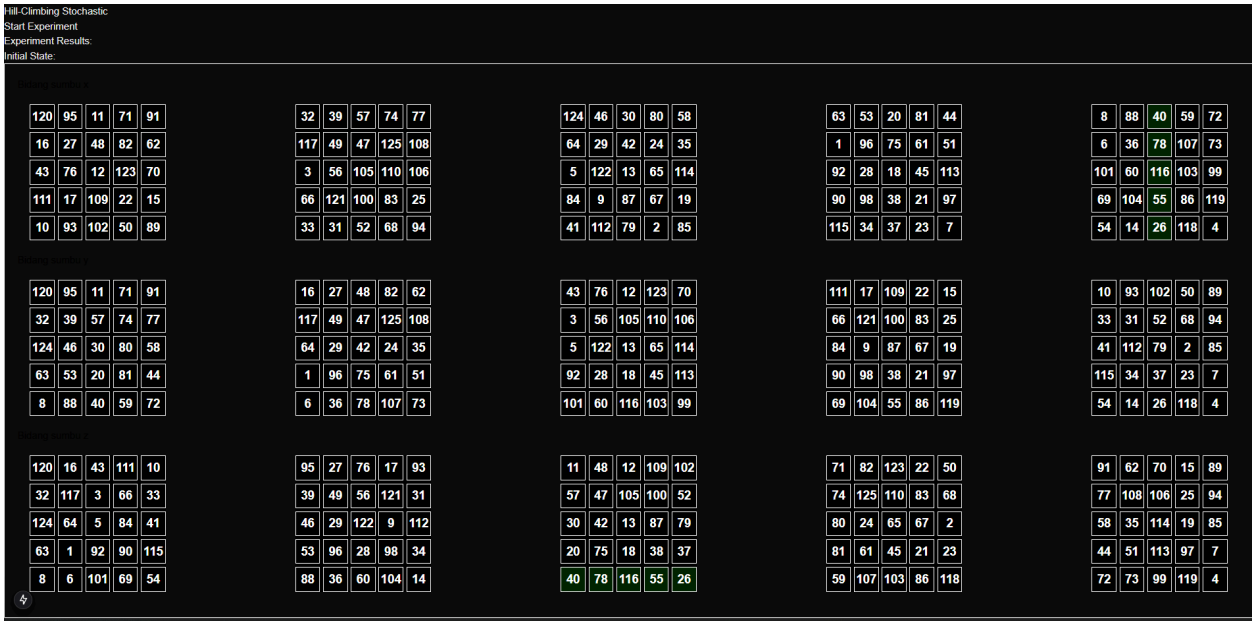
Stochastic Hill Climbing

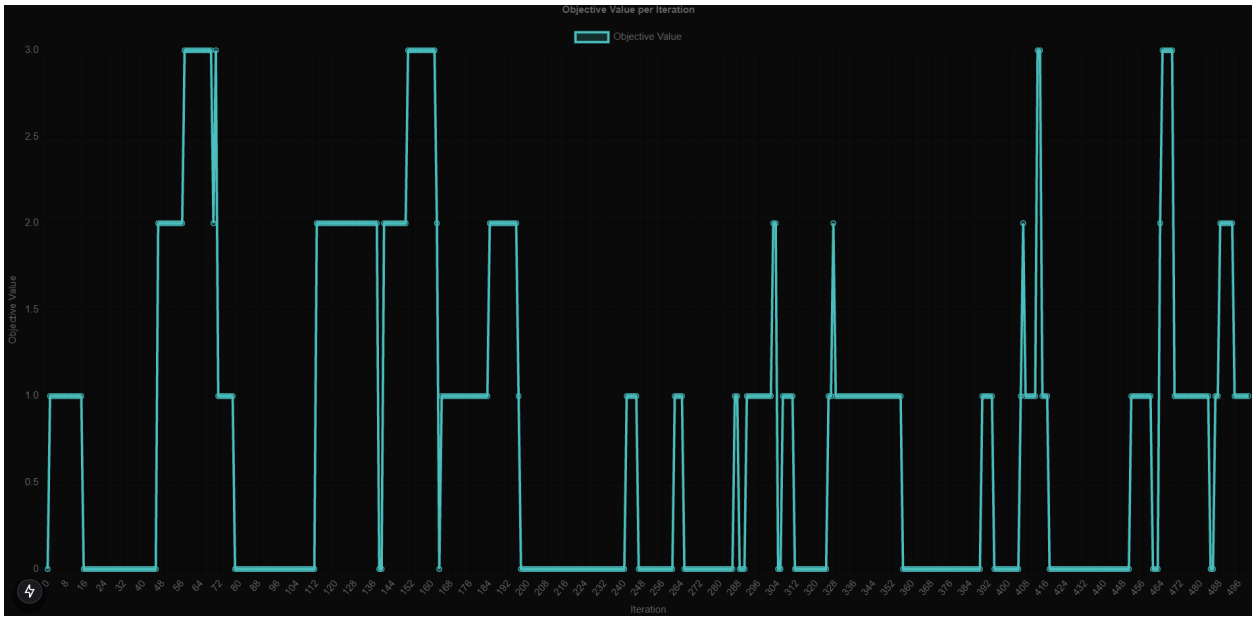
Percobaan 1:



Dalam percobaan ini, didapat hasil yang cukup baik, yaitu state paling optimal dengan objective value sebesar 50. Hasil ini juga didapat dalam waktu yang cukup cepat, yaitu sekitar 0,02 detik. Terdapat sebanyak 500 iterasi. Local optima berhasil dicapai di iterasi ke-340.

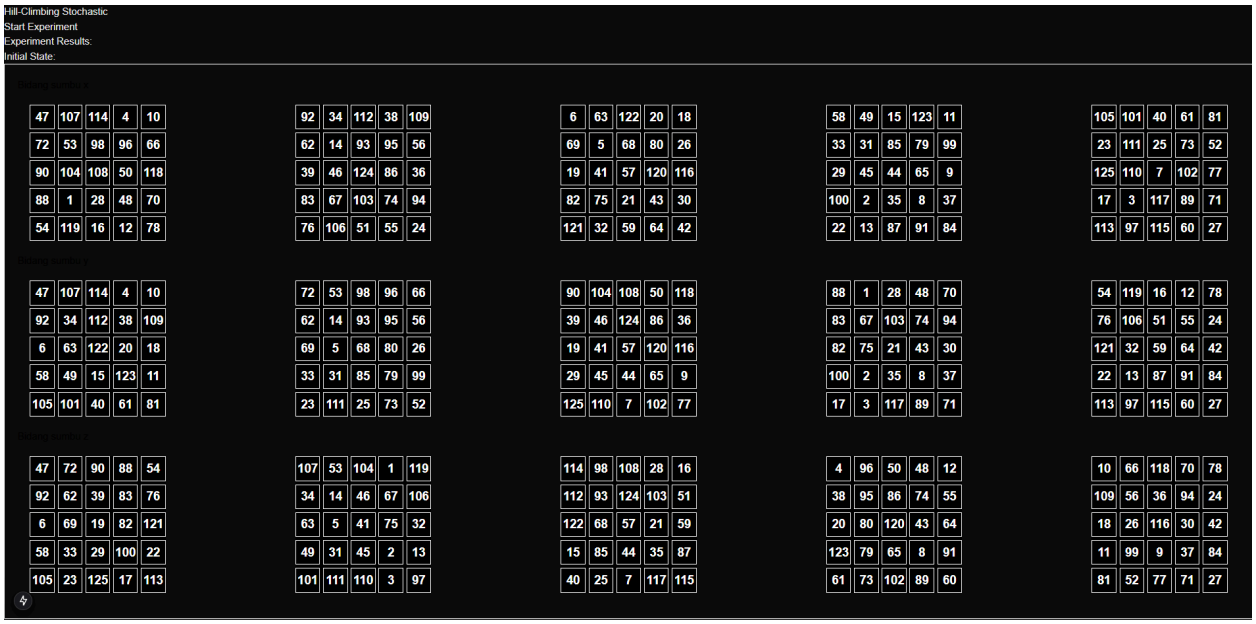
Percobaan 2:



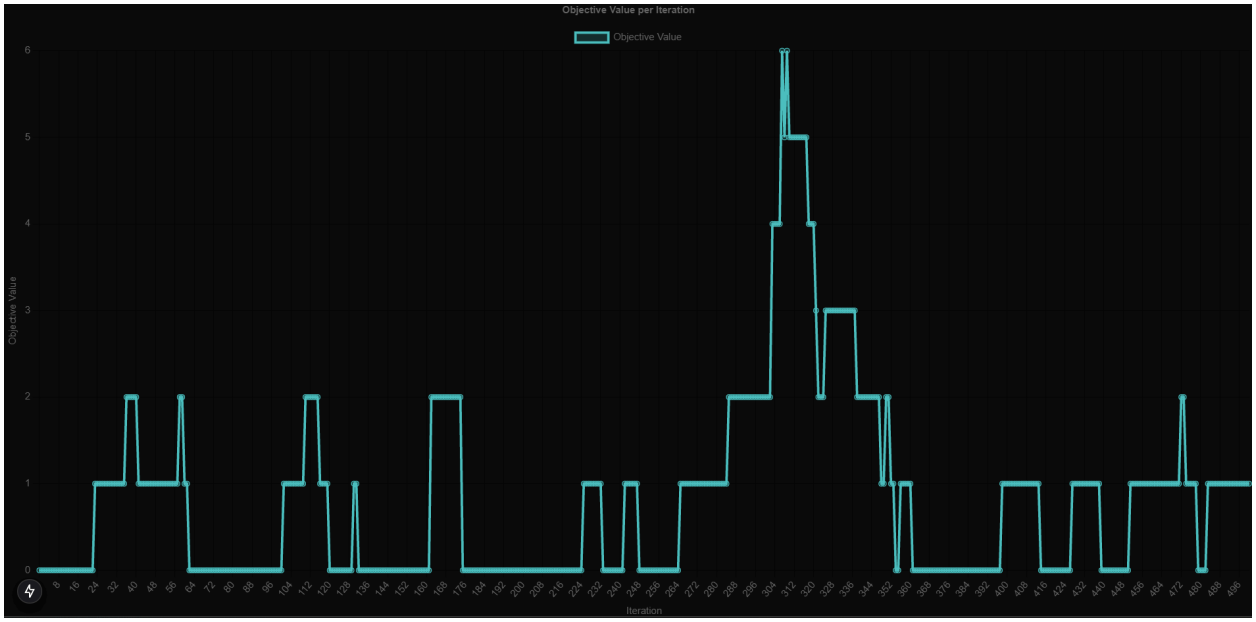


Dalam percobaan ini, didapat hasil yang sangat buruk, yaitu state-state paling optimal dengan objective value sebesar 3. Hasil ini juga didapat dalam waktu yang cukup cepat, yaitu sekitar 0,02 detik. Terdapat sebanyak 500 iterasi. Local optima berhasil dicapai di beberapa iterasi.

Percobaan 3:


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

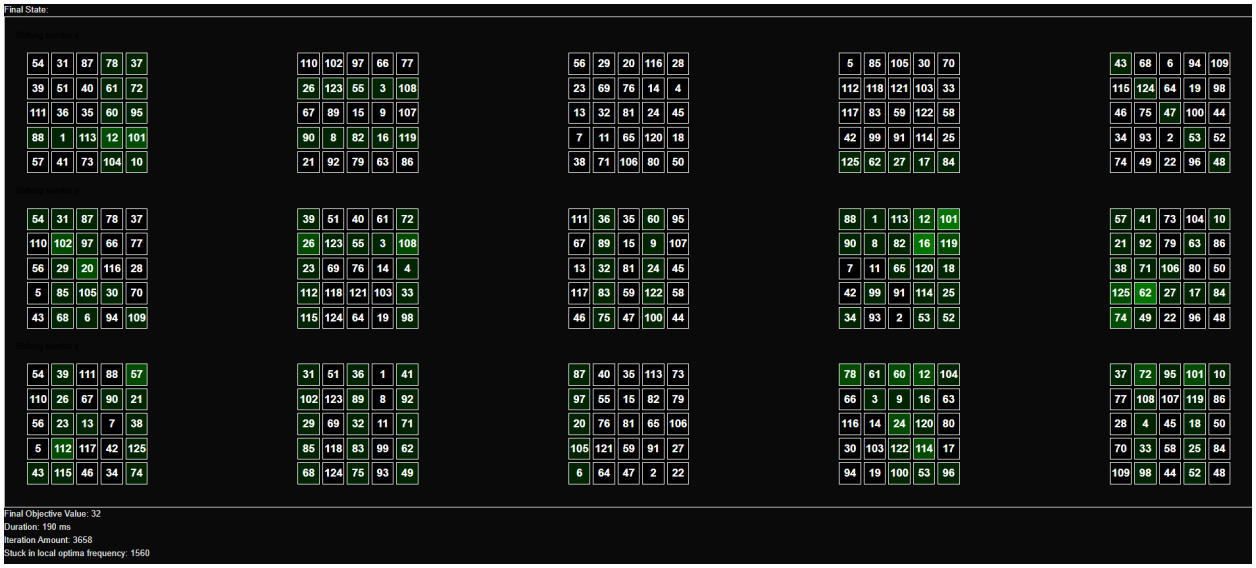

Final State:
Solving sonbs x
35 58 76 40 51
80 95 107 9 36
33 99 82 31 23
102 120 74 119 50
111 118 65 122 1
6 13 59 116 70
29 78 12 124 42
62 46 71 22 10
67 60 38 77 91
103 53 17 45 20
87 93 3 68 25
39 15 97 88 34
4 123 14 37 109
56 115 121 26 57
73 41 18 66 101
112 89 85 16 43
27 84 104 113 79
64 32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

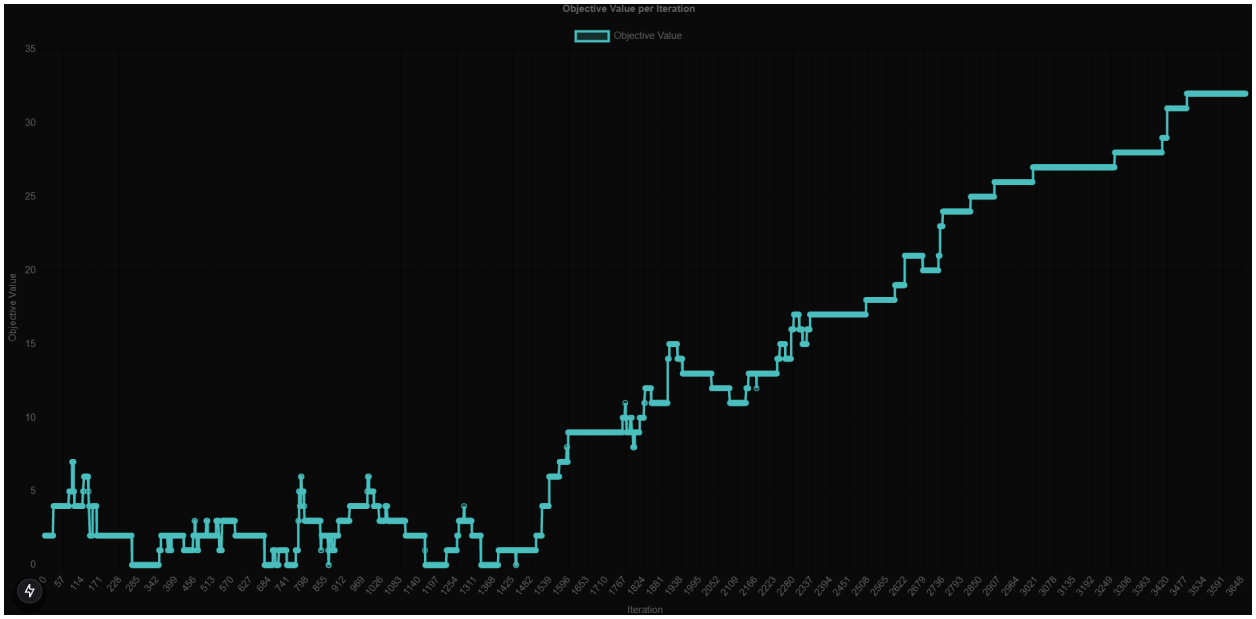


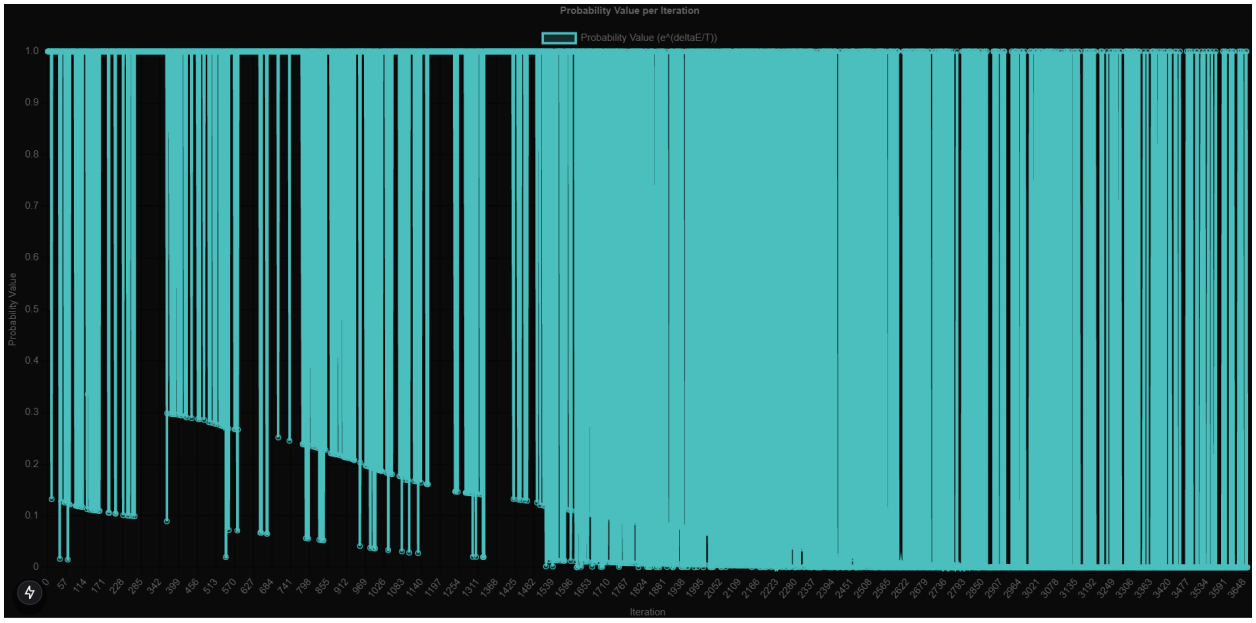
Dalam percobaan ini, didapat hasil yang sangat buruk, yaitu state paling optimal dengan objective value sebesar 6. Hasil ini juga didapat dalam waktu yang cukup cepat, yaitu sekitar 0,02 detik. Terdapat sebanyak 500 iterasi. Local optima berhasil dicapai di iterasi ke-312.

Simulated Annealing

Percobaan 1:

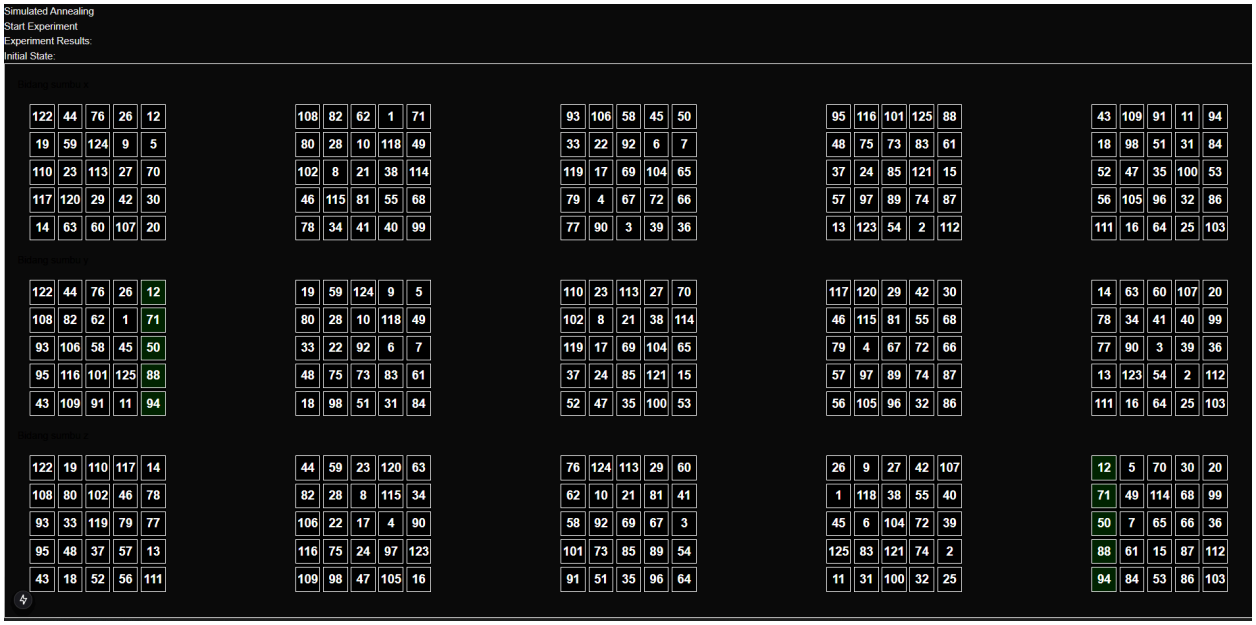

Final State:
Final Objective Value: 32
Duration: 190 ms
Iteration Amount: 3658
Stuck in local optima frequency: 1560

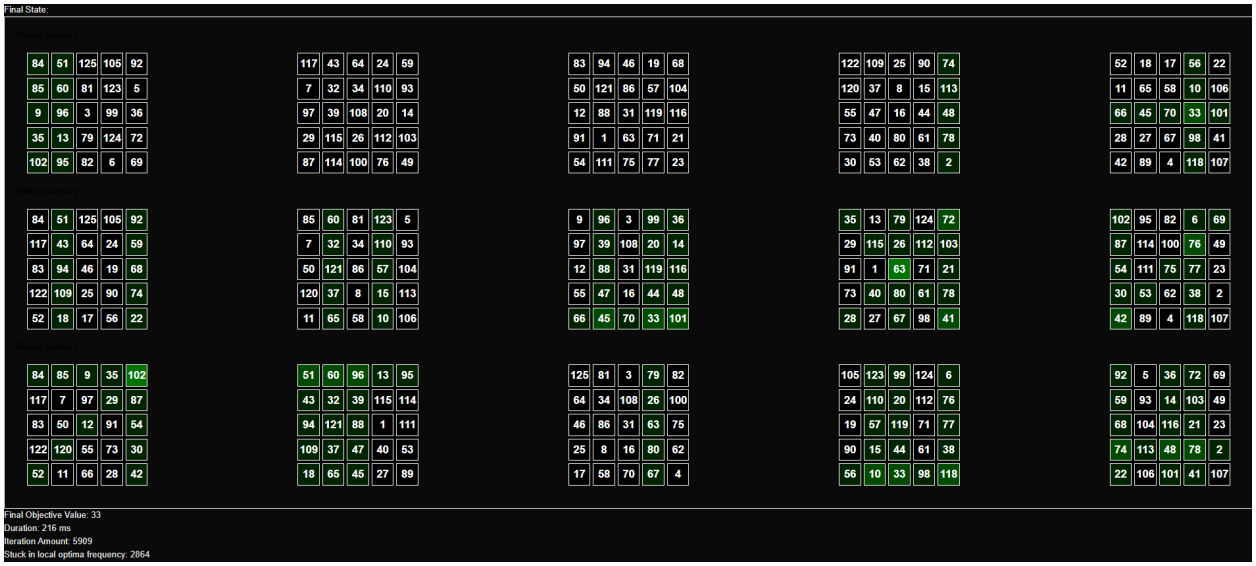

Objective Value per Iteration
Objective Value
Objective Value
Iteration

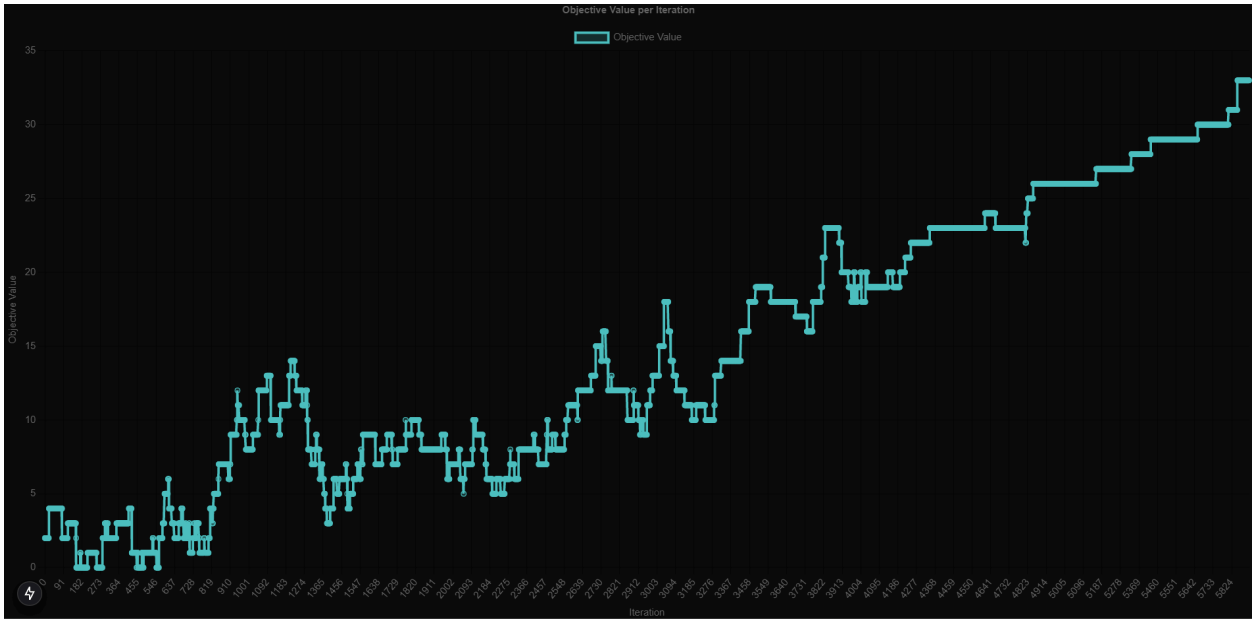


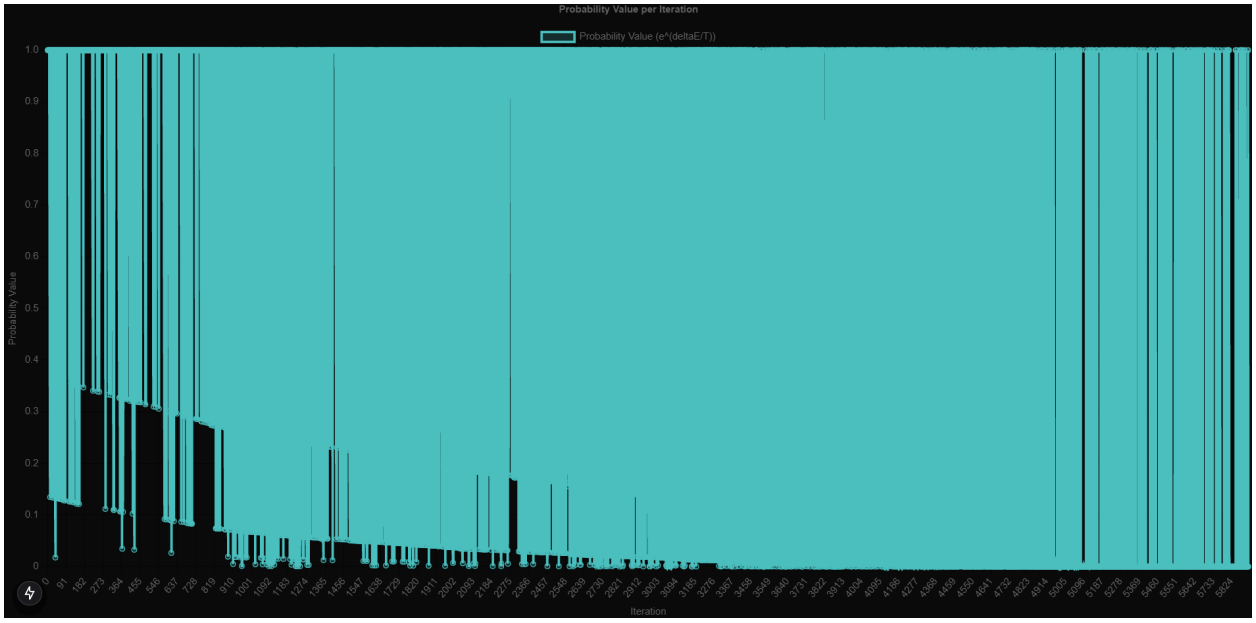
Dalam percobaan ini, didapat hasil yang cukup baik, yaitu state akhir dengan objective value sebesar 32. Hasil ini juga didapat dalam waktu yang sangat cepat, yaitu sekitar 0,2 detik. Terdapat sebanyak 3658 iterasi. Local optima berhasil diraih. State terjebak di local optima sebanyak total 1560 iterasi.

Percobaan 2:



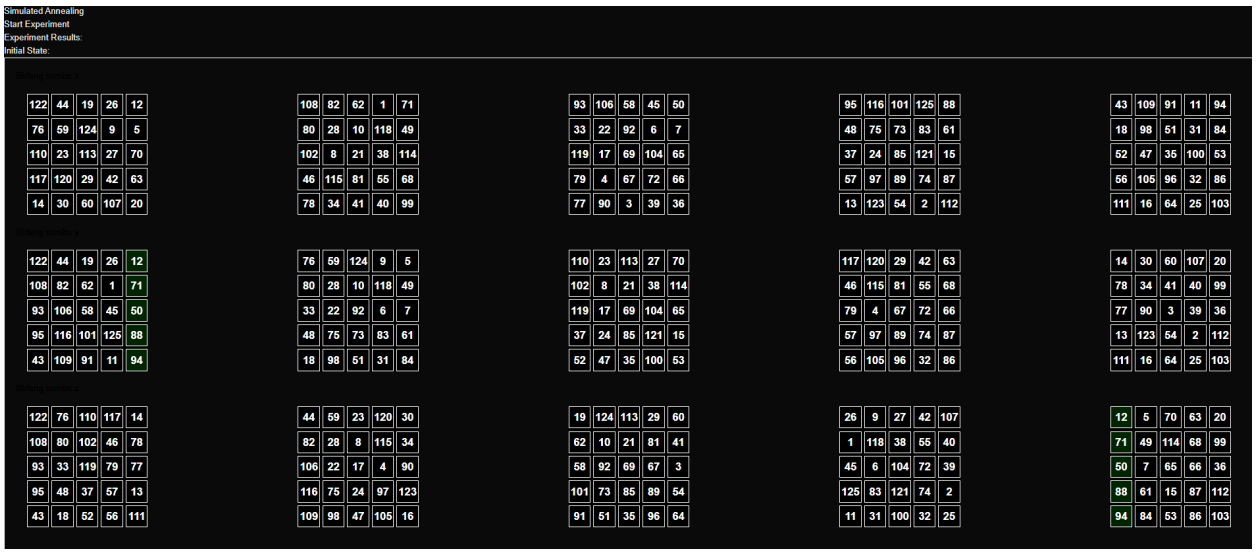

Final State:
Final Objective Value: 33
Duration: 216 ms
Iteration Amount: 5909
Stuck in local optima frequency: 2864

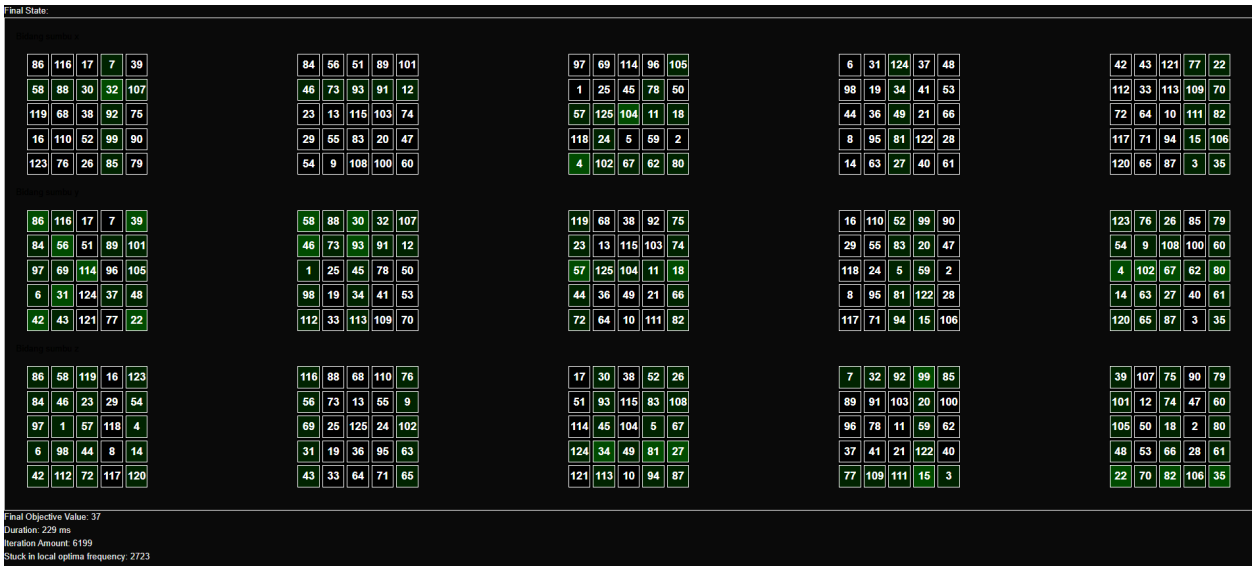

Objective Value per Iteration
Objective Value
Objective Value
Iteration

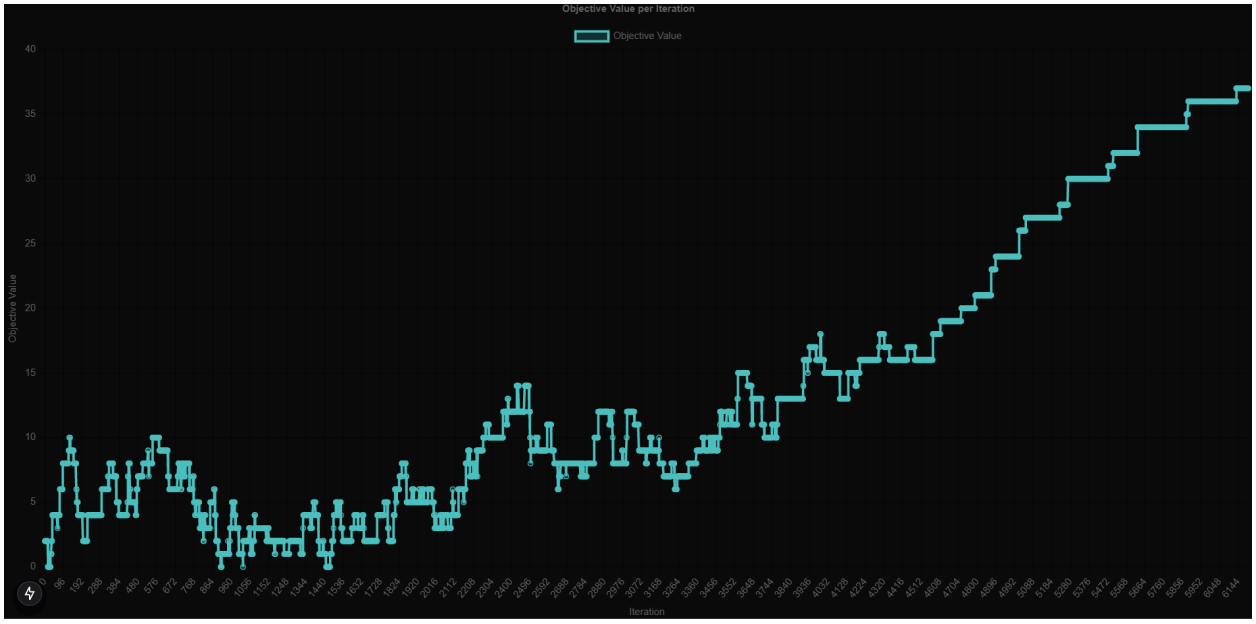


Dalam percobaan ini, didapat hasil yang cukup baik, yaitu state akhir dengan objective value sebesar 33. Hasil ini juga didapat dalam waktu yang sangat cepat, yaitu sekitar 0,2 detik. Terdapat sebanyak 5909 iterasi. Local optima berhasil diraih. State terjebak di local optima sebanyak total 2864 iterasi.

Percobaan 3:



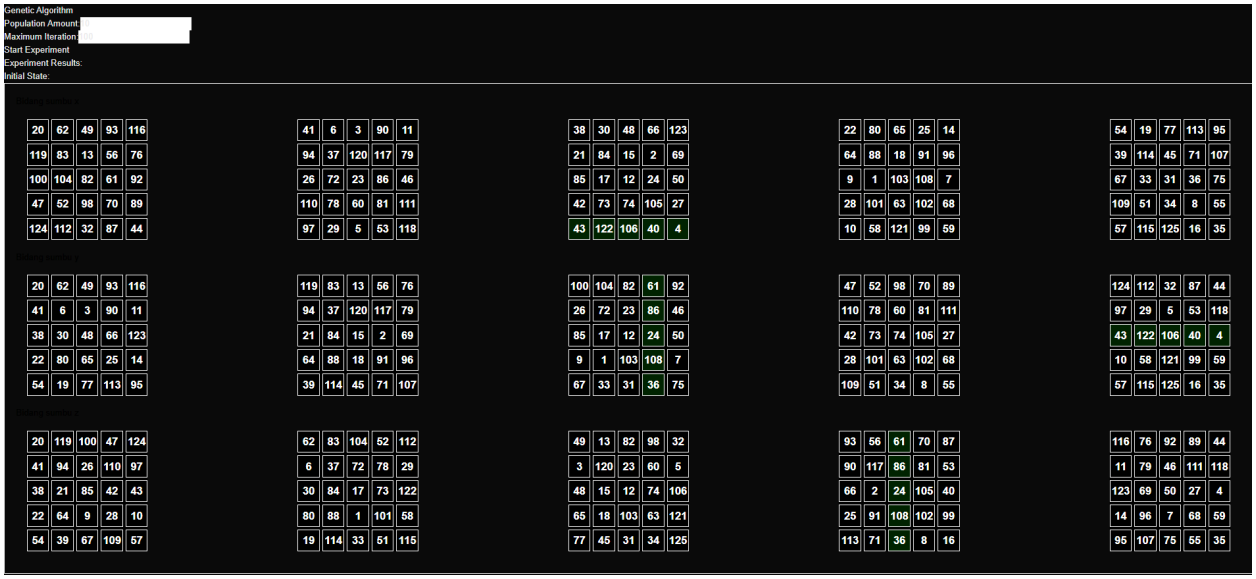


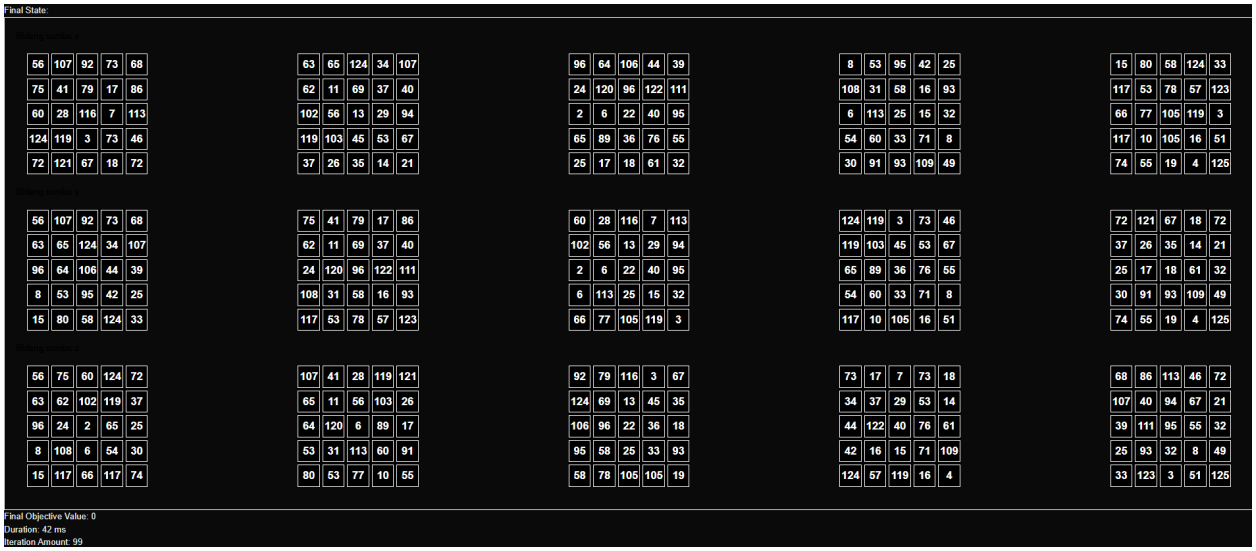


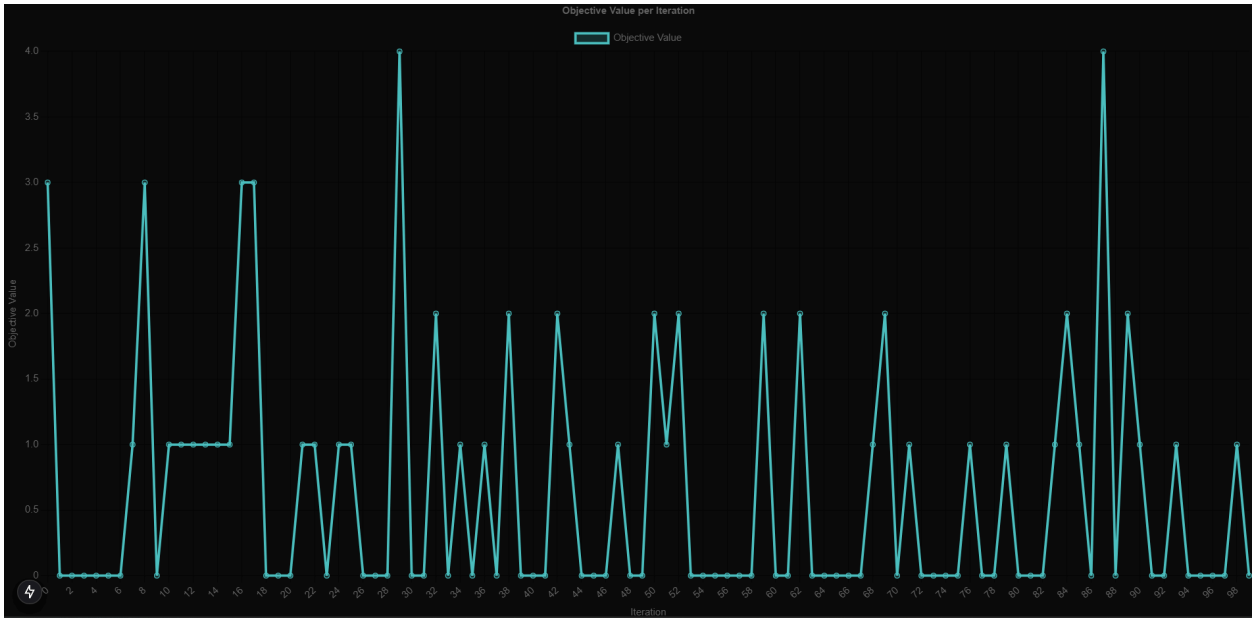
Dalam percobaan ini, didapat hasil yang cukup baik, yaitu state akhir dengan objective value sebesar 37. Hasil ini juga didapat dalam waktu yang sangat cepat, yaitu sekitar 0,2 detik. Terdapat sebanyak 6199 iterasi. Local optima berhasil diraih. State terjebak di local optima sebanyak total 2723 iterasi.

Genetic Algorithm

Percobaan 1:

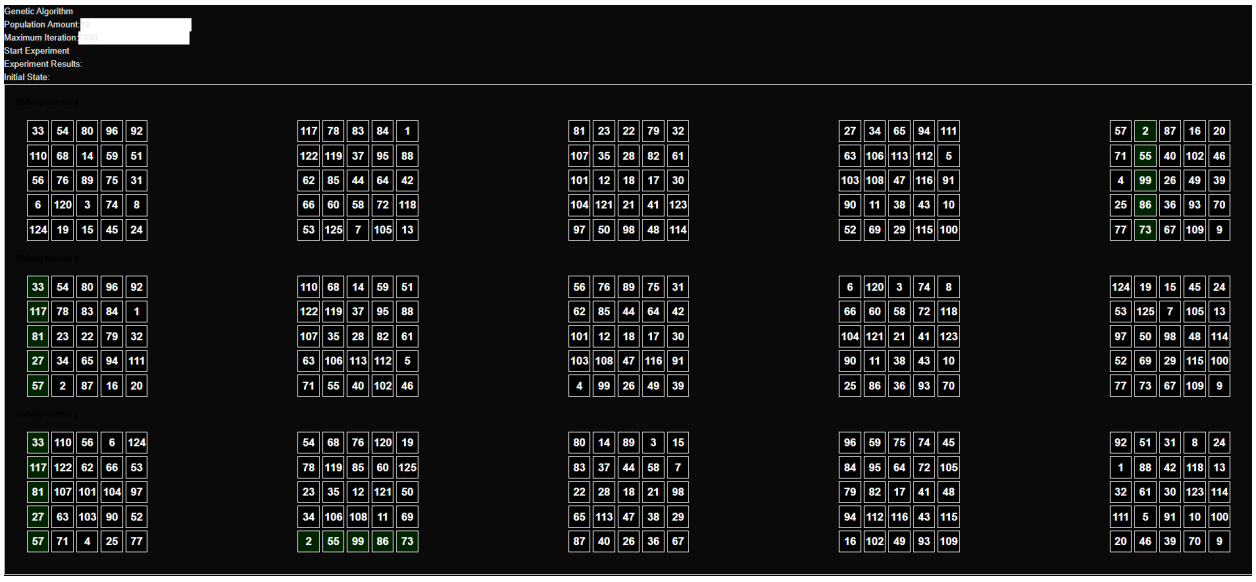


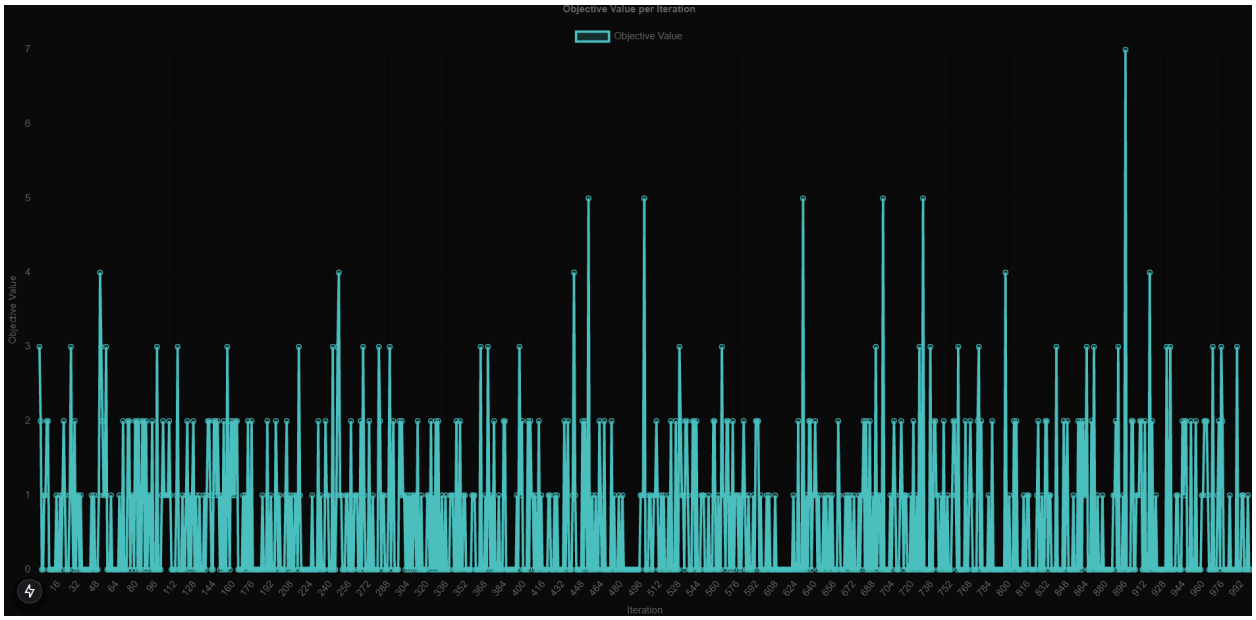




Dari percobaan menggunakan genetic algorithm di atas, didapatkan hasil yang sangat buruk. Genetic Algorithm terjebak di dalam state yang memiliki state value sebesar sekitar 0 dan 1. Hal ini dapat terjadi karena Genetic Algorithm hanya bergantung pada persilangan genome dan mutasi yang dilakukan secara acak. Akibatnya, tidak ada state dengan hasil terbaik yang disimpan.

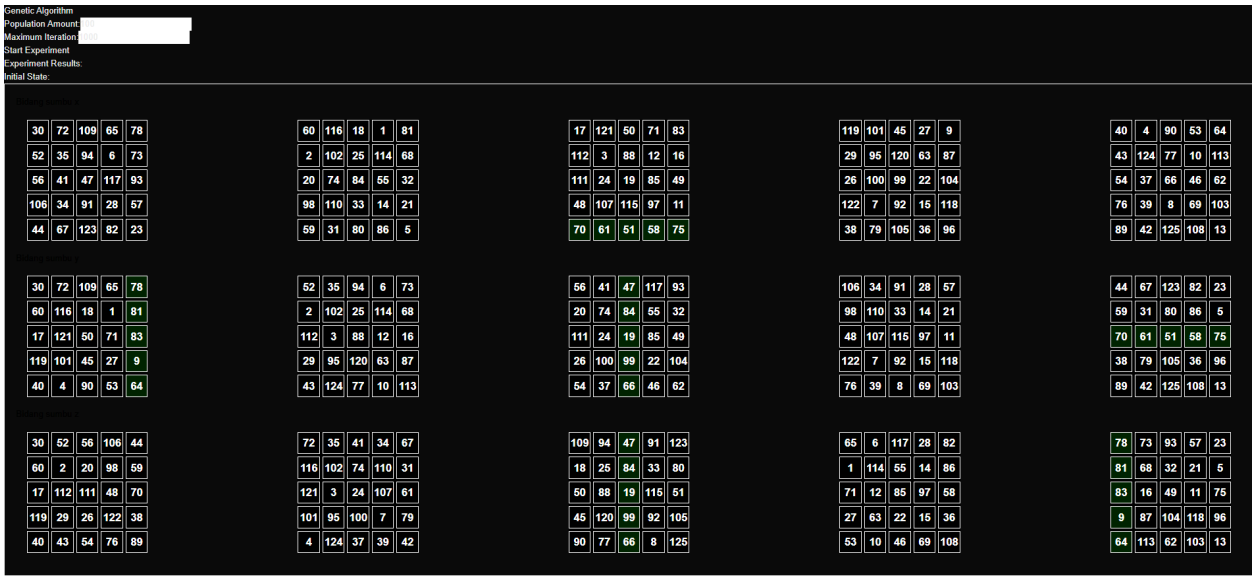
Percobaan 2:



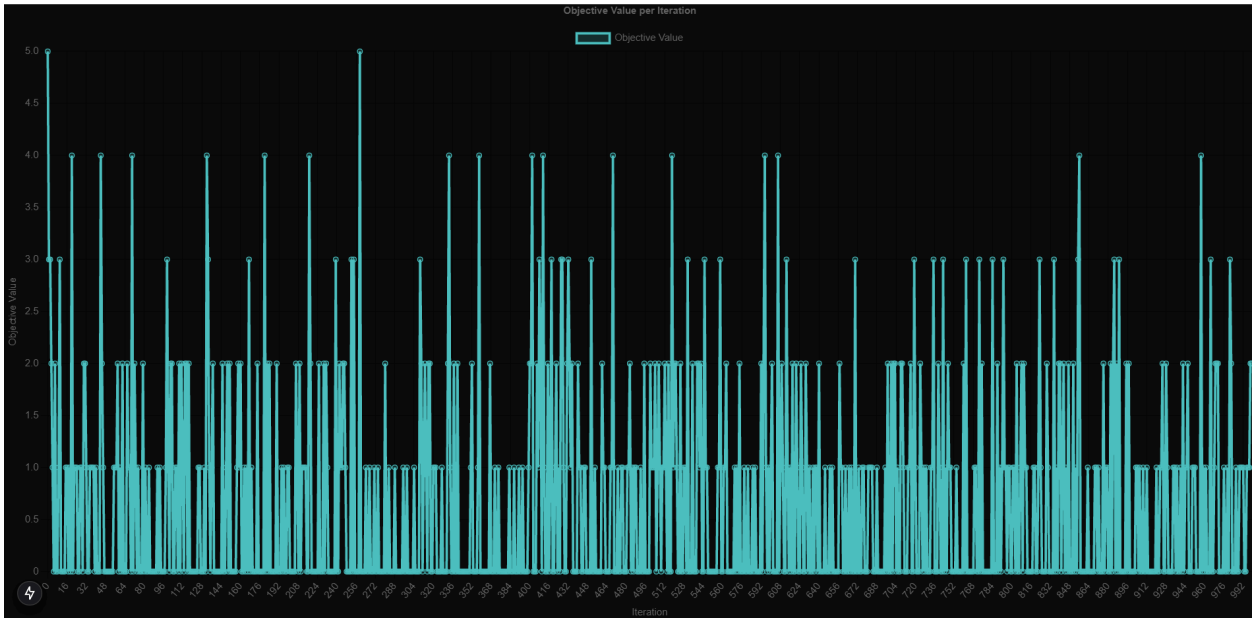


Percobaan ke-2 dengan jumlah populasi dan banyak iterasi yang diperbesar juga tidak berdampak signifikan terhadap kedekatan dengan state solusi. Hasil yang didapatkan tetap mayoritas merupakan state yang memiliki objective value sekitar 0 dan 1, dan hanya sedikit sekali yang mencapai nilai objective function yang bagus.

Percobaan 3:


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


Final State:
Cooling number 4
43 89 88 124 64
25 19 35 90 73
89 97 82 85 110
47 72 121 72 11
9 6 14 118 54
16 106 94 35 29
113 82 34 42 86
28 24 39 65 51
71 67 5 69 3
91 38 61 65 123
4 107 49 113 59
62 110 58 70 65
10 88 14 18 4
41 47 16 46 90
52 102 120 76 24
104 54 8 52 11
118 5 59 26 125
58 91 105 112 56
97 86 89 98 42
72 33 66 53 95
45 32 30 49 125
19 17 24 87 48
82 76 89 5 86
41 123 106 3 46
17 14 27 60 24
Cooling number 5
43 89 88 124 64
16 106 94 35 29
4 107 49 113 59
104 54 8 52 11
45 32 30 49 125
25 19 35 90 73
113 82 34 42 86
62 110 58 70 65
118 5 59 26 125
19 17 24 87 48
89 97 82 85 110
28 24 39 65 51
10 88 14 18 4
58 91 105 112 56
82 76 89 5 86
47 72 121 72 11
71 67 5 69 3
41 47 16 46 90
97 86 89 98 42
41 123 106 3 46
9 6 14 118 54
91 38 61 65 123
52 102 120 76 24
72 33 66 53 95
17 14 27 60 24
Cooling number 6
43 25 89 47 9
16 113 28 71 91
4 62 10 41 52
104 118 58 97 72
45 19 82 41 17
89 19 97 72 6
106 82 24 67 38
107 110 88 47 102
54 5 91 86 33
32 17 76 123 14
88 35 82 121 14
94 34 39 5 61
49 58 14 16 120
8 59 105 89 66
30 24 89 106 27
124 90 85 72 118
35 42 65 69 65
113 70 18 46 76
52 26 112 98 53
49 87 5 3 60
64 73 110 11 54
29 86 51 3 123
59 65 4 90 24
11 125 56 42 95
125 48 86 46 24
Final Objective Value: 2
Duration: 1605 ms
Iteration Amount: 999



Percobaan ke-3 juga tidak menghasilkan hasil yang lebih baik daripada percobaan 1 dan 2. Hasilnya juga sama buruknya dengan percobaan menggunakan genetic algorithm sebelumnya meskipun banyak iterasi dan banyak populasi diubah.

Kesimpulan dan Saran

Dari percobaan semua algoritma local search di atas, genetic algorithm memiliki performa yang paling buruk karena hasil state akhirnya kemungkinan besar terjebak di state yang memiliki objective value yang memiliki nilai 0 dan 1, dan sepanjang iterasi, genetic algorithm kesulitan untuk mendapatkan state yang memiliki objective value yang lebih baik daripada sebelumnya. Hal ini bisa terjadi karena genetic algorithm benar-benar melakukan crossover dan mutation secara acak tanpa mempedulikan genome terbaik. Algoritma terburuk berikutnya diraih oleh simulated annealing, karena state akhirnya yang hanya berada di kisaran <10 pada setiap percobaan. Hal ini bisa terjadi karena temperatur dan cooling rate yang belum disesuaikan semaksimal mungkin untuk permasalahan magic cube.

Jika kita membandingkan algoritma lainnya dari sisi objective value tertinggi yang berhasil diraih, algoritma steepest ascent, sideways move, dan random-restart menduduki posisi teratas dengan mencapai objective value tertinggi sebesar >=50 di semua percobaan. Di posisi berikutnya, terdapat algoritma simulated annealing dengan objective value tertinggi di kisaran 30an pada semua percobaan.

Pembagian Tugas

NIM	Kontribusi
13522036	Backend (Genetic Algorithm, Magic Cube Class), Frontend (Visualisasi Magic Cube)
13522064	Backend (Simulated Annealing), Laporan
13522099	Backend (Steepest Ascent HC, Sideways Move HC), Frontend (Visualisasi hasil eksperimen, visualisasi plot)
13522106	Backend (Random Restart HC, Stochastic HC), Laporan

Referensi

https://www.magischvierkant.com/three-dimensional-eng/magic-features/

https://www.trump.de/magic-squares/magic-cubes/cubes-1.html

https://medium.com/the-stem/ai-simulated-annealing-algorithm-to-solve-magic-square-problem-4102ff9ae3fe