Tugas Besar 1

IF3170 Inteligensi Artifisial

Pencarian Solusi Diagonal Magic Cube dengan Local Search

Disusun oleh:

13522073 Juan Alfred Widjaya

13522077 Enrique Yanuar

13522081 Albert

13522115 Derwin Rustanly

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

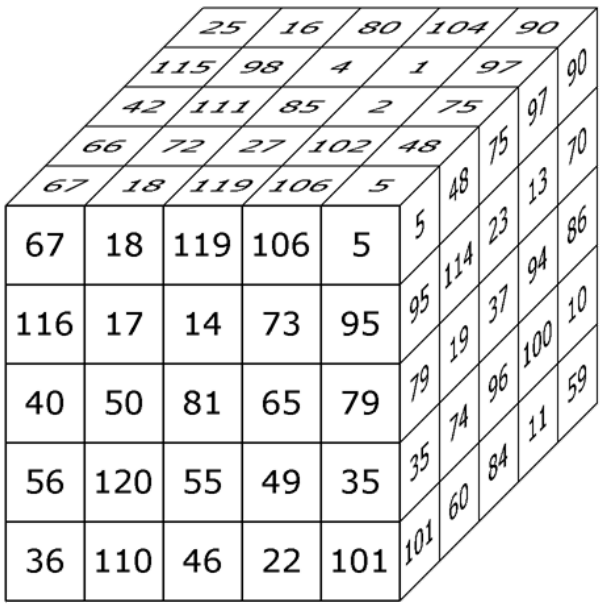
INSTITUT TEKNOLOGI BANDUNG

2024

Bab I
Deskripsi Persoalan

Diagonal *magic cube* adalah sebuah kubus yang disusun dari angka 1 hingga n^3 tanpa pengulangan, dengan n merupakan panjang sisi kubus tersebut. Angka-angka tersebut disusun sedemikian rupa sehingga memenuhi beberapa properti sebagai berikut:

- Terdapat satu angka yang disebut sebagai *magic number* dari kubus tersebut .
- Jumlah angka pada setiap baris adalah sama dengan *magic number*.
- Jumlah angka pada setiap kolom adalah sama dengan *magic number*.
- Jumlah angka pada setiap tiang adalah sama dengan *magic number*.
- Jumlah angka pada semua diagonal ruang kubus adalah sama dengan *magic number*.
- Jumlah angka pada semua diagonal pada suatu bidang potongan kubus juga adalah sama dengan *magic number*.



Gambar I.1 Ilustrasi Magic Cube berukuran 5x5

Dalam tugas ini, permasalahan *Diagonal Magic Cube* berukuran 5x5x5 akan diselesaikan dengan keadaan awal berupa susunan acak angka dari 1 hingga 125, memanfaatkan berbagai algoritma pencarian *local search*, termasuk Steepest Ascent Hill-climbing, Hill-climbing with Sideways Move, Random Restart Hill-climbing, Stochastic Hill-climbing, Simulated Annealing, dan Genetic Algorithm. Setiap iterasi algoritma local search memungkinkan dua angka di dalam kubus untuk ditukar posisinya tanpa syarat harus berdekatan, sehingga memberi fleksibilitas dalam pencarian solusi. Mahasiswa diharapkan mengimplementasikan dan menguji efektivitas dari masing-masing algoritma pencarian lokal, serta menganalisis dan membandingkan performa algoritma-algoritma tersebut dalam upaya menemukan solusi optimal untuk permasalahan *Diagonal Magic Cube*.

Bab II

Pembahasan

a. Pemilihan objective function

Sebuah *magic cube* dengan ukuran $n \times n \times n$ yang setiap selnya tersusun dari angka 1 hingga n^3 memiliki sejumlah $n \times n$ buah baris, kolom, dan tiang. Angka-angka tersebut disusun sedemikian rupa sehingga setiap baris, kolom, tiang, dan diagonal di dalam kubus memiliki jumlah yang sama, yang disebut sebagai *magic number*. Adapun, jumlah angka pada keseluruhan sel dari *magic cube* dapat direpresentasikan oleh persamaan matematis

$$S_{cell} = \sum_{i=1}^{n^3} i = \frac{n^3(n^3+1)}{2}$$

Magic cube memiliki beberapa sifat:

1. Setiap angka dalam rentang 1 hingga n^3 harus ditempatkan tanpa pengulangan di dalam kubus.
2. Jumlah angka-angka di setiap baris, kolom, dan tiang (kedalaman) harus sama dengan *magic number*.
3. Jumlah angka pada semua diagonal ruang (diagonal yang terbentang dari satu sudut kubus ke sudut berlawanan dalam tiga dimensi) harus sama dengan *magic number*.
4. Jumlah angka pada diagonal dalam potongan bidang (2D diagonal pada setiap lapisan dari kubus) juga harus sama dengan *magic number*.

Dengan memanfaatkan sifat-sifat dari *magic cube*, jumlah angka di seluruh sel magic cube harus terdistribusi secara merata di setiap baris, kolom, dan tiang pada kubus. Oleh karena itu, nilai *magic number M* dapat dinyatakan dalam bentuk persamaan matematis

$$M = \frac{S_{cell}}{\Sigma baris,kolom,\, atau\, tiang} = \frac{n^3(n^3+1)/2}{n^2} = \frac{n(n^3+1)}{2}$$

Instansiasi persoalan berupa *magic cube* berukuran $5 \times 5 \times 5$ memiliki nilai magic number berupa

$$M = \frac{n(n^3+1)}{2} = \frac{5(5^3+1)}{2} = 315$$

Untuk mengukur seberapa baik suatu *state* yang merepresentasikan susunan angka di dalam kubus berukuran $5 \times 5 \times 5$, diperlukan adanya fungsi yang membandingkan seberapa dekat jumlah dari setiap baris, kolom, tiang, serta diagonal bidang dan ruang pada kubus terhadap nilai *magic number M* yang telah dirumuskan sebelumnya. Oleh karena itu, digunakan *objective function* $f(x)$ berupa besar deviasi (simpangan) masing-masing jumlah terhadap nilai *magic number M*, yang dituliskan dalam persamaan matematis

$$f(x) = \sum_{i=1}^{5}\sum_{j=1}^{5}\left|S_{row} - M\right| + \sum_{j=1}^{5}\sum_{k=1}^{5}\left|S_{col} - M\right| + \sum_{k=1}^{5}\sum_{i=1}^{5}\left|S_{tow} - M\right| + \sum_{l=1}^{D}\left|S_{diag,l} - M\right|$$

dengan

- $S_{row} = \sum_{k=1}^{5} x_{ijk}$ merupakan jumlah angka pada baris di potongan bidang *ij*.
- $S_{col} = \sum_{i=1}^{5} x_{ijk}$ merupakan jumlah angka pada kolom di potongan bidang *jk*.
- $S_{tow} = \sum_{j=1}^{5} x_{ijk}$ merupakan jumlah angka pada tiang di potongan bidang *ki*.
- $S_{diag,l}$ merupakan jumlah elemen pada diagonal ke - *l* dalam kubus. Terdapat sejumlah *D* diagonal yang mencakup diagonal-diagonal di ruang dan potongan bidang kubus.

Secara matematis, persamaan $S_{diag,l}$ dapat dituliskan sebagai

$$\sum_{l=1}^{D}\left|S_{diag,l} - M\right| = \sum_{i=1}^{5}\left|\sum_{j=1}^{5} x_{ijj} - M\right| + \sum_{j=1}^{5}\left|\sum_{k=1}^{5} x_{kjk} - M\right| + \sum_{k=1}^{5}\left|\sum_{i=1}^{5} x_{iik} - M\right| \text{ (diagonal primer bidang)}$$

$$+ \sum_{i=1}^{5}\left|\sum_{j=1}^{5} x_{ij6-j} - M\right| + \sum_{j=1}^{5}\left|\sum_{k=1}^{5} x_{kj6-k} - M\right| + \sum_{k=1}^{5}\left|\sum_{i=1}^{5} x_{i6-ik} - M\right| \text{ (diagonal sekunder bidang)}$$

$$+\left|\sum_{i=1}^{5} x_{iii} - M\right| + \left|\sum_{i=1}^{5} x_{ii6-i} - M\right| + \left|\sum_{i=1}^{5} x_{i6-ii} - M\right| + \left|\sum_{i=1}^{5} x_{6-iii} - M\right| \quad \text{(diagonal ruang)}$$

Dalam implementasinya, tujuan dari setiap algoritma pencarian adalah untuk meminimalkan nilai fungsi objektif $f(x)$, dengan target akhir $f(x) = 0$, yang menandakan bahwa semua baris, kolom, tiang, dan diagonal memiliki jumlah yang sama dengan *magic number M*.

b. Penjelasan Implementasi local search

Fungsi calculateScore

Fungsi calculateScore menghitung total deviasi jumlah elemen dalam kubus 3D terhadap magicNumber, yang merupakan nilai target jumlah elemen per baris, kolom, dan diagonal. Pertama, fungsi menghitung deviasi pada setiap baris di lapisan z, kolom di lapisan z, dan tiang vertikal sepanjang sumbu z. Selanjutnya, fungsi mengevaluasi deviasi pada diagonal-diagonal di bidang XY, XZ, dan YZ. Terakhir, fungsi menghitung deviasi pada empat diagonal utama yang melintasi kubus 3D. Total deviasi ini dikembalikan sebagai ukuran seberapa jauh konfigurasi kubus dari susunan ideal yang diharapkan dalam magic cube.

```
function calculateScore(cube, magicNumber) {
  const n = cube.length;

  let totalDeviation = 0;

  // 1. BAGIAN X
  for (let z = 0; z < n; z++) {
    for (let y = 0; y < n; y++) {
      let rowSum = 0;
      for (let x = 0; x < n; x++) {
        rowSum += cube[z][y][x];
      }
      totalDeviation += Math.abs(rowSum - magicNumber);
    }
  }

  // 2. BAGIAN Y
  for (let z = 0; z < n; z++) {
    for (let x = 0; x < n; x++) {
      let colSum = 0;
```

```
      for (let y = 0; y < n; y++) {
        colSum += cube[z][y][x];
      }
      totalDeviation += Math.abs(colSum - magicNumber);
    }
  }

  // 3. BAGIAN Z
  for (let y = 0; y < n; y++) {
    for (let x = 0; x < n; x++) {
      let pillarSum = 0;
      for (let z = 0; z < n; z++) {
        pillarSum += cube[z][y][x];
      }
      totalDeviation += Math.abs(pillarSum - magicNumber);
    }
  }

  // 4. DIAGONAL
  // a) XY
  for (let z = 0; z < n; z++) {
    let diagSum1 = 0;
    let diagSum2 = 0;
    for (let i = 0; i < n; i++) {
      diagSum1 += cube[z][i][i];
      diagSum2 += cube[z][i][n - i - 1];
    }
    totalDeviation += Math.abs(diagSum1 - magicNumber);
    totalDeviation += Math.abs(diagSum2 - magicNumber);
  }

  // b) XZ
  for (let y = 0; y < n; y++) {
    let diagSum1 = 0;
    let diagSum2 = 0;
    for (let i = 0; i < n; i++) {
      diagSum1 += cube[i][y][i];
      diagSum2 += cube[n - i - 1][y][i];
    }
    totalDeviation += Math.abs(diagSum1 - magicNumber);
    totalDeviation += Math.abs(diagSum2 - magicNumber);
  }

  // c) YZ
  for (let x = 0; x < n; x++) {
    let diagSum1 = 0;
    let diagSum2 = 0;
    for (let i = 0; i < n; i++) {
      diagSum1 += cube[i][i][x];
      diagSum2 += cube[i][n - i - 1][x];
    }
    totalDeviation += Math.abs(diagSum1 - magicNumber);
    totalDeviation += Math.abs(diagSum2 - magicNumber);
  }

  // 5. XYZ DIAGONAL
```

```
  let diagSpace1 = 0;
  let diagSpace2 = 0;
  let diagSpace3 = 0;
  let diagSpace4 = 0;
  for (let i = 0; i < n; i++) {
    diagSpace1 += cube[i][i][i];
    diagSpace2 += cube[i][i][n - i - 1];
    diagSpace3 += cube[i][n - i - 1][i];
    diagSpace4 += cube[n - i - 1][i][i];
  }
  totalDeviation += Math.abs(diagSpace1 - magicNumber);
  totalDeviation += Math.abs(diagSpace2 - magicNumber);
  totalDeviation += Math.abs(diagSpace3 - magicNumber);
  totalDeviation += Math.abs(diagSpace4 - magicNumber);

  return totalDeviation;
}
```

Fungsi generateRandomData

Fungsi generateRandomData bertujuan untuk membuat data kubus 3D berukuran gridSize x gridSize x gridSize dengan angka unik dari 1 hingga jumlah total elemen dalam kubus (totalCubes). Angka-angka tersebut pertama kali dimasukkan ke dalam array numbers, kemudian diacak-. Selanjutnya, fungsi ini mengisi array data 3D dengan elemen-elemen acak tersebut. Hasil akhir berupa kubus 3D acak yang diisi dengan angka-angka unik dari 1 hingga totalCubes.

```
function generateRandomData() {
  const data = [];
  const totalCubes = gridSize * gridSize * gridSize;

  // Create an array with numbers from 1 to totalCubes (1 to 125)
  const numbers = [];
  for (let i = 1; i <= totalCubes; i++) {
    numbers.push(i);
  }

  let currentIndex = numbers.length;

  console.log(numbers);

  // While there remain elements to shuffle...
  while (currentIndex != 0) {
    // Pick a remaining element...
    let randomIndex = Math.floor(Math.random() * currentIndex);
    currentIndex--;

    // And swap it with the current element.
    [numbers[currentIndex], numbers[randomIndex]] = [
      numbers[randomIndex],
```

```
      numbers[currentIndex],
    ];
  }

  // Fill the data array with shuffled numbers
  let index = 0;
  for (let x = 0; x < gridSize; x++) {
    data[x] = [];
    for (let y = 0; y < gridSize; y++) {
      data[x][y] = [];
      for (let z = 0; z < gridSize; z++) {
        data[x][y][z] = numbers[index];
        index++;
      }
    }
  }

  return data;
}
```

Fungsi calculateScore

Fungsi calculateScore digunakan untuk menghitung total deviasi dari kubus 3D tertentu dibandingkan dengan angka magicNumber yang diharapkan, yang merupakan jumlah target elemen dalam setiap baris, kolom, dan diagonal kubus. Pertama, fungsi ini menghitung deviasi pada setiap baris di sepanjang sumbu X, kolom di sepanjang sumbu Y, dan tiang di sepanjang sumbu Z. Selanjutnya, fungsi ini menghitung deviasi pada diagonal-diagonal di bidang XY, XZ, dan YZ, masing-masing dalam dua arah diagonal. Terakhir, fungsi ini juga menghitung deviasi pada empat diagonal utama di seluruh ruang 3D. Semua deviasi dihitung sebagai selisih absolut dari magicNumber, kemudian dijumlahkan dan dikembalikan sebagai nilai totalDeviation, yang mencerminkan seberapa jauh susunan elemen dalam kubus menyimpang dari susunan ideal magic cube.

```
function calculateScore(cube, magicNumber) {
  const n = cube.length;

  let totalDeviation = 0;

  // 1. BAGIAN X
  for (let z = 0; z < n; z++) {
    for (let y = 0; y < n; y++) {
      let rowSum = 0;
      for (let x = 0; x < n; x++) {
        rowSum += cube[z][y][x];
      }
      totalDeviation += Math.abs(rowSum - magicNumber);
    }
```

```
}

// 2. BAGIAN Y
for (let z = 0; z < n; z++) {
  for (let x = 0; x < n; x++) {
    let colSum = 0;
    for (let y = 0; y < n; y++) {
      colSum += cube[z][y][x];
    }
    totalDeviation += Math.abs(colSum - magicNumber);
  }
}

// 3. BAGIAN Z
for (let y = 0; y < n; y++) {
  for (let x = 0; x < n; x++) {
    let pillarSum = 0;
    for (let z = 0; z < n; z++) {
      pillarSum += cube[z][y][x];
    }
    totalDeviation += Math.abs(pillarSum - magicNumber);
  }
}

// 4. DIAGONAL
// a) XY
for (let z = 0; z < n; z++) {
  let diagSum1 = 0;
  let diagSum2 = 0;
  for (let i = 0; i < n; i++) {
    diagSum1 += cube[z][i][i];
    diagSum2 += cube[z][i][n - i - 1];
  }
  totalDeviation += Math.abs(diagSum1 - magicNumber);
  totalDeviation += Math.abs(diagSum2 - magicNumber);
}

// b) XZ
for (let y = 0; y < n; y++) {
  let diagSum1 = 0;
  let diagSum2 = 0;
  for (let i = 0; i < n; i++) {
    diagSum1 += cube[i][y][i];
    diagSum2 += cube[n - i - 1][y][i];
  }
  totalDeviation += Math.abs(diagSum1 - magicNumber);
  totalDeviation += Math.abs(diagSum2 - magicNumber);
}

// c) YZ
for (let x = 0; x < n; x++) {
  let diagSum1 = 0;
  let diagSum2 = 0;
  for (let i = 0; i < n; i++) {
    diagSum1 += cube[i][i][x];
    diagSum2 += cube[i][n - i - 1][x];
```

```
    }
    totalDeviation += Math.abs(diagSum1 - magicNumber);
    totalDeviation += Math.abs(diagSum2 - magicNumber);
  }

  // 5. XYZ DIAGONAL
  let diagSpace1 = 0;
  let diagSpace2 = 0;
  let diagSpace3 = 0;
  let diagSpace4 = 0;
  for (let i = 0; i < n; i++) {
    diagSpace1 += cube[i][i][i];
    diagSpace2 += cube[i][i][n - i - 1];
    diagSpace3 += cube[i][n - i - 1][i];
    diagSpace4 += cube[n - i - 1][i][i];
  }
  totalDeviation += Math.abs(diagSpace1 - magicNumber);
  totalDeviation += Math.abs(diagSpace2 - magicNumber);
  totalDeviation += Math.abs(diagSpace3 - magicNumber);
  totalDeviation += Math.abs(diagSpace4 - magicNumber);

  return totalDeviation;
}
```

Fungsi findBestNeighbor

Fungsi findBestNeighbor digunakan untuk menemukan tetangga terbaik dari kubus 3D yang diberikan dengan mencoba semua kemungkinan pertukaran dua elemen di dalam kubus. Pertama, fungsi ini menghitung nilai deviasi awal dari kubus saat ini menggunakan calculateScore. Kemudian, fungsi ini melakukan iterasi melalui semua pasangan koordinat (i1, j1, k1) dan (i2, j2, k2) di dalam kubus, dan untuk setiap pasangan yang berbeda, fungsi ini menukar elemen-elemen pada kedua posisi tersebut. Setelah melakukan pertukaran, fungsi ini menghitung skor baru kubus untuk mengevaluasi apakah pertukaran tersebut mengurangi deviasi total dari magicNumber. Jika skor yang diperoleh lebih baik (lebih kecil) daripada skor saat ini, fungsi ini mencatat pasangan posisi sebagai tetangga terbaik. Setelah selesai mengiterasi semua kemungkinan pasangan, fungsi mengembalikan tetangga terbaik yang ditemukan bersama nilai deviasinya.

```
function findBestNeighbor(cube) {
  const n = cube.length;
  const magicNumber = calculateMagicNumber(n);
  let currentValue = calculateScore(cube, magicNumber);
  let bestValue = currentValue;
```

```
  let bestNeighbor = null;

  // Check all possible pairs of positions in the cube
  for (let i1 = 0; i1 < gridSize; i1++) {
    for (let j1 = 0; j1 < gridSize; j1++) {
      for (let k1 = 0; k1 < gridSize; k1++) {
        for (let i2 = 0; i2 < gridSize; i2++) {
          for (let j2 = 0; j2 < gridSize; j2++) {
            for (let k2 = 0; k2 < gridSize; k2++) {
              // Skip if the positions are the same
              if (i1 === i2 && j1 === j2 && k1 === k2) {
                continue;
              }

              // Swap the two positions
              swapElement(cube, i1, j1, k1, i2, j2, k2);

              // Calculate objective function after swap
              let newValue = calculateScore(cube, magicNumber);

              // Record the best neighbor if an improvement is found
              if (newValue <= bestValue) {
                bestValue = newValue;
                bestNeighbor = [
                  [k1, j1, i1],
                  [k2, j2, i2],
                ];
              }
              swapElement(cube, i2, j2, k2, i1, j1, k1);
            }
          }
        }
      }
    }
  }

  return { bestNeighbor, bestValue };
}
```

Fungsi hillClimbSteepest

Fungsi hillClimbSteepest dimulai dengan menghasilkan kubus acak sebagai solusi awal. Fungsi ini menghitung nilai magicNumber sebagai target jumlah elemen di tiap baris, kolom, dan diagonal. Solusi saat ini dan skor awal dihitung menggunakan calculateScore. Kemudian, fungsi memulai iterasi untuk mencari tetangga terbaik dari solusi saat ini dengan memanggil findBestNeighbor.

Jika tetangga terbaik yang ditemukan memiliki skor lebih rendah daripada skor solusi saat ini, fungsi menukar elemen dalam solusi, memperbarui skor, menyimpan data

solusi terbaik ke dalam cubeDataSets, dan mencatat pergerakan ke dalam moveDataSets. Skor dari setiap iterasi juga disimpan dalam array scores untuk analisis performa. Fungsi updateCubes dipanggil untuk memperbarui tampilan visual kubus, dan terdapat jeda yang dikendalikan dengan delayTime. Jika tidak ada perbaikan lebih lanjut (skor tidak membaik), iterasi dihentikan. Fungsi ini mengembalikan solusi terbaik yang ditemukan, skor terbaik, dan riwayat skor selama proses.

```
async function hillClimbSteepest() {
  let cube = generateRandomData();

  paused = false;
  canceled = false;

  // Clear data
  cubeDataSets = [];
  moveDataSets = [];
  let scores = [];
  currentDataIndex = -1;

  cubeDataSets.push(cube);
  currentDataIndex++;
  updateCubes();

  const n = cube.length;
  const magicNumber = calculateMagicNumber(n);

  let currentSolution = cube;
  let currentScore = calculateScore(currentSolution, magicNumber);

  let bestSolution = currentSolution;
  let bestScore = currentScore;

  while (true) {
    if (canceled) {
      console.log("Simulation canceled.");
      break;
    }

    while (paused) {
      await new Promise((resolve) => setTimeout(resolve, 100));
    }
    let { bestNeighbor, bestValue: newScore } =
      findBestNeighbor(currentSolution);

    if (bestNeighbor) {
      const [[z1, y1, x1], [z2, y2, x2]] = bestNeighbor;
      if (newScore >= currentScore) {
        break;
      }

      swapElement(currentSolution, x1, y1, z1, x2, y2, z2);
```

```
      // Next step
      currentScore = newScore;
      bestSolution = JSON.parse(JSON.stringify(currentSolution)); //
Deep copy
      bestScore = currentScore;
      scores.push(bestScore);

      // Add deep copy to dataset
      cubeDataSets.push(JSON.parse(JSON.stringify(bestSolution)));
      moveDataSets.push([
        [z1, y1, x1],
        [z2, y2, x2],
      ]);
      currentDataIndex = cubeDataSets.length - 1;
      console.log(bestScore);

      updateCubes();

      await new Promise((resolve) => setTimeout(resolve, delayTime));
      console.log("5-second delay complete.");
    } else {
      break;
    }
  }

  updateCubes();

  return { solution: bestSolution, bestScore: bestScore, scores: scores
};
}
```

Fungsi hillClimbSideways

Fungsi hillClimbSideways dimulai dengan membuat kubus acak dan inisialisasi variabel kontrol seperti paused, canceled, dan sidewaysCount untuk menghitung jumlah langkah sideways yang diambil. Langkah pertama adalah menghitung nilai magicNumber dan skor awal solusi saat ini dengan memanfaatkan fungsi calculateScore. Solusi awal kemudian disimpan dalam cubeDataSets dan di-update pada antarmuka visual melalui updateCubes.

Dalam perulangan, fungsi mencari tetangga terbaik dari solusi saat ini menggunakan findBestNeighbor. Jika skor tetangga terbaik sama dengan skor saat ini, maka langkah sideways diizinkan dengan peningkatan sidewaysCount. Jika skor tetangga lebih rendah, currentScore diperbarui, dan sidewaysCount diatur ulang. Jika tidak ada perbaikan atau batas langkah sideways tercapai, perulangan dihentikan.

Fungsi mengembalikan solusi terbaik yang ditemukan, skor terbaik yang dicapai, dan riwayat skor selama proses pencarian.

```
async function hillClimbSideways() {
  let cube = generateRandomData();

  paused = false;
  canceled = false;

  // Clear data
  cubeDataSets = [];
  moveDataSets = [];
  let scores = [];
  currentDataIndex = -1;

  cubeDataSets.push(cube);
  currentDataIndex++;
  updateCubes();

  const n = cube.length;
  const magicNumber = calculateMagicNumber(n);

  let currentSolution = cube;
  let currentScore = calculateScore(currentSolution, magicNumber);

  let bestSolution = currentSolution;
  let bestScore = currentScore;

  let sidewaysCount = 0;
  console.log(maxSideways);
  while (sidewaysCount < maxSideways) {
    if (canceled) {
      console.log("Simulation canceled.");
      break;
    }

    while (paused) {
      await new Promise((resolve) => setTimeout(resolve, 100));
      await new Promise((resolve) => setTimeout(resolve, 100));
    }

    let { bestNeighbor, bestValue: newScore } =
      findBestNeighbor(currentSolution);

    if (bestNeighbor) {
      const [[z1, y1, x1], [z2, y2, x2]] = bestNeighbor;
      if (newScore === currentScore) {
        sidewaysCount++;
        console.log(sidewaysCount);
      } else if (newScore < currentScore) {
        currentScore = newScore;
        sidewaysCount = 0;
      } else {
        break;
      }
```

```
      swapElement(currentSolution, x1, y1, z1, x2, y2, z2);

      // Next step
      currentScore = newScore;
      bestSolution = JSON.parse(JSON.stringify(currentSolution)); //
Deep copy
      bestScore = currentScore;
      scores.push(bestScore);

      // Add deep copy to dataset
      cubeDataSets.push(JSON.parse(JSON.stringify(bestSolution)));
      moveDataSets.push([
        [z1, y1, x1],
        [z2, y2, x2],
      ]);
      currentDataIndex = cubeDataSets.length - 1;
      console.log(bestScore);

      updateCubes();

      await new Promise((resolve) => setTimeout(resolve, delayTime));
      console.log("Step complete.");
      await new Promise((resolve) => setTimeout(resolve, 100));
      console.log("100ms delay complete.");
    } else {
      console.log("No better neighbor found, stopping.");
      break;
    }
  }

  updateCubes();

  return { solution: bestSolution, bestScore: bestScore, scores: scores
};
}
```

Fungsi HillClimbRandomRestart

Fungsi HillClimbRandomRestart mencoba mencari solusi optimal dengan melakukan beberapa kali restart independen dari posisi acak. Proses dimulai dengan menginisialisasi variabel untuk menyimpan solusi terbaik secara keseluruhan (bestOverallSolution) dan skornya (bestOverallScore). Fungsi mengulangi pencarian sebanyak restarts yang ditentukan. Setiap iterasi restart memulai ulang pencarian dari kubus acak baru yang dihasilkan oleh generateRandomData.

Selama pencarian di setiap restart, fungsi mencari tetangga terbaik dari solusi saat ini menggunakan findBestNeighbor. Jika ditemukan perbaikan pada solusi, elemen dalam

kubus ditukar, dan solusi baru disimpan beserta skor terbarunya. Jika tidak ada perbaikan yang lebih baik, pencarian berhenti dan restart baru dimulai.

Fungsi mengembalikan solusi terbaik yang ditemukan selama semua restart, skor terbaik yang dicapai, riwayat skor, dan jumlah iterasi per restart.

```
async function HillClimbRandomRestart(restarts = 5) {
  let bestOverallSolution = null;
  let bestOverallScore = Infinity;
  let scores = [];
  let perRestartIterations = [];
  let iterationCount = 0;

  paused = false;
  canceled = false;

  // Clear data
  cubeDataSets = [];
  moveDataSets = [];
  currentDataIndex = -1;

  for (let restart = 0; restart < restarts; restart++) {
    console.log(`Restart #${restart + 1}`);
    reset();

    let cube = generateRandomData();
    cubeDataSets.push(cube);
    currentDataIndex++;
    updateCubes();

    const n = cube.length;
    const magicNumber = calculateMagicNumber(n);

    let currentSolution = cube;
    let currentScore = calculateScore(currentSolution, magicNumber);
    let bestSolution = JSON.parse(JSON.stringify(currentSolution));
    let bestScore = currentScore;
    let restartScores = [];
    let iterationCounter = 0;

    while (true) {
      if (canceled) {
        console.log("Simulation canceled.");
        return {
          solution: bestOverallSolution,
          bestScore: bestOverallScore,
          scores,
          perRestartIterations,
        };
      }

      while (paused) {
        await new Promise((resolve) => setTimeout(resolve, 100));
      }
```

```
    let { bestNeighbor, bestValue: newScore } =
      findBestNeighbor(currentSolution);

    if (bestNeighbor) {
      const [[z1, y1, x1], [z2, y2, x2]] = bestNeighbor;
      if (newScore >= currentScore) {
        break;
      }

      swapElement(currentSolution, x1, y1, z1, x2, y2, z2);

      currentScore = newScore;
      bestSolution = JSON.parse(JSON.stringify(currentSolution));
      bestScore = currentScore;
      restartScores.push(bestScore);
      iterationCounter++;
      iterationCount++;

      cubeDataSets.push(JSON.parse(JSON.stringify(bestSolution)));
      moveDataSets.push([
        [z1, y1, x1],
        [z2, y2, x2],
      ]);
      currentDataIndex = cubeDataSets.length - 1;
      updateCubes();

      await new Promise((resolve) => setTimeout(resolve, delayTime));
    } else {
      break;
    }
  }

  perRestartIterations.push(iterationCounter);

  scores.push(...restartScores);

  if (bestScore < bestOverallScore) {
    bestOverallScore = bestScore;
    bestOverallSolution = JSON.parse(JSON.stringify(bestSolution));
  }

  console.log(`Best score for restart #${restart + 1}: ${bestScore}`);
}

console.log("Best overall solution found:", bestOverallSolution);
console.log("Best overall score:", bestOverallScore);

updateCubes();

return {
  solution: bestOverallSolution,
  bestScore: bestOverallScore,
  scores: scores,
  perRestartIterations: perRestartIterations,
};
```

```
}
```

Fungsi hillClimbStochastic

Fungsi hillClimbStochastic menggunakan pendekatan acak dalam pencarian solusi untuk menghindari jebakan local optima. Fungsi dimulai dengan membuat kubus acak menggunakan generateRandomData dan inisialisasi variabel seperti cubeDataSets untuk menyimpan data kubus selama iterasi. Variabel bestSolution dan bestScore digunakan untuk melacak solusi terbaik yang ditemukan selama proses.

Dalam loop iteratif hingga batas maxIterations tercapai, algoritma secara acak memilih dua elemen dalam kubus untuk ditukar posisinya menggunakan swapRandomElements, lalu menghitung skor dari solusi baru tersebut dengan calculateScore. Jika skor solusi baru lebih buruk atau sama dengan solusi sebelumnya, pertukaran dibatalkan dengan memanggil swapElement untuk mengembalikan posisi awal. Jika skor lebih baik, solusi baru diterima, disimpan, dan dimasukkan ke dalam cubeDataSets dan moveDataSets untuk keperluan visualisasi. Fungsi mengembalikan solusi terbaik yang ditemukan, skor terbaik, dan daftar skor setiap iterasi.

```
async function hillClimbStochastic() {
  let cube = generateRandomData();

  paused = false;
  canceled = false;

  // Clear data
  cubeDataSets = [];
  moveDataSets = [];
  let scores = [];
  let iterations = 0;
  currentDataIndex = -1;

  cubeDataSets.push(cube);
  currentDataIndex++;
  updateCubes();

  const n = cube.length;
  const magicNumber = calculateMagicNumber(n);

  let currentSolution = cube;
  let currentScore = calculateScore(currentSolution, magicNumber);

  let bestSolution = currentSolution;
  let bestScore = currentScore;
```

```
  const maxIterations = 200;

  while (iterations < maxIterations) {
    if (canceled) {
      console.log("Simulation canceled.");
      break;
    }

    while (paused) {
      await new Promise((resolve) => setTimeout(resolve, 100));
    }

    // Increment iteration count
    iterations++;

    // Randomly swap two elements and evaluate new score
    const [[z1, y1, x1], [z2, y2, x2]] =
swapRandomElements(currentSolution);
    const newScore = calculateScore(currentSolution, magicNumber);
    scores.push(newScore);

    // If the new solution is worse or equal, revert the swap
    if (newScore >= currentScore) {
      swapElement(currentSolution, x2, y2, z2, x1, y1, z1);
    } else {
      // Accept the new solution
      currentScore = newScore;
      bestSolution = JSON.parse(JSON.stringify(currentSolution)); //
Deep copy
      bestScore = currentScore;

      // Add deep copy to dataset
      cubeDataSets.push(JSON.parse(JSON.stringify(bestSolution)));
      moveDataSets.push([
        [z1, y1, x1],
        [z2, y2, x2],
      ]);
      currentDataIndex = cubeDataSets.length - 1;
      console.log(`Iteration ${iterations}: Best Score = ${bestScore}`);

      updateCubes();

      await new Promise((resolve) => setTimeout(resolve, delayTime));
    }
  }

  updateCubes();

  return { solution: bestSolution, bestScore: bestScore, scores: scores
};
}
```

Fungsi simulatedAnnealing

Fungsi simulatedAnnealing dimulai dengan menghasilkan kubus acak sebagai solusi awal menggunakan fungsi generateRandomData. Kondisi awal seperti paused dan canceled diinisialisasi untuk mengontrol eksekusi algoritma, serta variabel cubeDataSets dan moveDataSets digunakan untuk menyimpan data solusi dan pergerakan.

Parameter penting yang digunakan dalam algoritma ini adalah initialTemp (temperatur awal) dan coolingRate (laju pendinginan), yang mengontrol penurunan temperatur selama iterasi. Nilai magicNumber dihitung berdasarkan ukuran kubus untuk mengevaluasi solusi. Solusi awal dikloning menjadi currentSolution, dan skornya dihitung dengan calculateScore untuk mendapatkan currentScore dan bestScore.

Dalam loop utama, algoritma membuat solusi sementara tempSolution dengan menukar elemen secara acak menggunakan swapRandomElements. Skor solusi baru dihitung, dan nilai ET (eksponen probabilitas penerimaan) dihitung sebagai Math.exp(-deltaScore / currentTemp). Jika ET lebih besar dari 1, maka diatur ke 1. Solusi baru diterima jika deltaScore negatif (meningkatkan solusi) atau jika probabilitas acak lebih kecil dari ET, memungkinkan penerimaan solusi yang lebih buruk untuk menghindari local optima.

Temperatur currentTemp dikurangi pada setiap iterasi dengan faktor coolingRate. Jika solusi baru lebih baik dari solusi terbaik sebelumnya, maka currentSolution dan bestSolution diperbarui. Dataset solusi ditambahkan ke cubeDataSets dan moveDataSets untuk keperluan visualisasi dan pelacakan pergerakan.

Algoritma berlanjut hingga currentTemp mencapai batas yang ditentukan (0.0001), atau jika operasi dibatalkan. Fungsi mengembalikan solusi terbaik (bestSolution), skor terbaik (bestScore), array ET (etArray), dan array skor (scoresArray) untuk analisis lebih lanjut.

```
let initialTemp = 100;
let coolingRate = 0.9999;

async function simulatedAnnealing() {
  let cube = generateRandomData();

  paused = false;
  canceled = false;

  // Clear data
  cubeDataSets = [];
  moveDataSets = [];
  currentDataIndex = -1;

  cubeDataSets.push(cube);
  currentDataIndex++;
  updateCubes();

  const n = cube.length;
  const magicNumber = calculateMagicNumber(n);

  let currentTemp = initialTemp;
  let currentSolution = JSON.parse(JSON.stringify(cube));
  let currentScore = calculateScore(currentSolution, magicNumber);

  let bestSolution = JSON.parse(JSON.stringify(currentSolution)); //
Deep copy
  let bestScore = currentScore;

  let etArray = [];
  let scoresArray = [];

  while (currentTemp > 0.0001) {
    if (canceled) {
      console.log("Simulation canceled.");
      break;
    }

    while (paused) {
      await new Promise((resolve) => setTimeout(resolve, 100));
    }

    // Create a temporary copy of the current solution
    let tempSolution = JSON.parse(JSON.stringify(currentSolution));
    const [[z1, y1, x1], [z2, y2, x2]] =
swapRandomElements(tempSolution);
    const newScore = calculateScore(tempSolution, magicNumber);

    scoresArray.push(currentScore);

    const deltaScore = newScore - currentScore;

    // Calculate ET (exponential acceptance probability)
    let et = Math.exp(-deltaScore / currentTemp);
    if (et > 1) {
      et = 1;
```

```
    }
    etArray.push(et);

    if (deltaScore < 0 || Math.random() < et) {
      let tempScore = newScore;
      if (tempScore < bestScore) {
        currentScore = tempScore;
        currentSolution = JSON.parse(JSON.stringify(tempSolution)); //
Accept the new solution
        bestSolution = JSON.parse(JSON.stringify(currentSolution)); //
Deep copy
        bestScore = currentScore;

        // Add deep copy to dataset
        cubeDataSets.push(JSON.parse(JSON.stringify(bestSolution)));
        moveDataSets.push([
          [z1, y1, x1],
          [z2, y2, x2],
        ]);

        updateCubes();
        currentDataIndex = cubeDataSets.length - 1;

        await new Promise((resolve) => setTimeout(resolve, delayTime));
      }
    }
    // Decrease the temperature
    currentTemp *= coolingRate;
  }

  console.log(bestSolution, bestScore);

  updateCubes();

  return {
    bestSolution: bestSolution,
    bestScore: bestScore,
    et: etArray,
    scores: scoresArray,
  };
}
```

Fungsi orderedCrossover

Fungsi orderedCrossover digunakan dalam algoritma genetika untuk menghasilkan dua keturunan (offspring) dari dua parent yang berbentuk kubus 3D. Pertama, fungsi menerima dua input parent dalam bentuk kubus dan mengonversinya menjadi array 1D untuk memudahkan proses crossover. Selanjutnya, fungsi menentukan segmen acak dalam array untuk menyalin elemen dari parent pertama ke anak pertama

dan elemen dari parent kedua ke anak kedua. Elemen-elemen yang tersisa dari parent lain kemudian diisi ke posisi kosong di array anak, memastikan bahwa setiap elemen unik tetap terjaga. Setelah pengisian selesai, array 1D dari kedua anak dikonversi kembali menjadi bentuk kubus 3D. Hasil akhir dari fungsi ini adalah dua anak dalam bentuk kubus 3D yang merupakan kombinasi dari elemen-elemen parent, menjaga keragaman populasi dalam algoritma genetika.

```
function orderedCrossover(parent1, parent2) {
  const gridSize = parent1.length;
  const totalElements = gridSize * gridSize * gridSize;

  // Flatten the 3D arrays for crossover
  const flatParent1 = parent1.flat(2);
  const flatParent2 = parent2.flat(2);

  const start = Math.floor(Math.random() * totalElements);
  const end = start + Math.floor(Math.random() * (totalElements -
start));

  // Create child with a slice from parent1
  const child1 = new Array(totalElements).fill(null);
  const child2 = new Array(totalElements).fill(null);

  for (let i = start; i < end; i++) {
    child1[i] = flatParent1[i];
    child2[i] = flatParent2[i];
  }

  let currentIndex1 = 0;
  let currentIndex2 = 0;

  for (let i = 0; i < totalElements; i++) {
    if (!child1.includes(flatParent2[i])) {
      while (child1[currentIndex1] !== null) {
        currentIndex1++;
      }
      child1[currentIndex1] = flatParent2[i];
    }
    if (!child2.includes(flatParent1[i])) {
      while (child2[currentIndex2] !== null) {
        currentIndex2++;
      }
      child2[currentIndex2] = flatParent1[i];
    }
  }

  const child3D1 = [];
  const child3D2 = [];
  let index = 0;

  for (let x = 0; x < gridSize; x++) {
    child3D1[x] = [];
```

```
    child3D2[x] = [];
    for (let y = 0; y < gridSize; y++) {
      child3D1[x][y] = [];
      child3D2[x][y] = [];
      for (let z = 0; z < gridSize; z++) {
        child3D1[x][y][z] = child1[index];
        child3D2[x][y][z] = child2[index];
        index++;
      }
    }
  }

  return [child3D1, child3D2];
}
```

Fungsi calculateFitness

Fungsi calculateFitness bertujuan menghitung tingkat kesesuaian (fitness) dari kubus 3D yang diberikan berdasarkan nilai magic number. Fungsi ini memanggil calculateScore untuk menghitung total deviasi kubus dari konfigurasi sempurna. Fitness diukur menggunakan skor yang lebih rendah sebagai indikasi solusi yang lebih baik, di mana semakin kecil deviasi, semakin tinggi fitness.

```
function calculateFitness(cube, magicNumber) {
  return calculateScore(cube, magicNumber);
}
```

Fungsi selectParents

Fungsi selectParents bertanggung jawab untuk memilih dua parent dari populasi untuk digunakan dalam proses crossover. Fungsi ini menggunakan mekanisme roulette wheel selection di mana probabilitas pemilihan setiap individu dihitung berdasarkan kebalikannya dari skor fitness (1 / (1 + nilai deviasi)), sehingga individu dengan skor lebih rendah memiliki peluang lebih tinggi untuk dipilih. Proses seleksi dilakukan dengan mengakumulasi probabilitas seleksi dan membandingkannya dengan angka acak untuk memilih parent secara proporsional.

```
function selectParents(population, fitnessValues) {
  const totalFitness = fitnessValues.reduce(
    (sum, value) => sum + 1 / (1 + value),
    0
  );
  const selectionProbability = fitnessValues.map(
```

```
    (value) => 1 / (1 + value) / totalFitness
  );

  const select = () => {
    const rand = Math.random();
    let cumulative = 0;
    for (let i = 0; i < population.length; i++) {
      cumulative += selectionProbability[i];
      if (rand <= cumulative) {
        return population[i];
      }
    }
    return population[population.length - 1]; // Fallback
  };

  return [select(), select()];
}
```

Fungsi mutate

Fungsi mutate bertujuan melakukan perubahan acak pada kubus untuk menjaga keragaman populasi. Mutasi terjadi dengan probabilitas tertentu (ditentukan oleh mutationRate). Jika mutasi dipilih, dua elemen acak dalam kubus ditukar posisinya. Hal ini membantu algoritma menghindari jebakan local optima dan memperluas eksplorasi ruang solusi.

```
function mutate(cube, mutationRate) {
  const gridSize = cube.length;
  if (Math.random() < mutationRate) {
    const x1 = Math.floor(Math.random() * gridSize);
    const y1 = Math.floor(Math.random() * gridSize);
    const z1 = Math.floor(Math.random() * gridSize);
    const x2 = Math.floor(Math.random() * gridSize);
    const y2 = Math.floor(Math.random() * gridSize);
    const z2 = Math.floor(Math.random() * gridSize);

    // Swap elements
    const temp = cube[x1][y1][z1];
    cube[x1][y1][z1] = cube[x2][y2][z2];
    cube[x2][y2][z2] = temp;
  }
}
```

Fungsi geneticAlgorithm

Fungsi geneticAlgorithm bertujuan untuk mencari solusi terbaik dari kubus 3D berukuran 5x5x5 dengan menggunakan pendekatan algoritma genetika. Parameter yang

digunakan meliputi populationSize untuk menentukan jumlah individu dalam populasi dan maxIterations sebagai batas iterasi maksimum yang dijalankan algoritma.

Pada awalnya, fungsi ini menghitung magic number kubus dan membuat populasi awal menggunakan fungsi generateInitialPopulation. Solusi terbaik diinisialisasi dengan individu pertama dari populasi, dan nilai fitness-nya dihitung dengan fungsi calculateFitness. Variabel cubeDataSets disiapkan untuk menyimpan data state terbaik dari setiap iterasi guna memudahkan visualisasi.

Di setiap iterasi, algoritma mengevaluasi fitness dari setiap individu dalam populasi dan memperbarui solusi terbaik jika ditemukan individu dengan skor lebih baik (lebih rendah). Solusi terbaik pada setiap iterasi disimpan di cubeDataSets untuk navigasi data.

Populasi baru dibentuk melalui proses seleksi parent menggunakan fungsi selectParents, diikuti dengan crossover menggunakan metode orderedCrossover. Anak hasil crossover kemudian mengalami mutasi acak melalui fungsi mutate dengan probabilitas tertentu (di sini 0.1). Populasi diperbarui dengan anak-anak hasil proses ini, menggantikan populasi lama.

Selama iterasi, visualisasi diperbarui menggunakan fungsi updateVisualization, dan jeda waktu diterapkan untuk mengatur laju pembaruan. Algoritma akan berhenti jika skor terbaik mencapai 0, menandakan solusi optimal ditemukan, atau jika jumlah iterasi maksimum tercapai. Fungsi mengembalikan solusi terbaik, skor terbaik, dan array scores yang berisi skor terbaik di setiap iterasi.

```
async function geneticAlgorithm(populationSize = 10, maxIterations =
1000) {
  const gridSize = 5; // Fixed grid size of 5 (5x5x5)
  const magicNumber = calculateMagicNumber(gridSize);
  let population = generateInitialPopulation(populationSize);
  let bestSolution = population[0];
  let bestScore = calculateFitness(bestSolution, magicNumber);

  // Data variables to store iterations
  cubeDataSets = []; // Clear previous data
  currentDataIndex = -1;
  let scores = [];

  for (let iteration = 0; iteration < maxIterations; iteration++) {
    if (canceled) {
      console.log("Simulation canceled.");
      break;
```

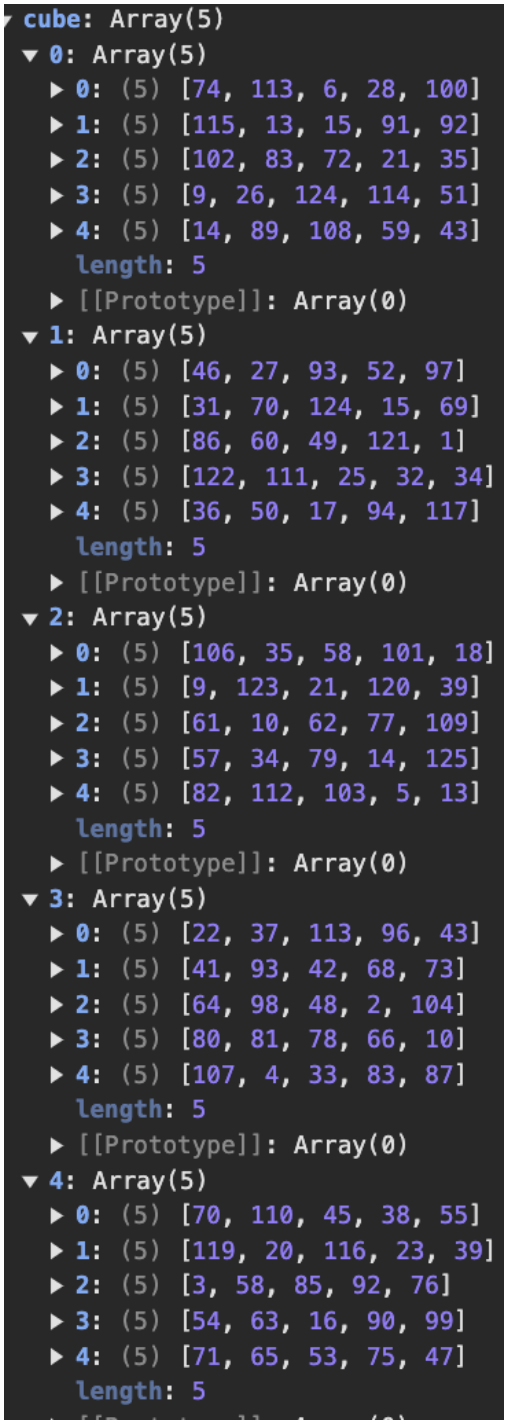
```
    }

    while (paused) {
      await new Promise((resolve) => setTimeout(resolve, 100));
    }

    const fitnessValues = population.map((individual) =>
      calculateFitness(individual, magicNumber)
    );

    // Find the best solution in the current population
    for (let i = 0; i < populationSize; i++) {
      const currentScore = fitnessValues[i];
      if (currentScore < bestScore) {
        bestSolution = JSON.parse(JSON.stringify(population[i])); //
Deep copy
        bestScore = currentScore;
      }
    }

    scores.push(bestScore);

    // Save current best state for navigation
    if (
      !cubeDataSets.some(
        (data) => JSON.stringify(data) === JSON.stringify(bestSolution)
      )
    ) {
      cubeDataSets.push(JSON.parse(JSON.stringify(bestSolution)));
      currentDataIndex = cubeDataSets.length - 1;
    }

    updateVisualization(bestSolution);
    await new Promise((resolve) => setTimeout(resolve, 100));

    // Create new population
    for (let i = 0; i < populationSize; i += 2) {
      const [parent1, parent2] = selectParents(population,
fitnessValues);
      let [child1, child2] = orderedCrossover(parent1, parent2);
      mutate(child1, 0.1);
      mutate(child2, 0.1);

      population[i] = child1;
      if (i + 1 < populationSize) {
        population[i + 1] = child2;
      }
    }

    console.log(`Iteration ${iteration + 1}: Best Score =
${bestScore}`);
    if (bestScore === 0) break;
  }

  return { bestSolution: bestSolution, bestScore: bestScore, scores:
scores };
```

```
}
```

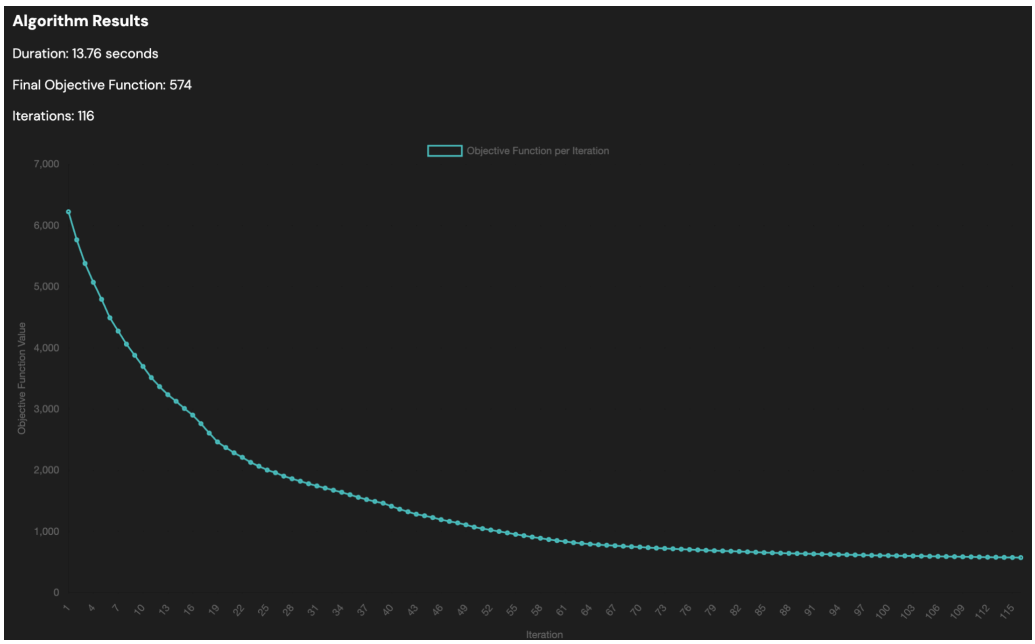
c. Hasil eksperimen

1. Steepest Ascent Hill Climbing

Percobaan 1	
State Awal:	State Akhir:

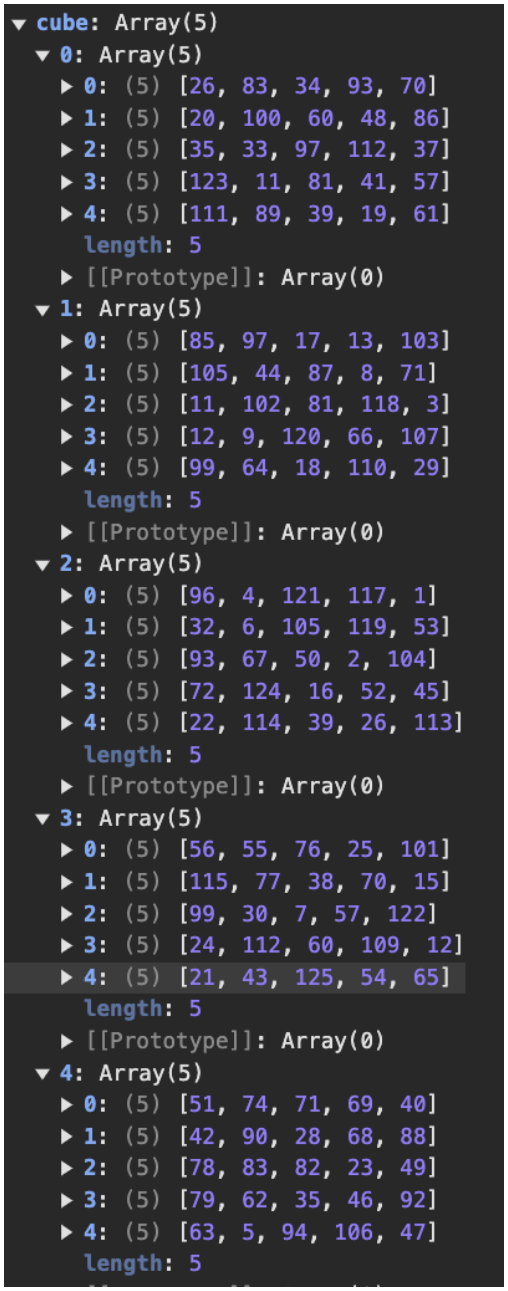

cube: Array(5)
0: Array(5)
0: (5) [74, 113, 6, 28, 100]
1: (5) [115, 13, 15, 91, 92]
2: (5) [102, 83, 72, 21, 35]
3: (5) [9, 26, 124, 114, 51]
4: (5) [14, 89, 108, 59, 43]
length: 5
[[Prototype]]: Array(0)
1: Array(5)
0: (5) [46, 27, 93, 52, 97]
1: (5) [31, 70, 124, 15, 69]
2: (5) [86, 60, 49, 121, 1]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

Hasil:

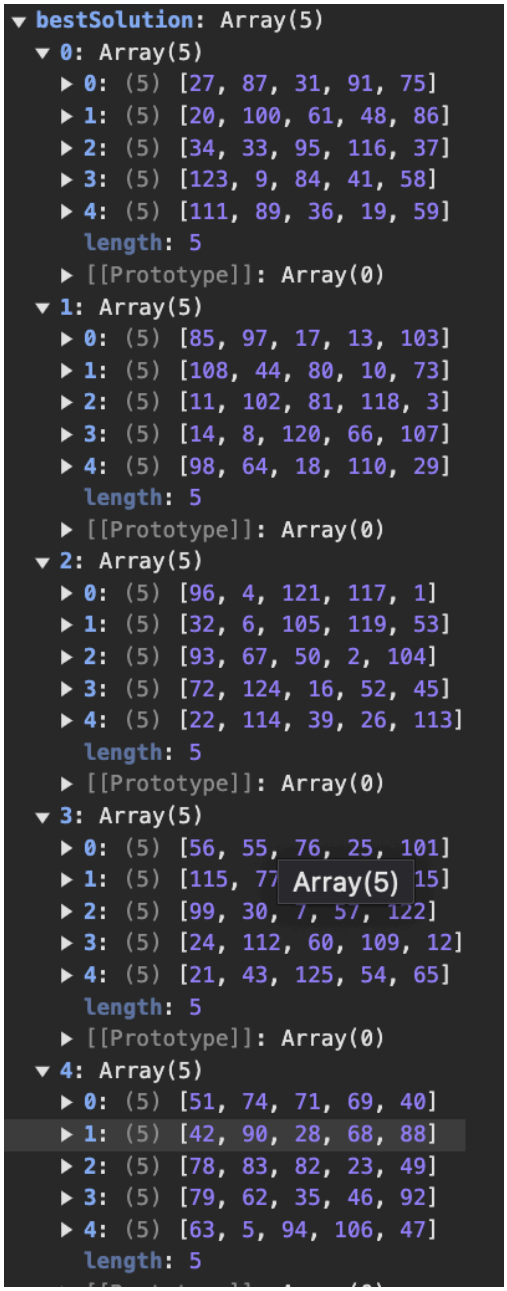

Algorithm Results
Duration: 13.76 seconds
Final Objective Function: 574
Iterations: 116
Objective Function per Iteration
Objective Function Value
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
Iteration

Percobaan 2

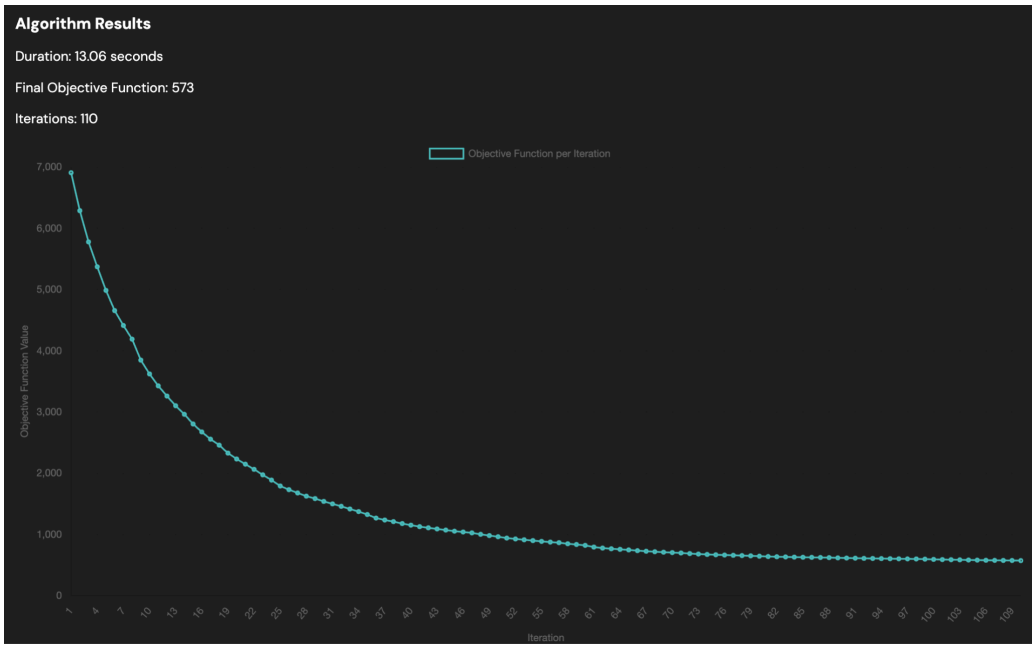
State Awal:



State Akhir:

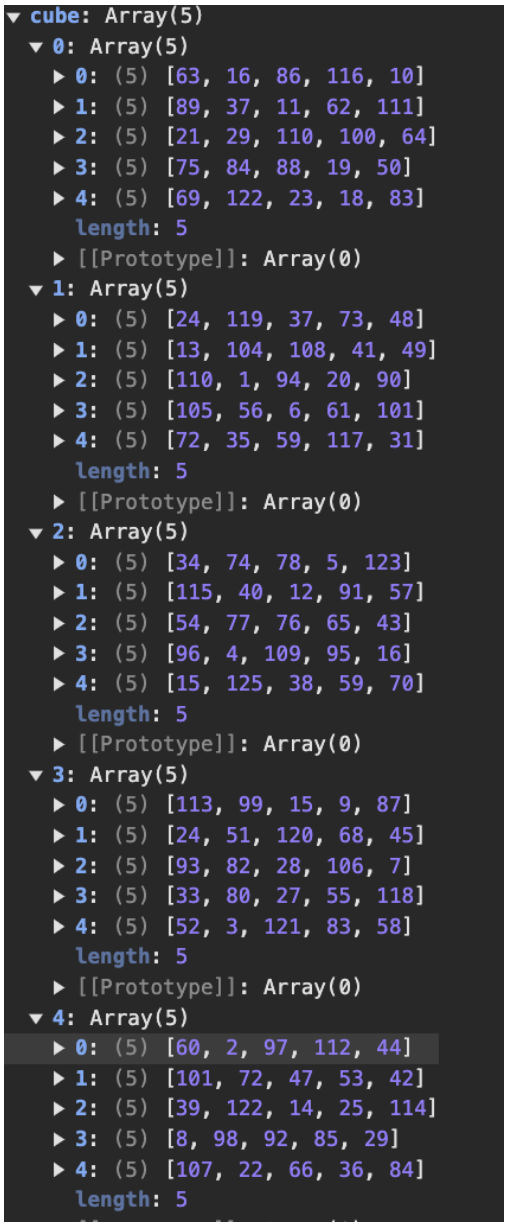


Hasil:

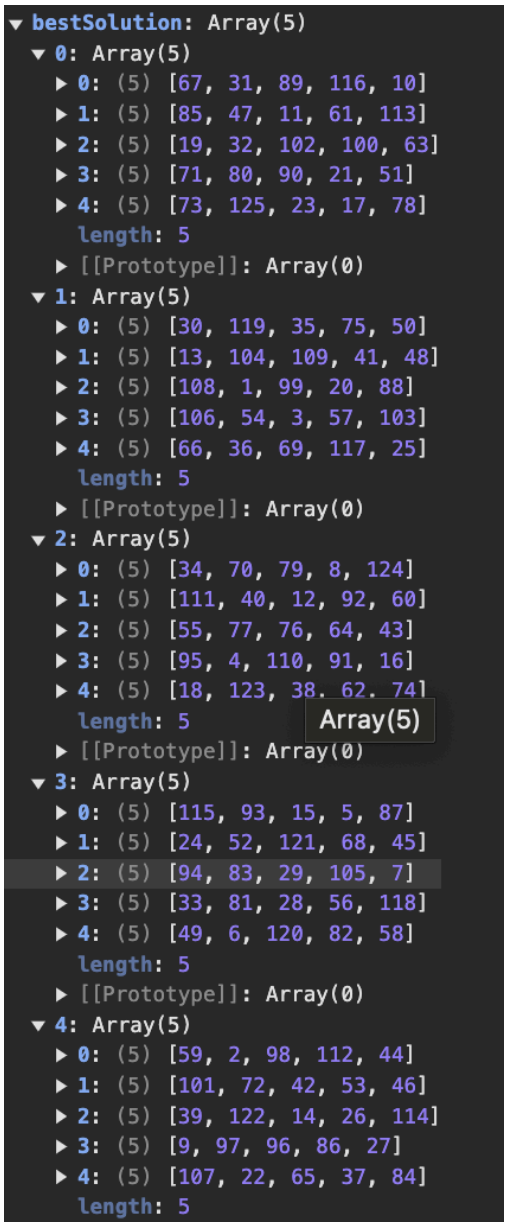

Algorithm Results
Duration: 13.06 seconds
Final Objective Function: 573
Iterations: 110
Objective Function per Iteration
Objective Function Value
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
1
4
7
10
13
16
19
22
25
28
31
34
37
40
43
46
49
52
55
58
61
64
67
70
73
76
79
82
85
88
91
94
97
100
103
106
109
Iteration

Percobaan 3

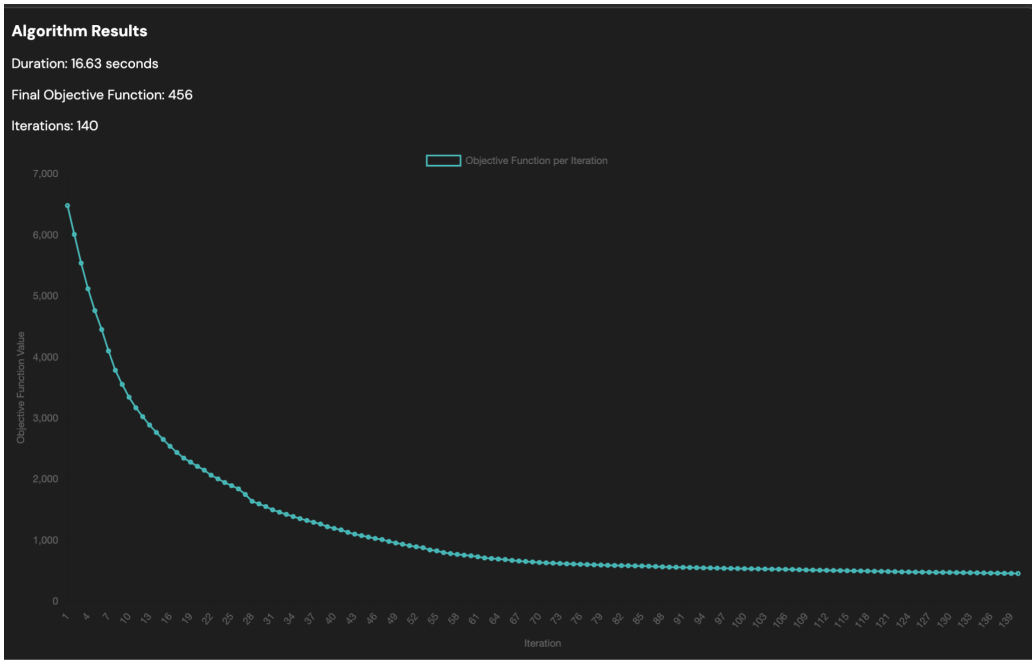
State Awal:



State Akhir:



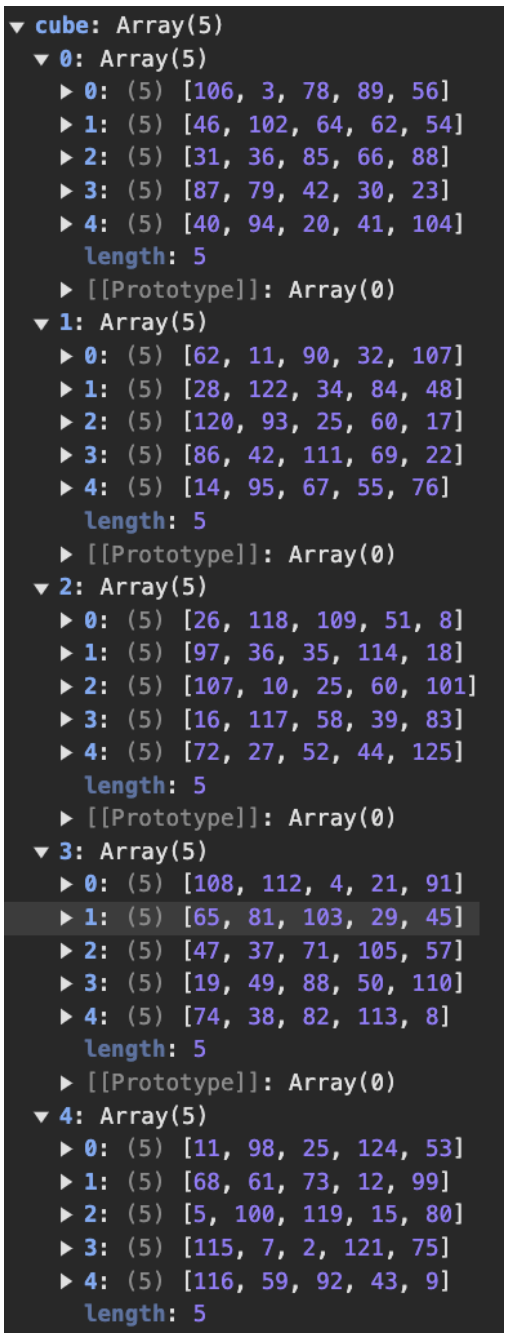
Hasil:


Algorithm Results
Duration: 16.63 seconds
Final Objective Function: 456
Iterations: 140
Objective Function per Iteration
Objective Function Value
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
1
4
7
10
13
16
19
22
25
28
31
34
37
40
43
46
49
52
55
58
61
64
67
70
73
76
79
82
85
88
91
94
97
100
103
106
109
112
115
118
121
124
127
130
133
136
139
Iteration

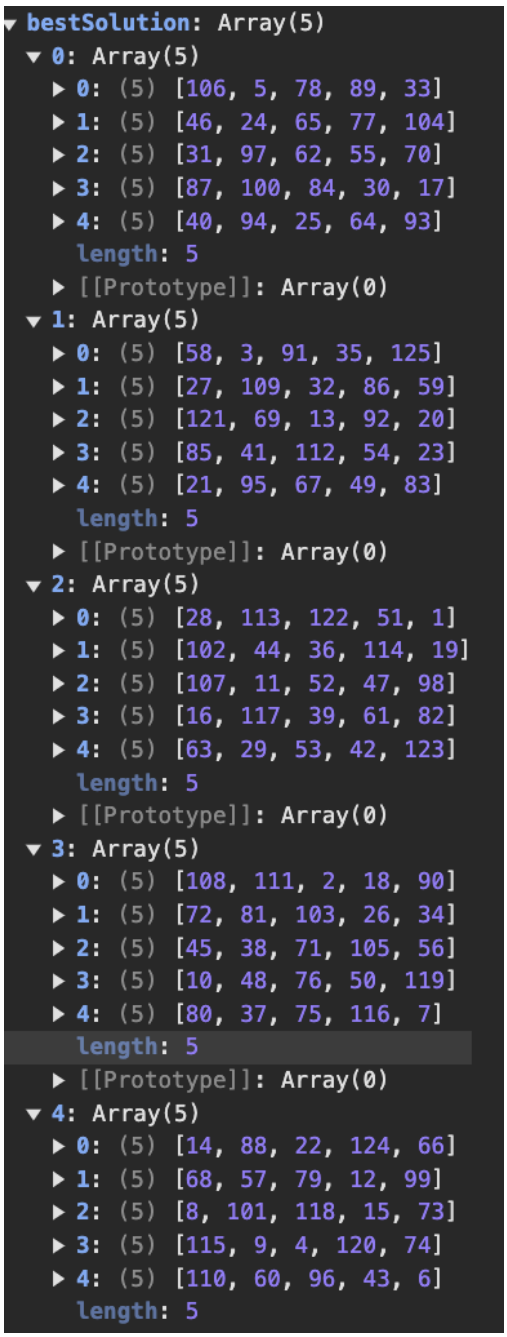
2. Sideways Move Hill Climbing (Banyak Sideways Move = 50)

Percobaan 1

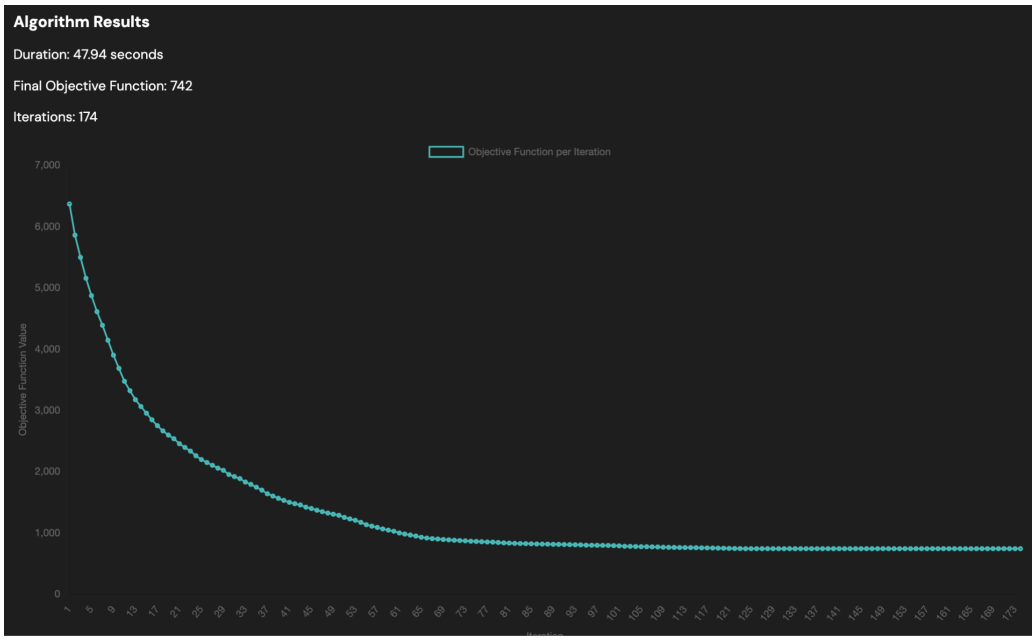
State Awal:



State Akhir:

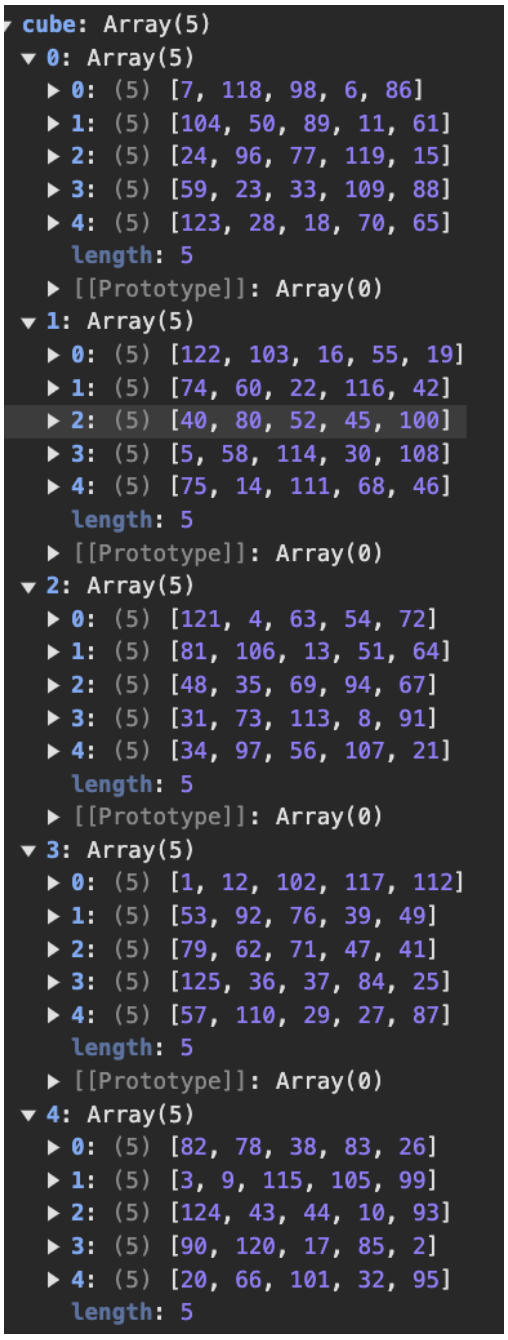


Hasil:

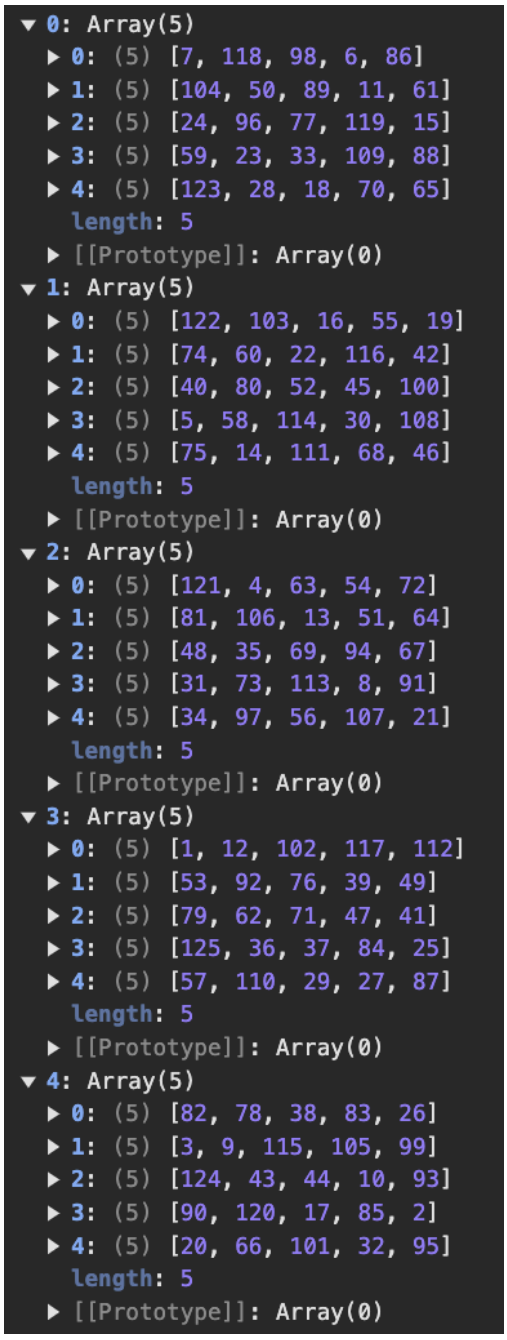

Algorithm Results
Duration: 47.94 seconds
Final Objective Function: 742
Iterations: 174
Objective Function per Iteration
Objective Function Value
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
1
5
9
13
17
21
25
29
33
37
41
45
49
53
57
61
65
69
73
77
81
85
89
93
97
101
105
109
113
117
121
125
129
133
137
141
145
149
153
157
161
165
169
173

Percobaan 2

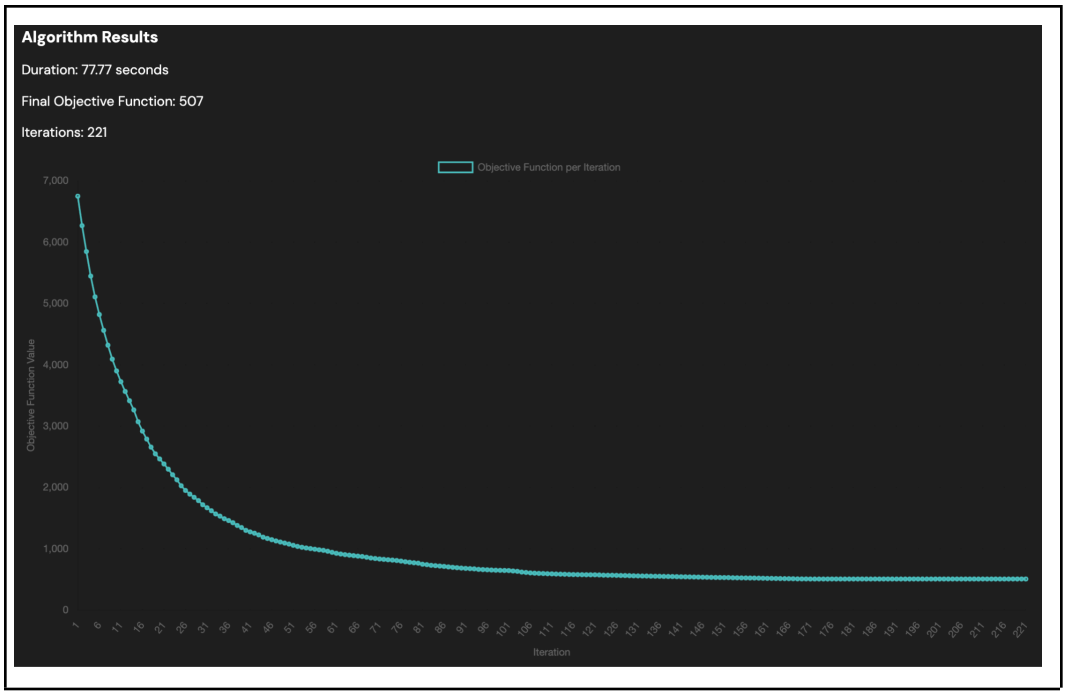
State Awal:



State Akhir:



Hasil:


Algorithm Results
Duration: 77.77 seconds
Final Objective Function: 507
Iterations: 221
Objective Function per Iteration
Objective Function Value
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
156
161
166
171
176
181
186
191
196
201
206
211
216
221
Iteration

Percobaan 3

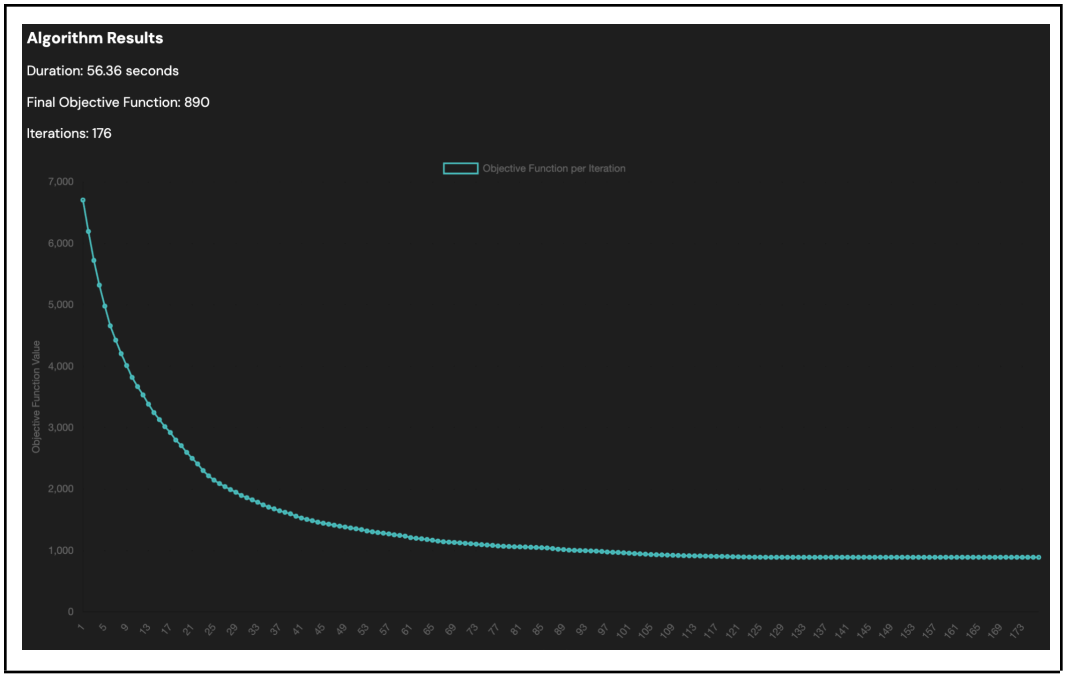
State Awal:

```
cube: Array(5)
▼ 0: Array(5)
  ▶ 0: (5) [112, 52, 77, 16, 59]
  ▶ 1: (5) [14, 46, 114, 119, 21]
  ▶ 2: (5) [81, 17, 60, 37, 123]
  ▶ 3: (5) [74, 111, 27, 19, 71]
  ▶ 4: (5) [36, 76, 15, 125, 58]
    length: 5
  ▶ [[Prototype]]: Array(0)
▼ 1: Array(5)
  ▶ 0: (5) [35, 22, 30, 98, 117]
  ▶ 1: (5) [55, 85, 2, 39, 89]
  ▶ 2: (5) [113, 92, 102, 86, 1]
  ▶ 3: (5) [104, 50, 118, 9, 33]
  ▶ 4: (5) [8, 75, 63, 78, 93]
    length: 5
  ▶ [[Prototype]]: Array(0)
▼ 2: Array(5)
  ▶ 0: (5) [97, 91, 56, 4, 65]
  ▶ 1: (5) [34, 23, 122, 120, 13]
  ▶ 2: (5) [53, 69, 29, 79, 72]
  ▶ 3: (5) [24, 6, 31, 124, 115]
  ▶ 4: (5) [96, 107, 76, 5, 51]
    length: 5
  ▶ [[Prototype]]: Array(0)
▼ 3: Array(5)
  ▶ 0: (5) [40, 66, 60, 109, 28]
  ▶ 1: (5) [103, 38, 54, 11, 108]
  ▶ 2: (5) [49, 67, 62, 41, 95]
  ▶ 3: (5) [10, 105, 92, 106, 18]
  ▶ 4: (5) [110, 44, 45, 48, 68]
    length: 5
  ▶ [[Prototype]]: Array(0)
▼ 4: Array(5)
  ▶ 0: (5) [20, 80, 89, 90, 36]
  ▶ 1: (5) [100, 116, 7, 26, 83]
  ▶ 2: (5) [12, 64, 94, 61, 79]
  ▶ 3: (5) [101, 51, 27, 57, 73]
  ▶ 4: (5) [82, 3, 114, 70, 43]
    length: 5
```

State Akhir:

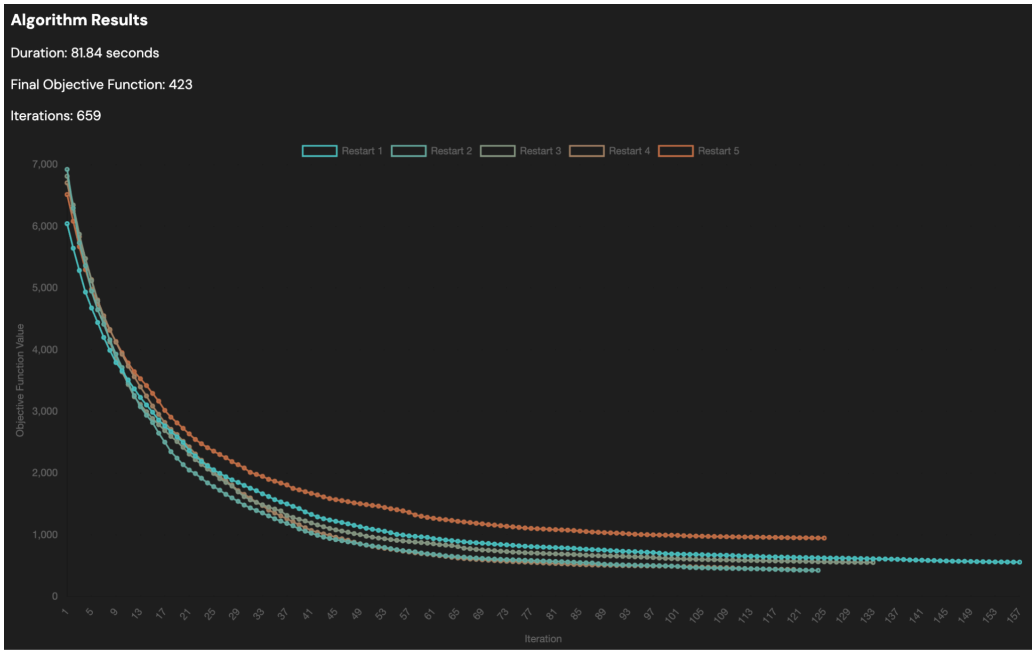
```
bestSolution: Array(5)
▼ 0: Array(5)
  ▶ 0: (5) [116, 47, 81, 16, 56]
  ▶ 1: (5) [17, 50, 121, 114, 14]
  ▶ 2: (5) [88, 21, 34, 37, 115]
  ▶ 3: (5) [69, 109, 48, 24, 66]
  ▶ 4: (5) [25, 82, 22, 122, 64]
    length: 5
  ▶ [[Prototype]]: Array(0)
▼ 1: Array(5)
  ▶ 0: (5) [35, 15, 30, 117, 118]
  ▶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

Hasil:


Algorithm Results
Duration: 56.36 seconds
Final Objective Function: 890
Iterations: 176
Objective Function per Iteration
Objective Function Value
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
1
5
9
13
17
21
25
29
33
37
41
45
49
53
57
61
65
69
73
77
81
85
89
93
97
101
105
109
113
117
121
125
129
133
137
141
145
149
153
157
161
165
169
173

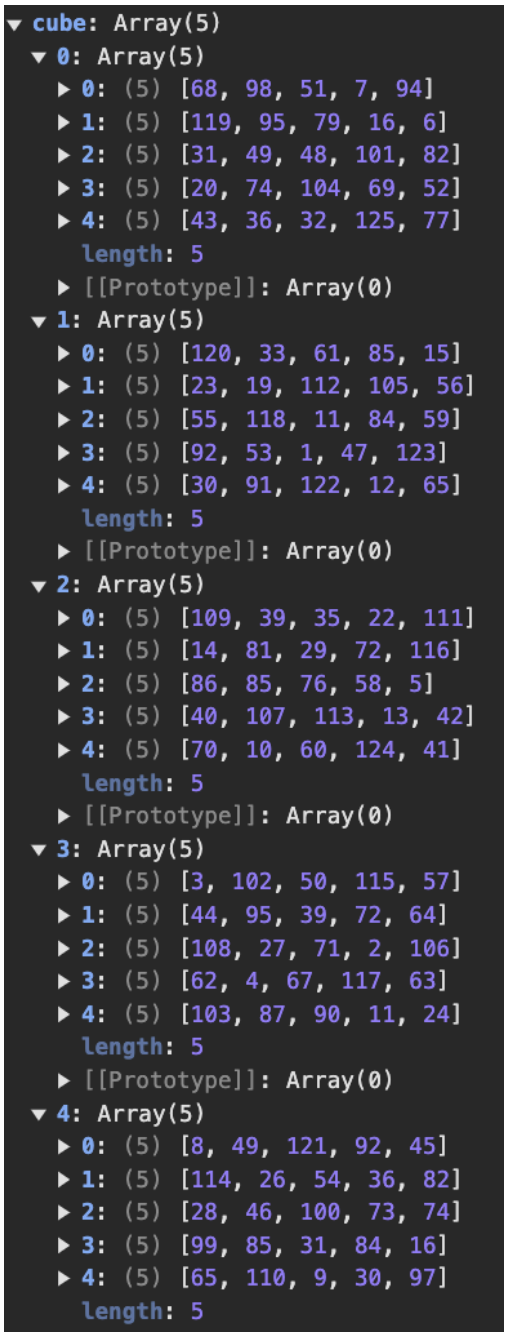
3. Random Restart Hill Climbing (Nb of Restart = 5)

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

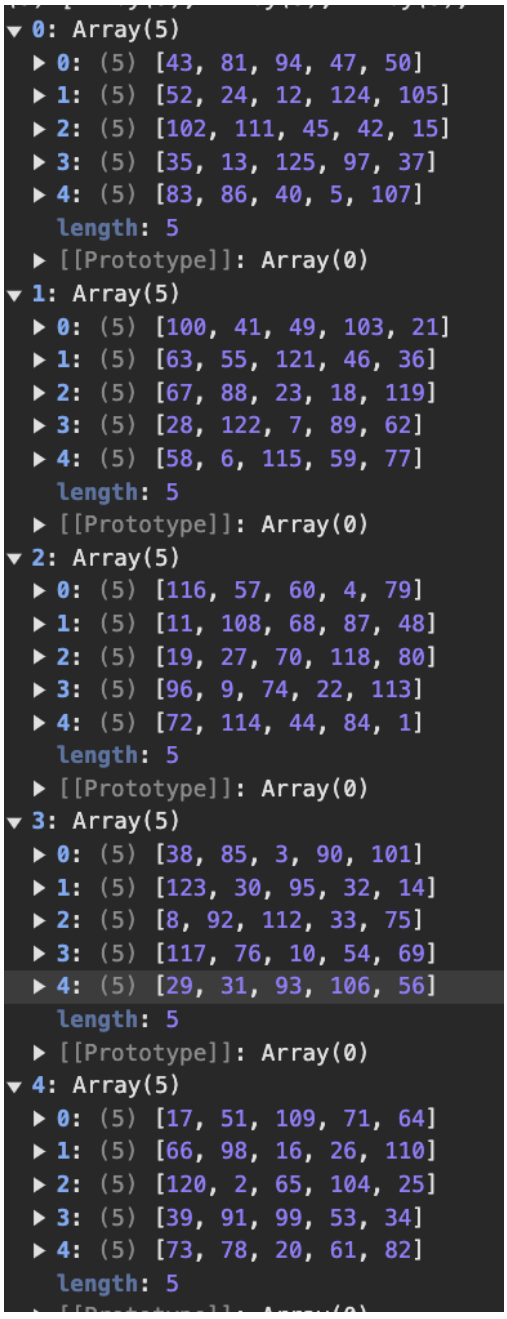

Algorithm Results
Duration: 81.84 seconds
Final Objective Function: 423
Iterations: 659
Restart 1
Restart 2
Restart 3
Restart 4
Restart 5
Objective Function Value
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
Iteration

Percobaan 2

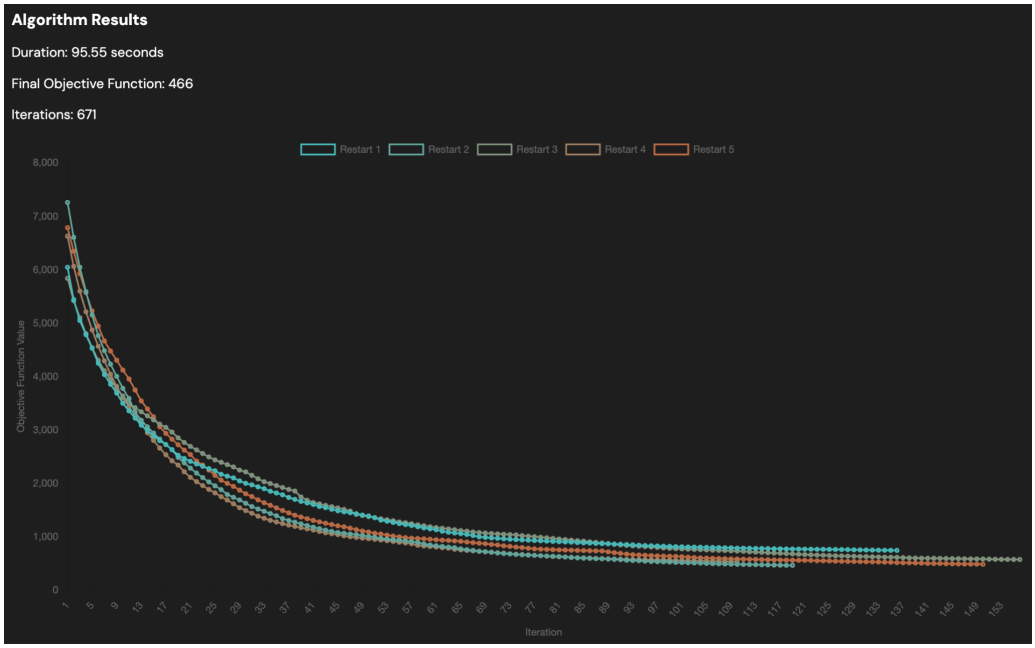
State Awal:



State Akhir:

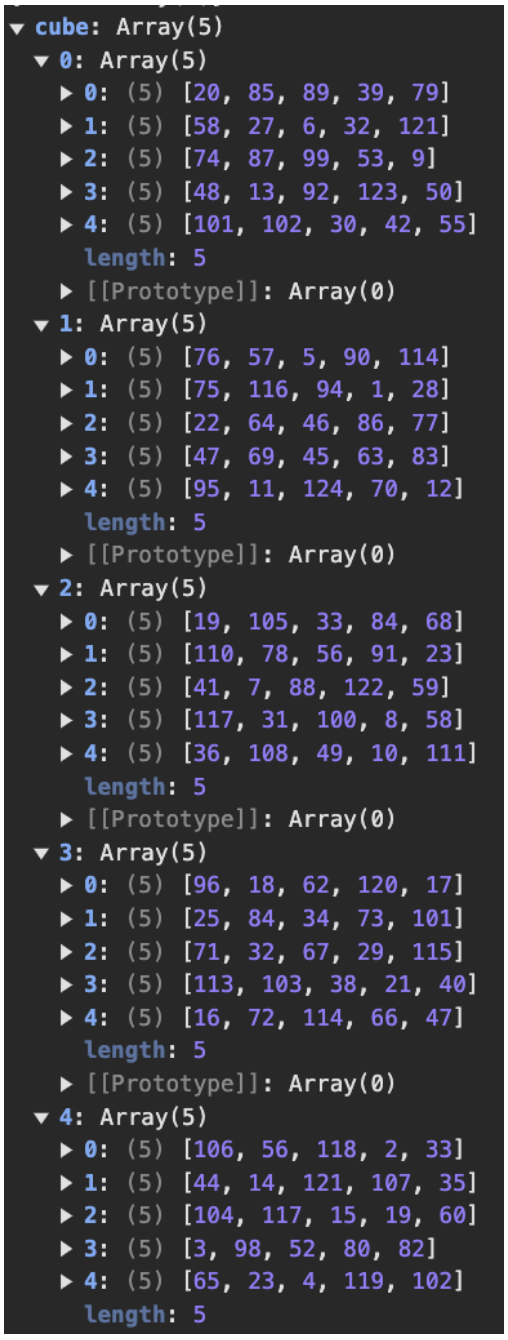


Hasil:

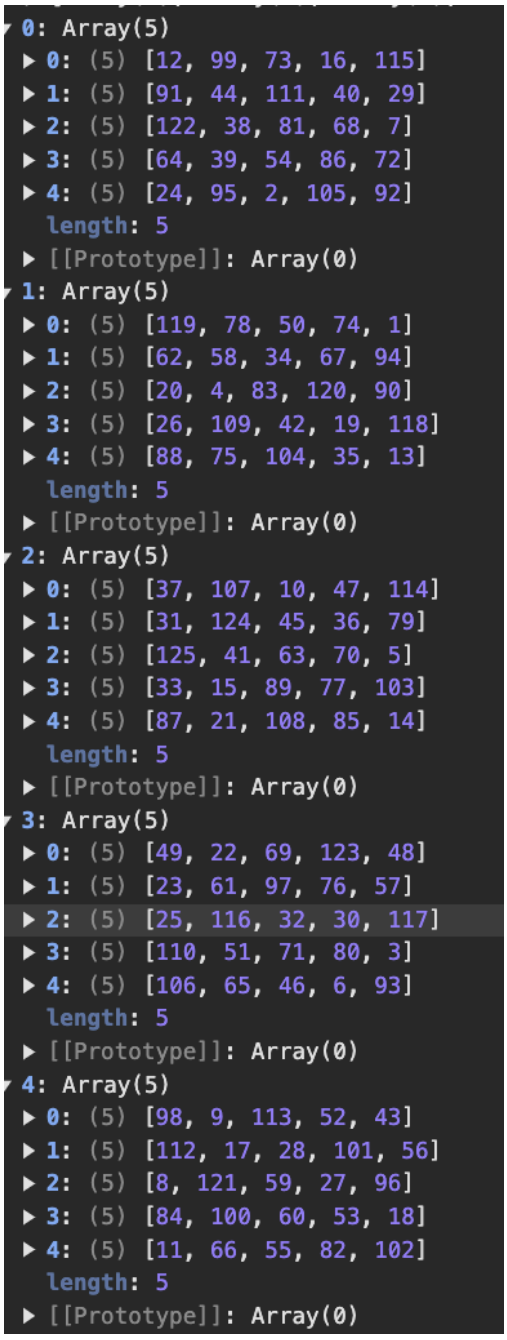

Algorithm Results
Duration: 95.55 seconds
Final Objective Function: 466
Iterations: 671
Restart 1
Restart 2
Restart 3
Restart 4
Restart 5
Objective Function Value
8,000
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
1
5
9
13
17
21
25
29
33
37
41
45
49
53
57
61
65
69
73
77
81
85
89
93
97
101
105
109
113
117
121
125
129
133
137
141
145
149
153
Iteration

Percobaan 3

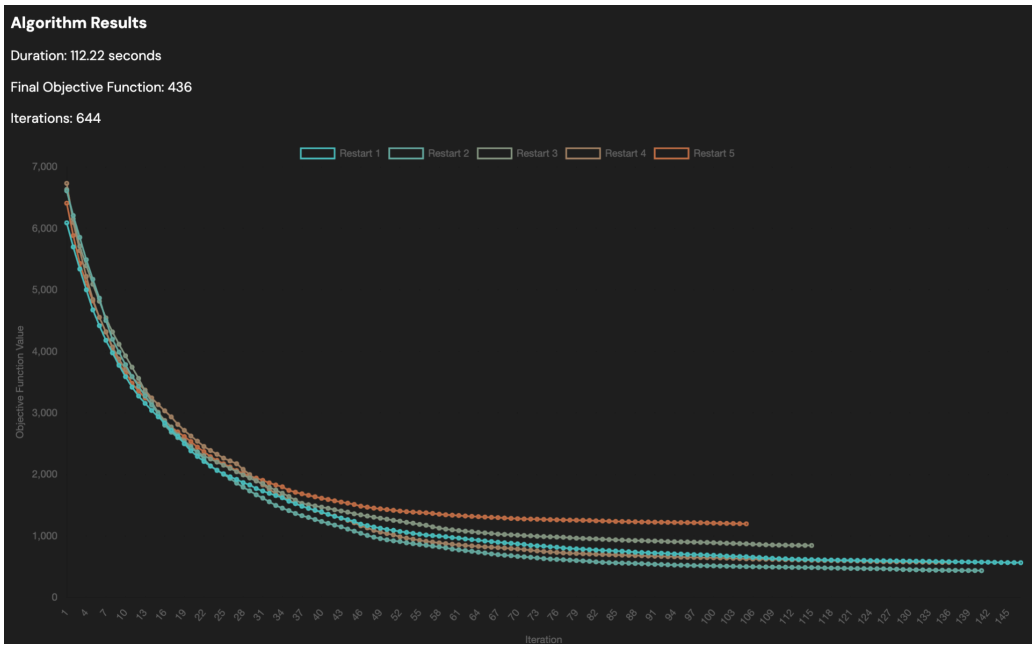
State Awal:



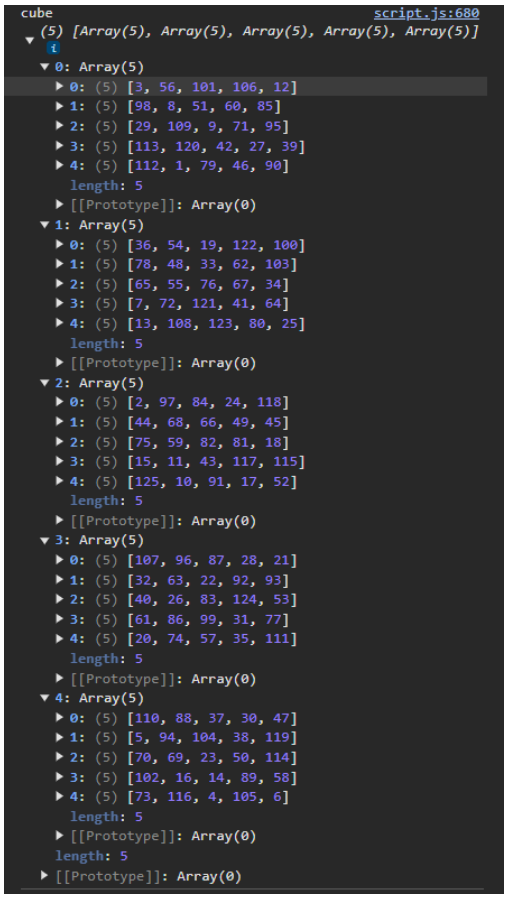
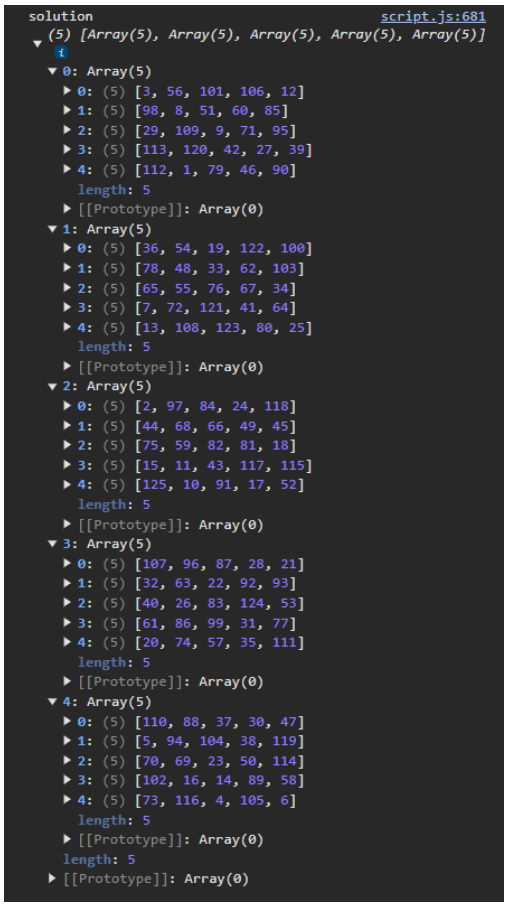
State Akhir:

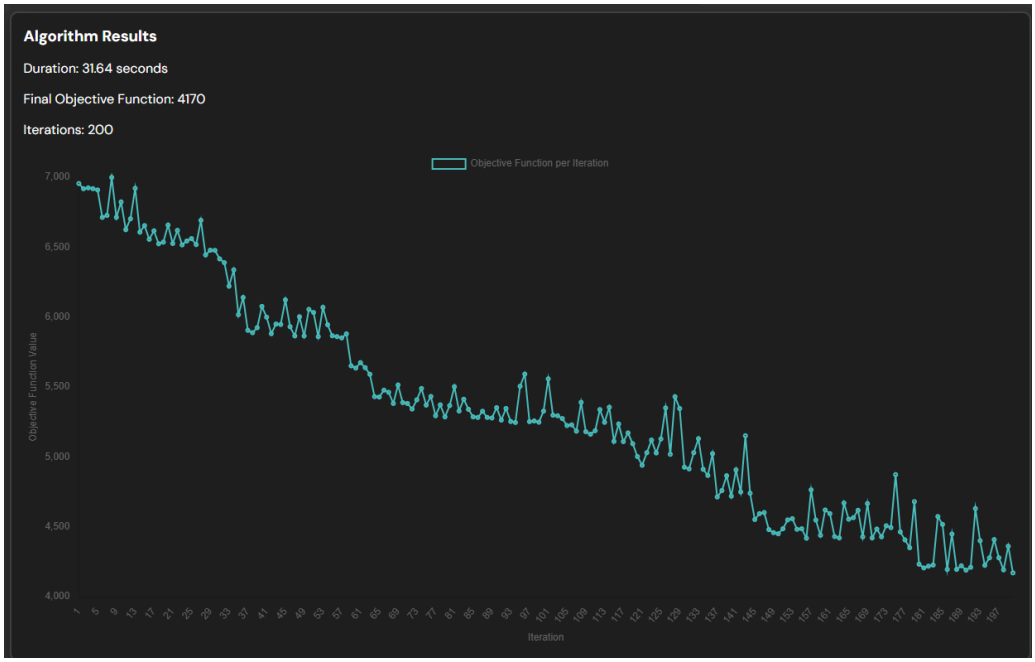


Hasil:


Algorithm Results
Duration: 112.22 seconds
Final Objective Function: 436
Iterations: 644
Restart 1
Restart 2
Restart 3
Restart 4
Restart 5
Objective Function Value
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
Iteration

4. Stochastic Hill Climbing

Percobaan 1	
State Awal: 	State Akhir: 
Hasil:	



Percobaan 2

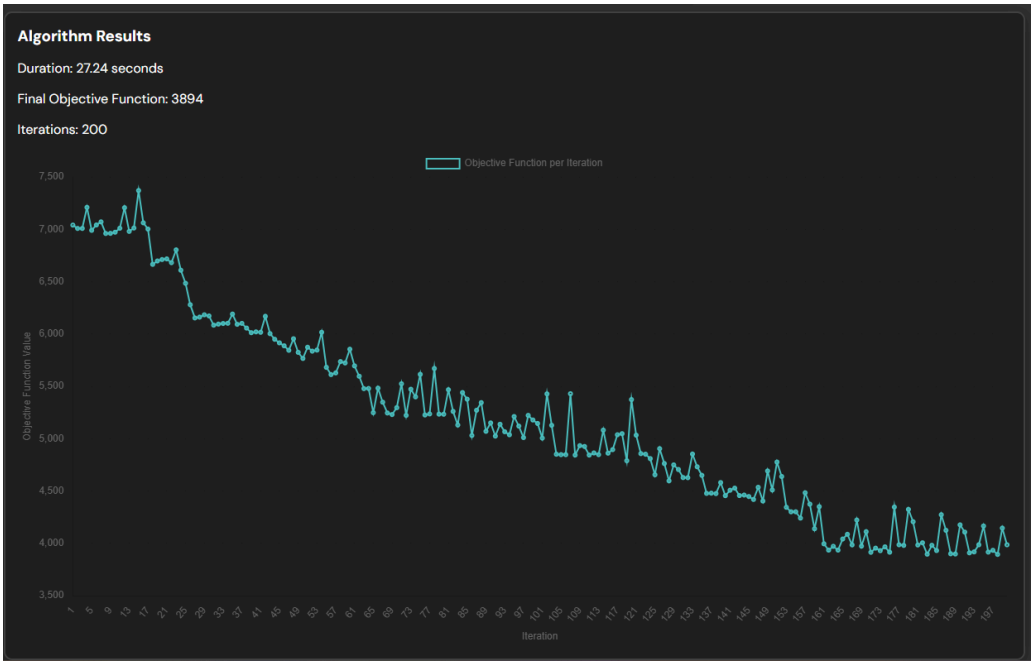
State Awal:

```
cube                                                    script.js:680
▼ (5) [Array(5), Array(5), Array(5), Array(5), Array(5)]
  ▼ 0: Array(5)
    ▶ 0: (5) [63, 5, 88, 122, 40]
    ▶ 1: (5) [73, 107, 8, 20, 115]
    ▶ 2: (5) [4, 31, 81, 113, 101]
    ▶ 3: (5) [21, 97, 70, 116, 17]
    ▶ 4: (5) [74, 28, 30, 85, 39]
      length: 5
    ▶ [[Prototype]]: Array(0)
  ▼ 1: Array(5)
    ▶ 0: (5) [58, 120, 49, 60, 111]
    ▶ 1: (5) [34, 37, 22, 83, 51]
    ▶ 2: (5) [98, 86, 9, 1, 72]
    ▶ 3: (5) [64, 52, 121, 106, 11]
    ▶ 4: (5) [87, 32, 56, 3, 75]
      length: 5
    ▶ [[Prototype]]: Array(0)
  ▼ 2: Array(5)
    ▶ 0: (5) [100, 53, 54, 61, 50]
    ▶ 1: (5) [27, 59, 119, 80, 23]
    ▶ 2: (5) [69, 99, 77, 42, 48]
    ▶ 3: (5) [14, 110, 36, 66, 117]
    ▶ 4: (5) [15, 18, 41, 125, 112]
      length: 5
    ▶ [[Prototype]]: Array(0)
  ▼ 3: Array(5)
    ▶ 0: (5) [89, 45, 104, 96, 68]
    ▶ 1: (5) [43, 55, 108, 12, 93]
    ▶ 2: (5) [38, 47, 76, 67, 29]
    ▶ 3: (5) [118, 57, 2, 62, 124]
    ▶ 4: (5) [44, 90, 103, 79, 10]
      length: 5
    ▶ [[Prototype]]: Array(0)
  ▼ 4: Array(5)
    ▶ 0: (5) [78, 84, 33, 46, 109]
    ▶ 1: (5) [102, 65, 26, 123, 16]
    ▶ 2: (5) [7, 71, 24, 114, 95]
    ▶ 3: (5) [94, 19, 92, 25, 13]
    ▶ 4: (5) [35, 105, 91, 6, 82]
      length: 5
    ▶ [[Prototype]]: Array(0)
    length: 5
  ▶ [[Prototype]]: Array(0)
```

State Akhir:

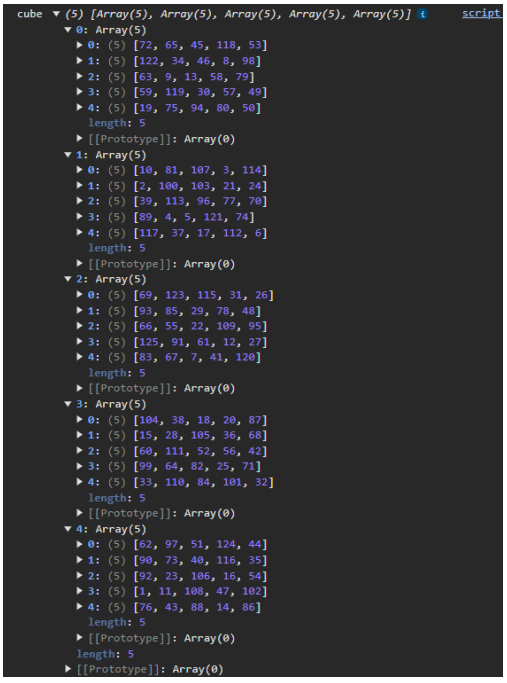
```
solution                                                script.js:681
▼ (5) [Array(5), Array(5), Array(5), Array(5), Array(5)]
  ▼ 0: Array(5)
    ▶ 0: (5) [63, 5, 88, 122, 40]
    ▶ 1: (5) [73, 107, 8, 20, 115]
    ▶ 2: (5) [4, 31, 81, 113, 101]
    ▶ 3: (5) [21, 97, 70, 116, 17]
    ▶ 4: (5) [74, 28, 30, 85, 39]
      length: 5
    ▶ [[Prototype]]: Array(0)
  ▼ 1: Array(5)
    ▶ 0: (5) [58, 120, 49, 60, 111]
    ▶ 1: (5) [34, 37, 22, 83, 51]
    ▶ 2: (5) [98, 86, 9, 1, 72]
    ▶ 3: (5) [64, 52, 121, 106, 11]
    ▶ 4: (5) [87, 32, 56, 3, 75]
      length: 5
    ▶ [[Prototype]]: Array(0)
  ▼ 2: Array(5)
    ▶ 0: (5) [100, 53, 54, 61, 50]
    ▶ 1: (5) [27, 59, 119, 80, 23]
    ▶ 2: (5) [69, 99, 77, 42, 48]
    ▶ 3: (5) [14, 110, 36, 66, 117]
    ▶ 4: (5) [15, 18, 41, 125, 112]
      length: 5
    ▶ [[Prototype]]: Array(0)
  ▼ 3: Array(5)
    ▶ 0: (5) [89, 45, 104, 96, 68]
    ▶ 1: (5) [43, 55, 108, 12, 93]
    ▶ 2: (5) [38, 47, 76, 67, 29]
    ▶ 3: (5) [118, 57, 2, 62, 124]
    ▶ 4: (5) [44, 90, 103, 79, 10]
      length: 5
    ▶ [[Prototype]]: Array(0)
  ▼ 4: Array(5)
    ▶ 0: (5) [78, 84, 33, 46, 109]
    ▶ 1: (5) [102, 65, 26, 123, 16]
    ▶ 2: (5) [7, 71, 24, 114, 95]
    ▶ 3: (5) [94, 19, 92, 25, 13]
    ▶ 4: (5) [35, 105, 91, 6, 82]
      length: 5
    ▶ [[Prototype]]: Array(0)
    length: 5
  ▶ [[Prototype]]: Array(0)
```

Hasil:

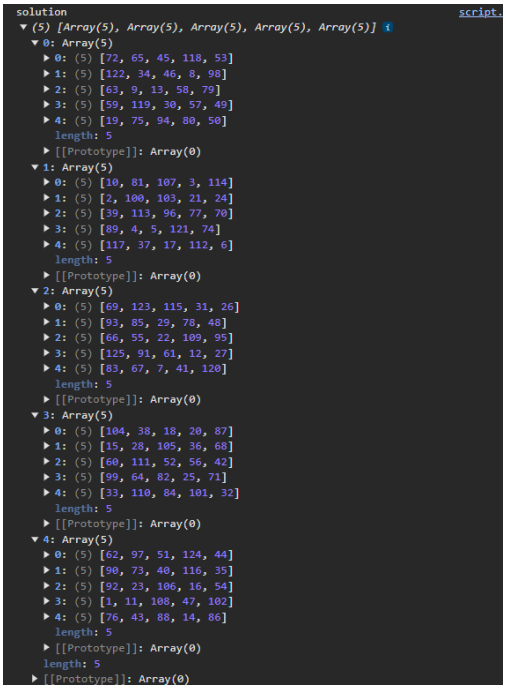


Percobaan 3

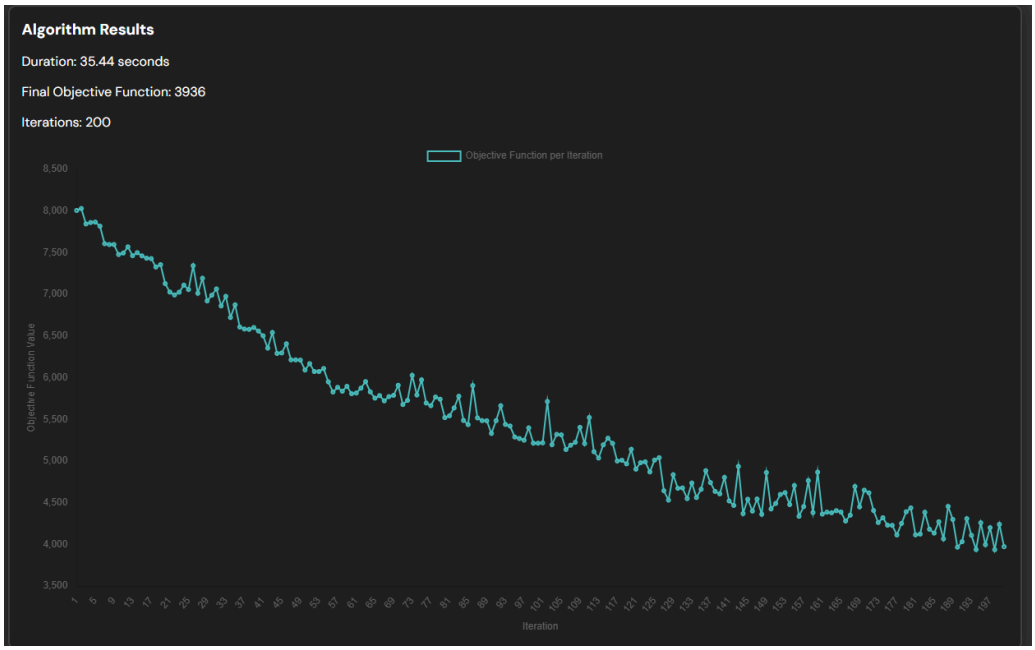
State Awal:



State Akhir:



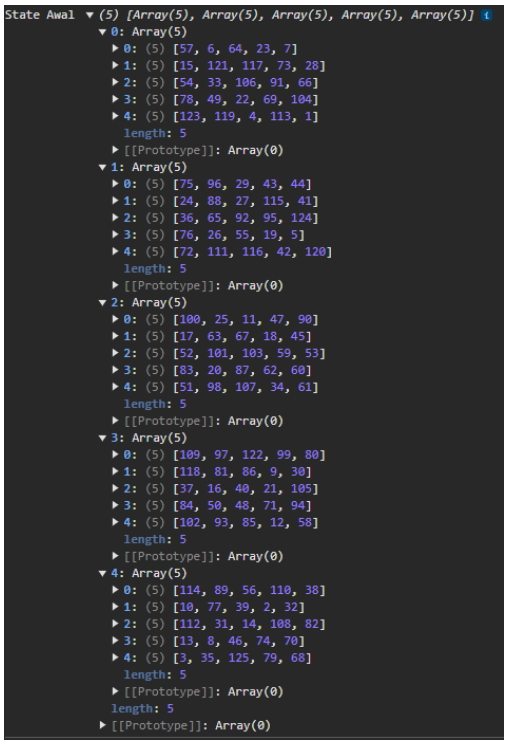
Hasil:



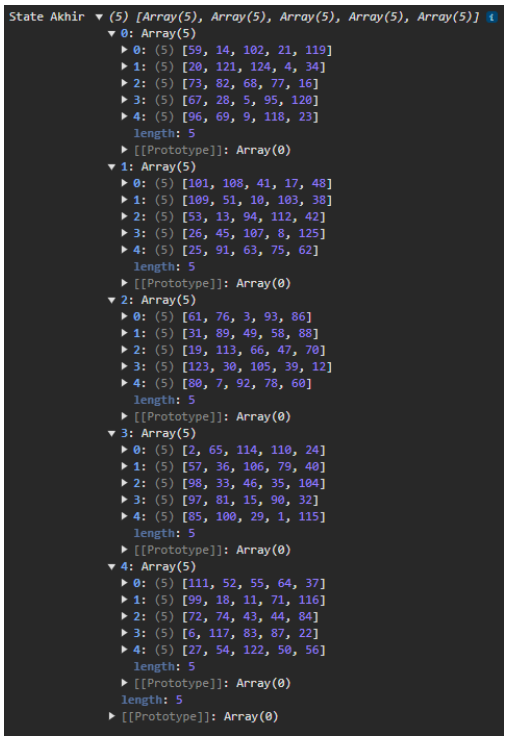
5. Simulated Annealing

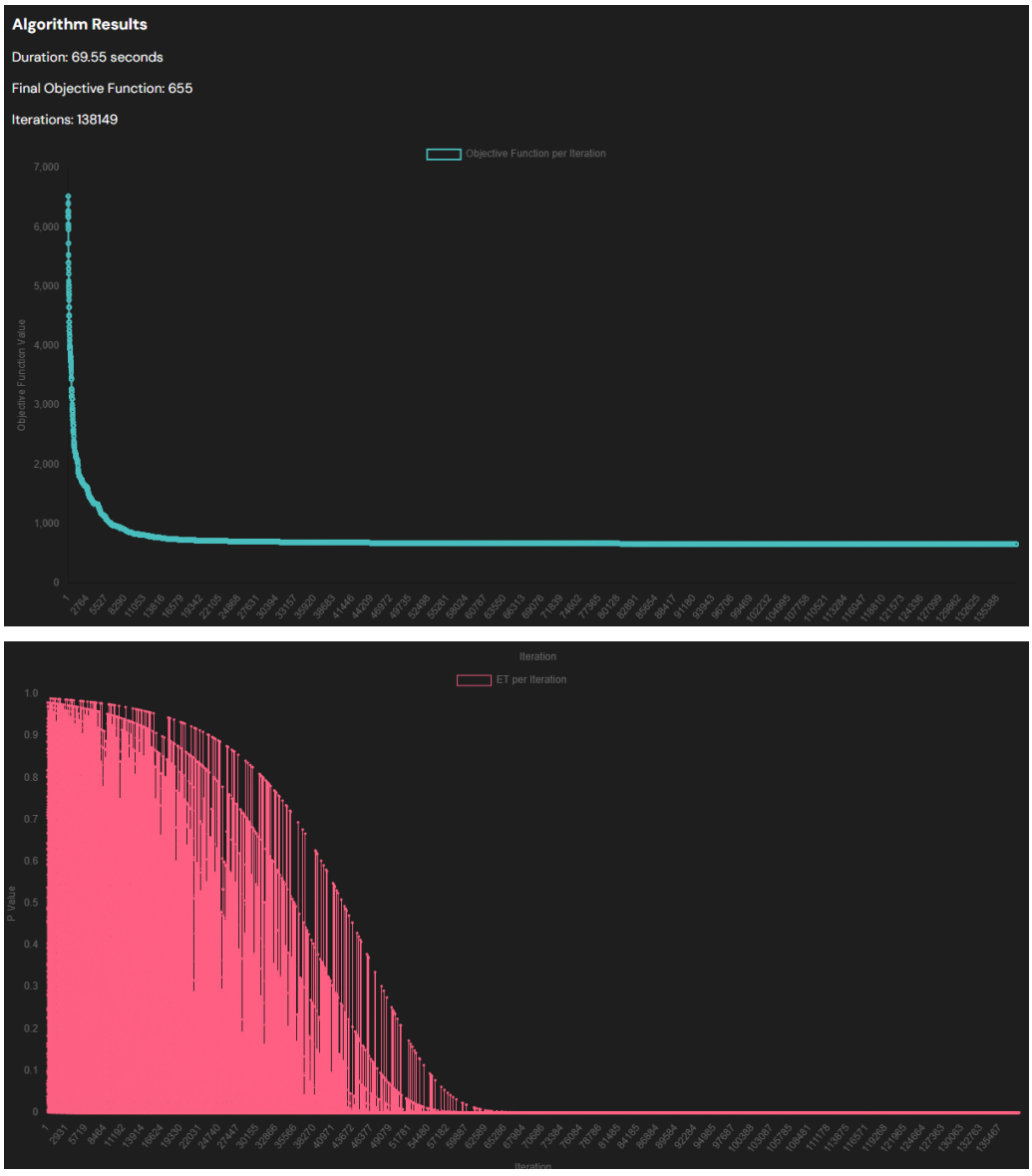
Percobaan 1

State Awal:

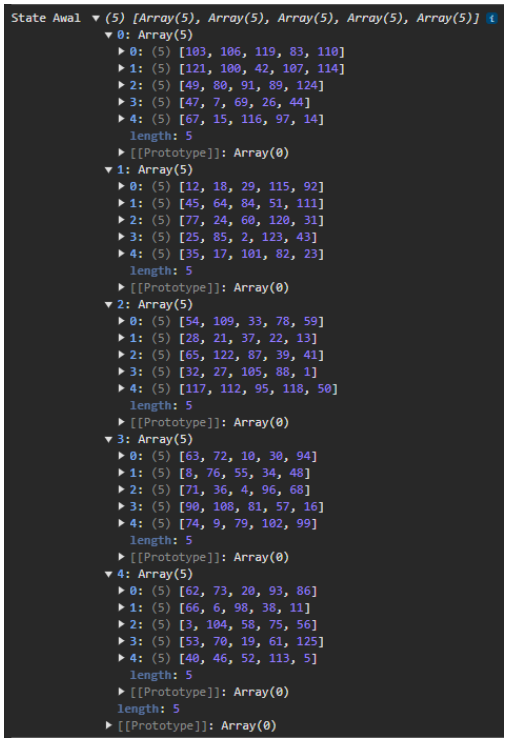


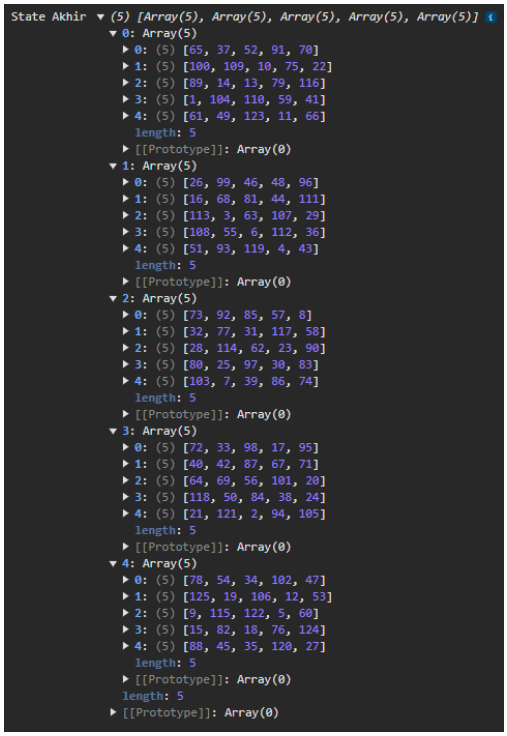
State Akhir:



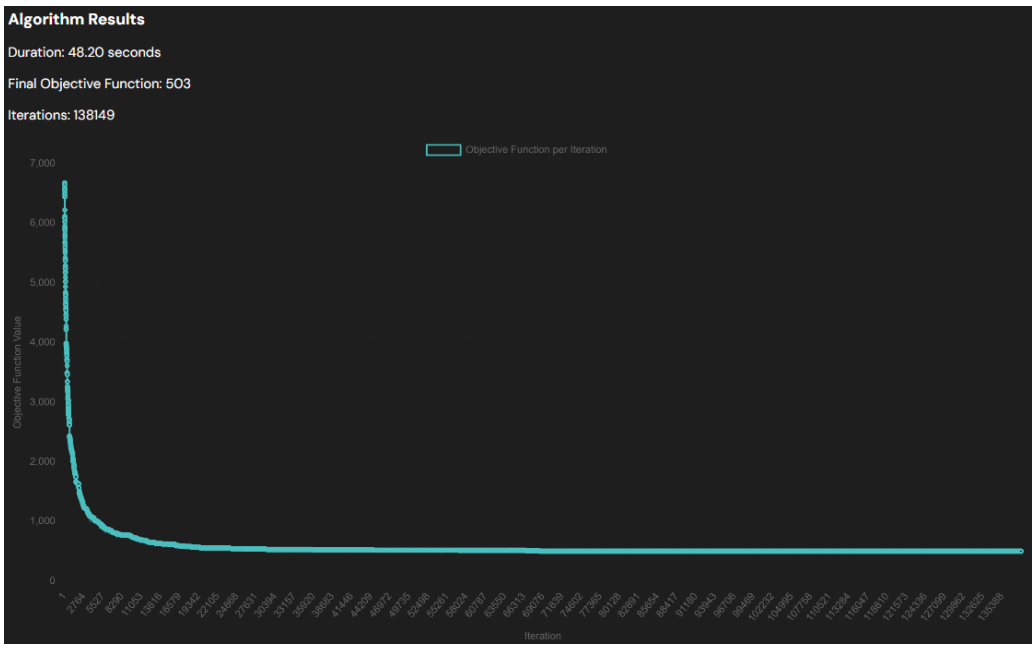
Hasil:  	

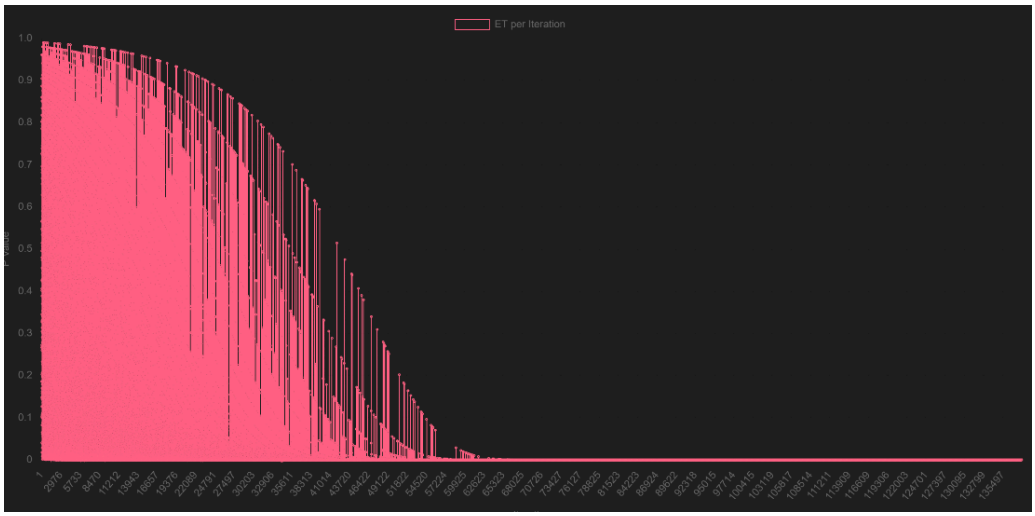
Percobaan 2	
State Awal:	State Akhir:


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


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

Hasil:



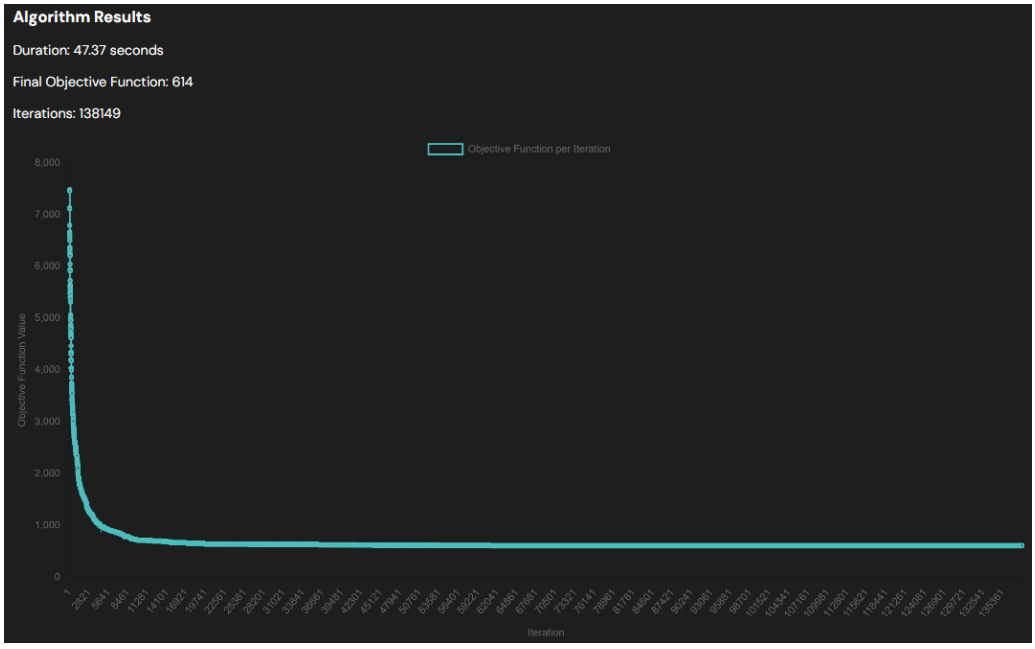


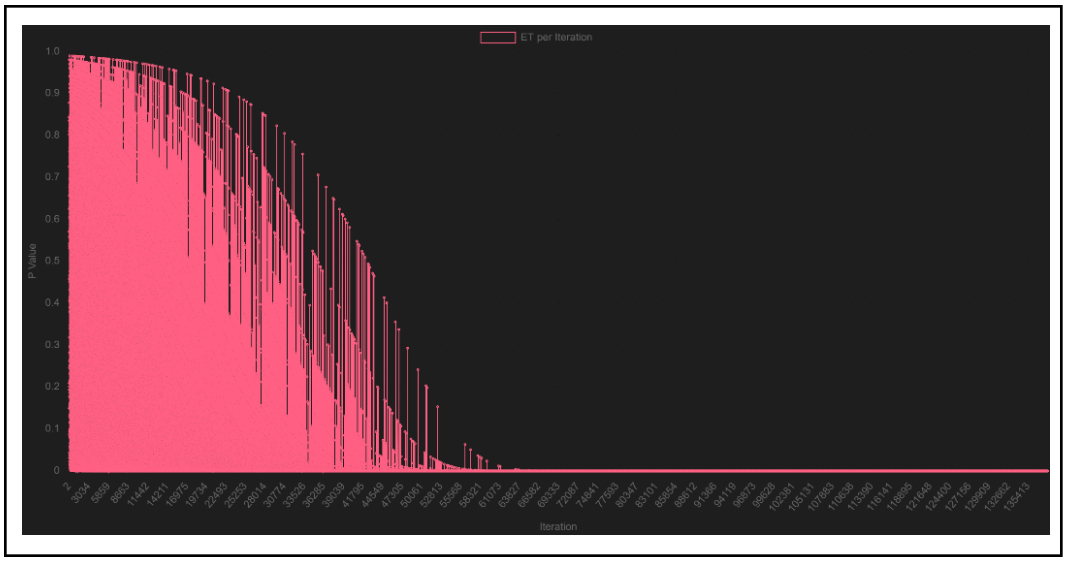
Percobaan 3	
State Awal:	State Akhir:

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

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


Hasil:

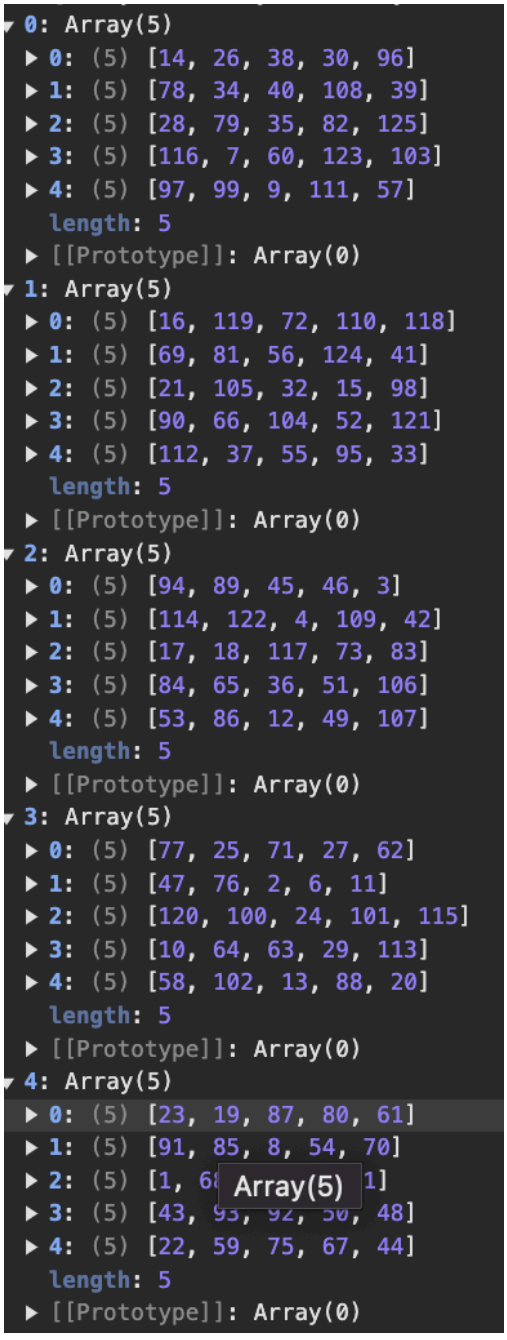



ET per Iteration
P Value
Iteration
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2
3034
5859
8683
11442
14211
16975
19734
22493
25253
28014
30774
33526
36285
39039
41796
44549
47305
50061
52813
55568
58321
61073
63827
66582
69333
72087
74841
77593
80347
83101
85854
88612
91366
94119
96873
99628
102381
105131
107883
110636
113390
116141
118895
121648
124400
127156
129909
132662
135413

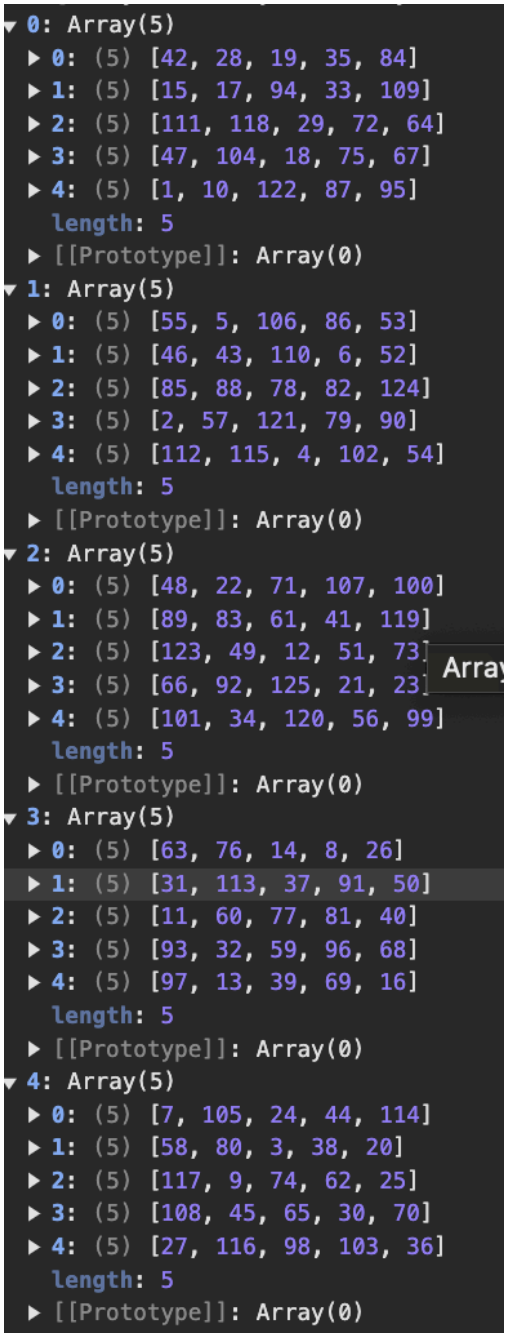
6. Genetic Algorithm
    a. Variabel Kontrol (Jumlah Iterasi = 1000)
        i. Jumlah populasi = 10

Percobaan 1

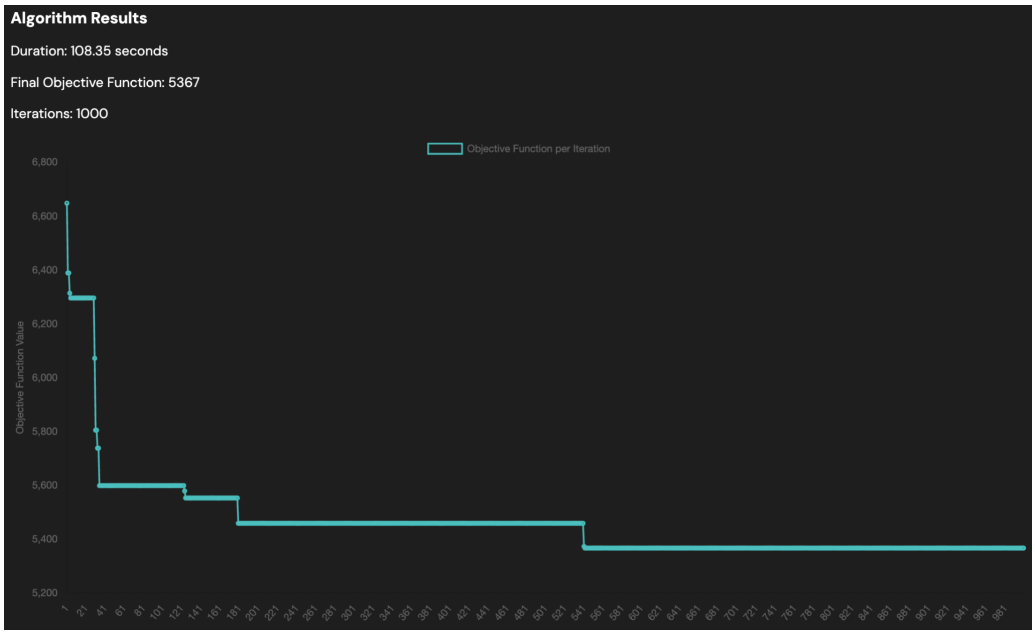
State Awal:



State Akhir:

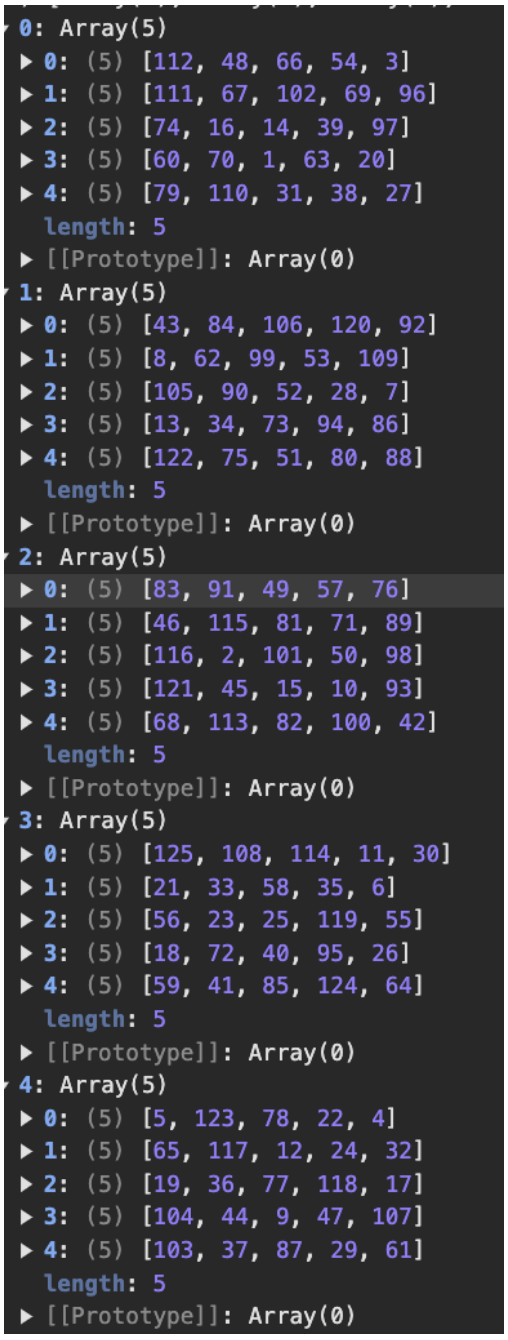


Hasil:

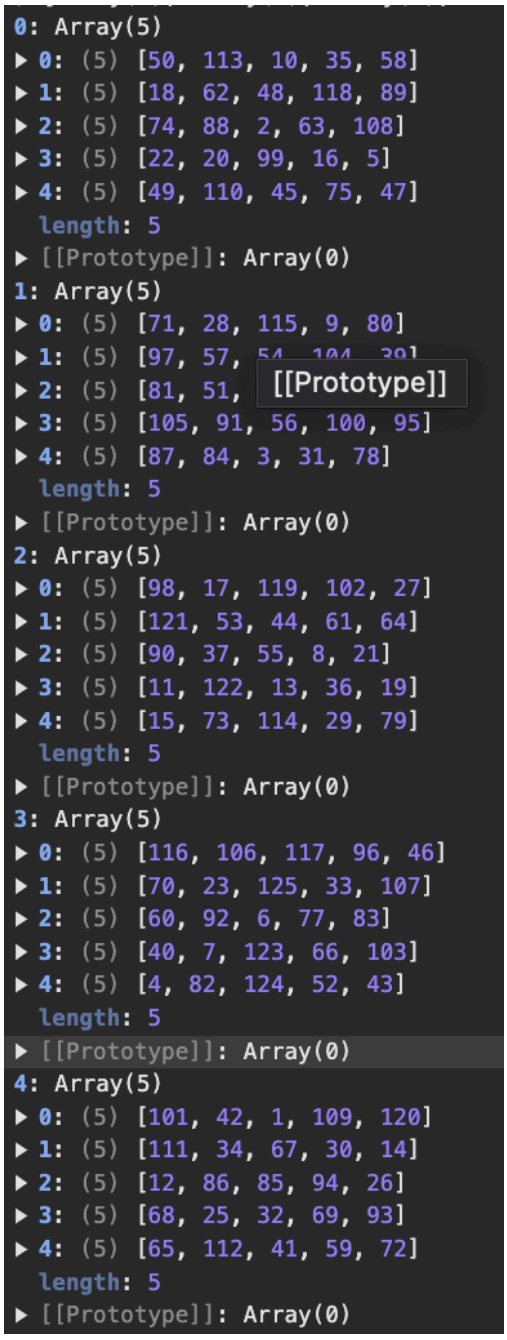

Algorithm Results
Duration: 108.35 seconds
Final Objective Function: 5367
Iterations: 1000
Objective Function per Iteration
Objective Function Value
6,800
6,600
6,400
6,200
6,000
5,800
5,600
5,400
5,200
1
21
41
61
81
101
121
141
161
181
201
221
241
261
281
301
321
341
361
381
401
421
441
461
481
501
521
541
561
581
601
621
641
661
681
701
721
741
761
781
801
821
841
861
881
901
921
941
961
981

Percobaan 2

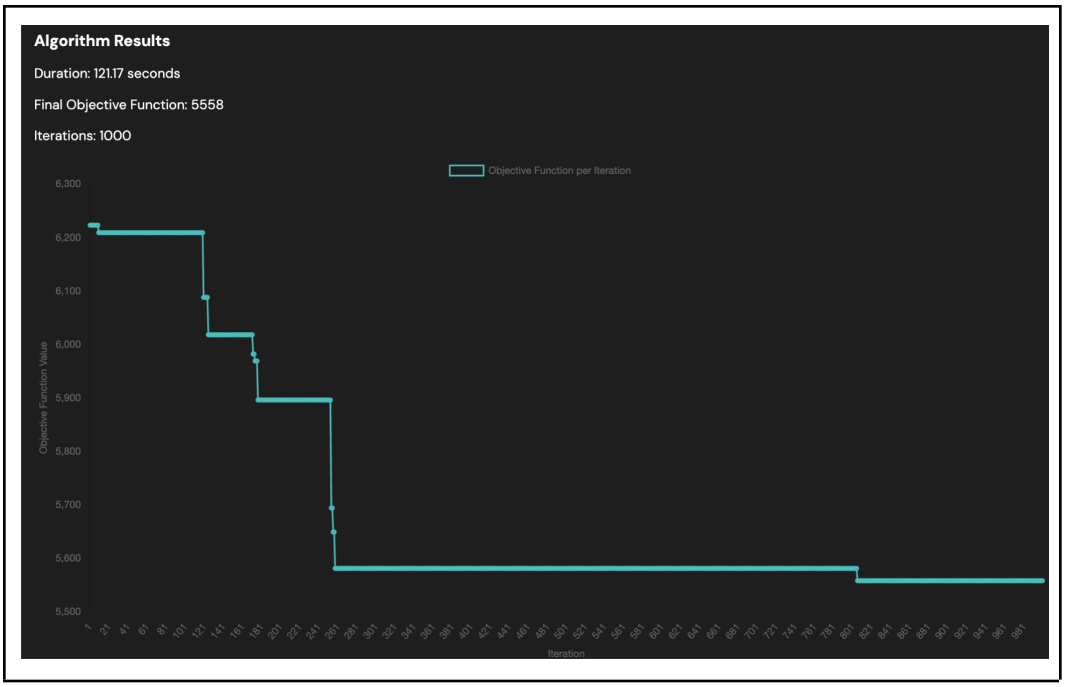
State Awal:



State Akhir:

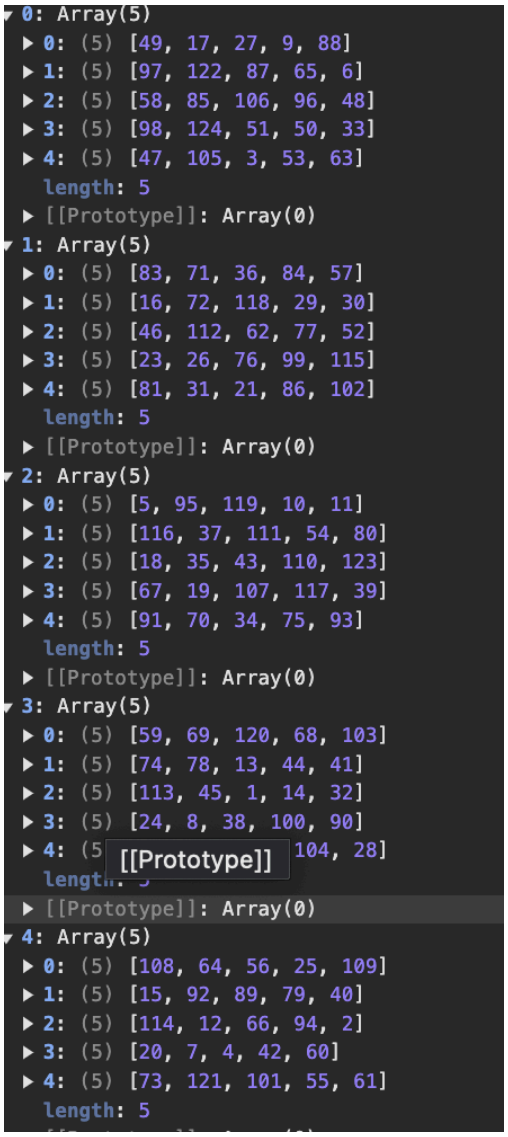


Hasil:

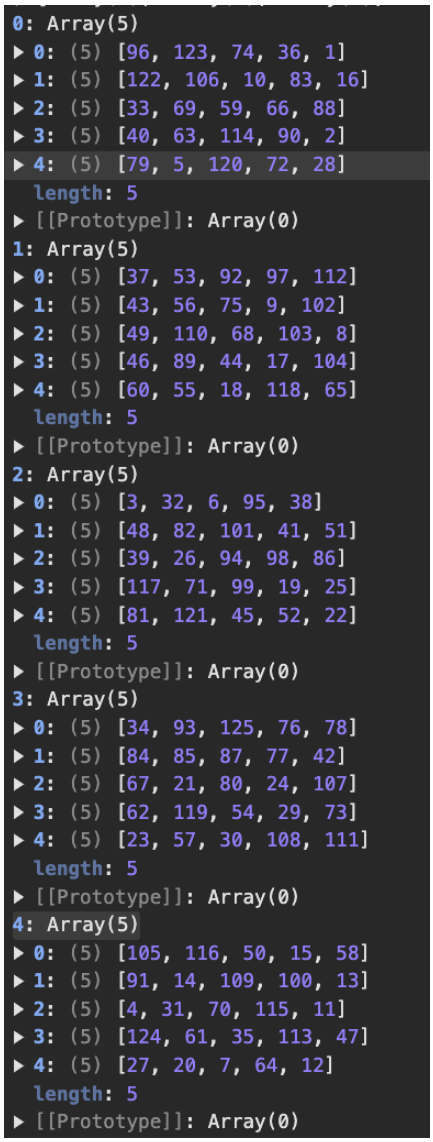

Algorithm Results
Duration: 121.17 seconds
Final Objective Function: 5558
Iterations: 1000
Objective Function per Iteration
Objective Function Value
6,300
6,200
6,100
6,000
5,900
5,800
5,700
5,600
5,500
1
21
41
61
81
101
121
141
161
181
201
221
241
261
281
301
321
341
361
381
401
421
441
461
481
501
521
541
561
581
601
621
641
661
681
701
721
741
761
781
801
821
841
861
881
901
921
941
961
981
Iteration

Percobaan 3

State Awal:



State Akhir:



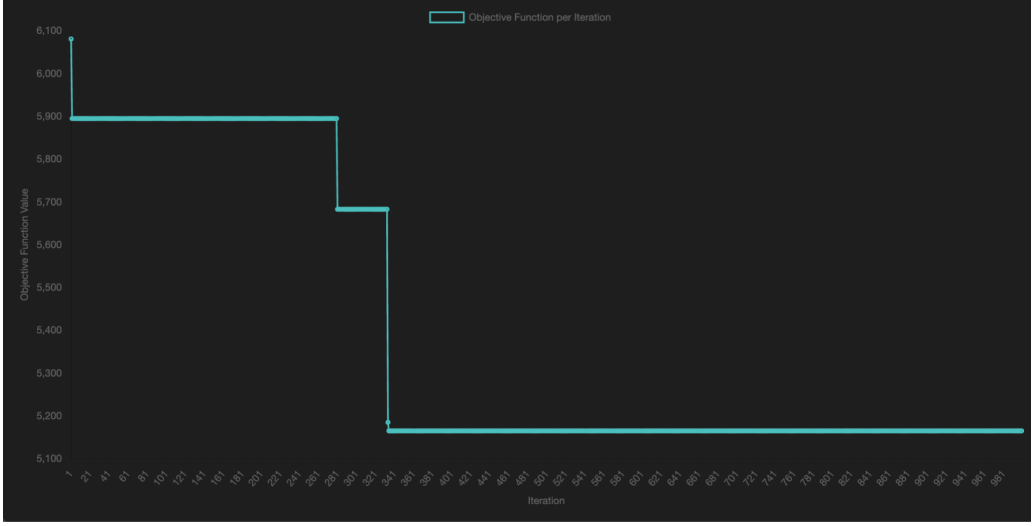
Hasil:

Algorithm Results

Duration: 277.86 seconds

Final Objective Function: 5165

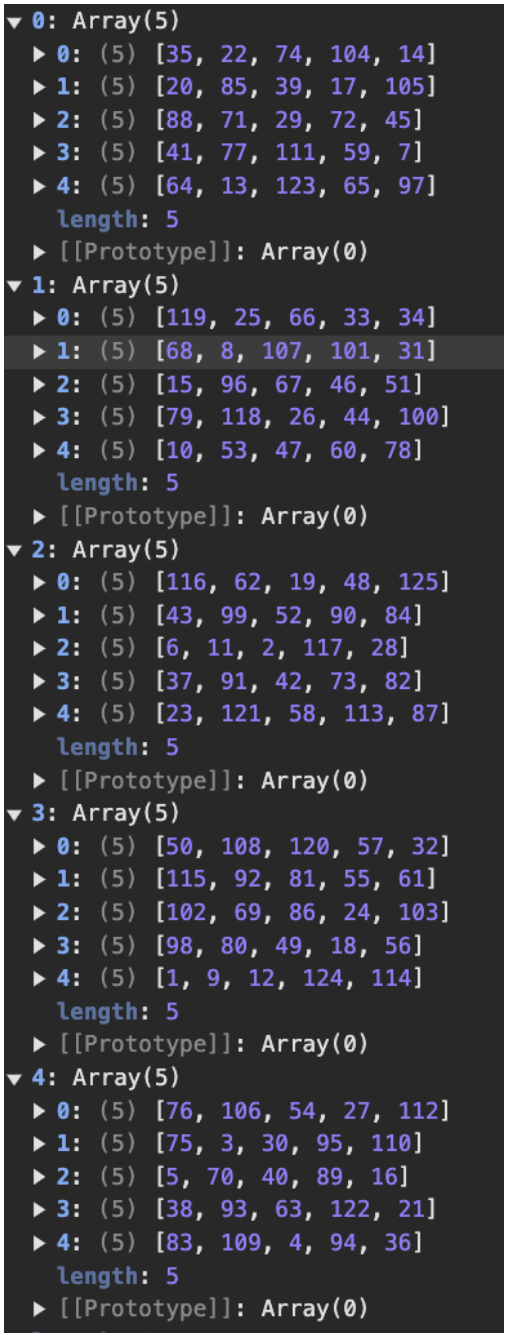
Iterations: 1000



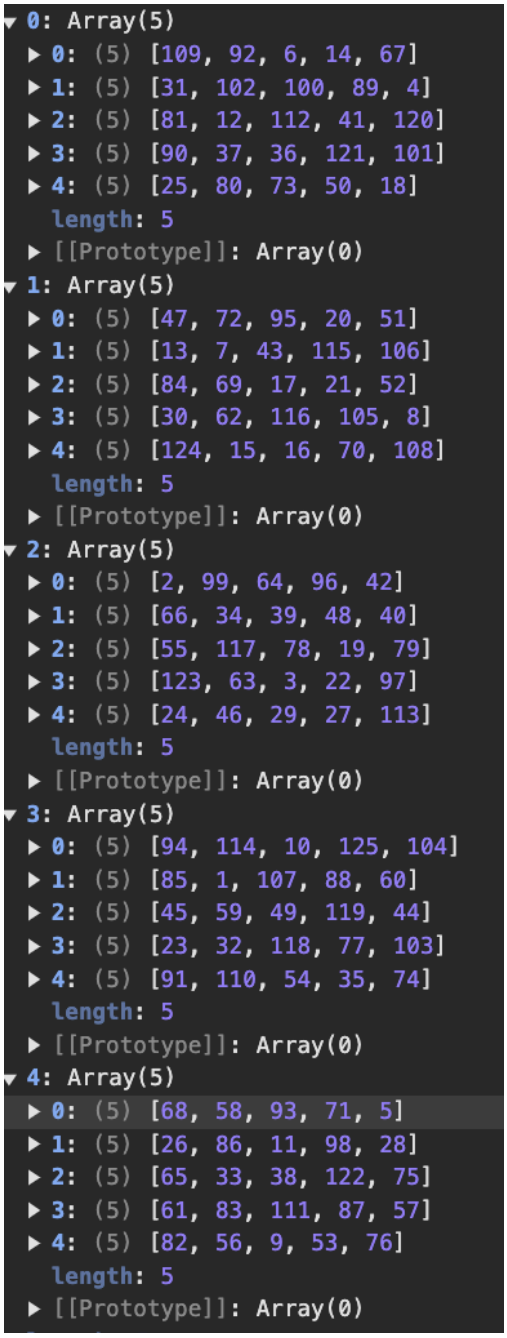
ii. Jumlah populasi = 50

Percobaan 1

State Awal:



State Akhir:



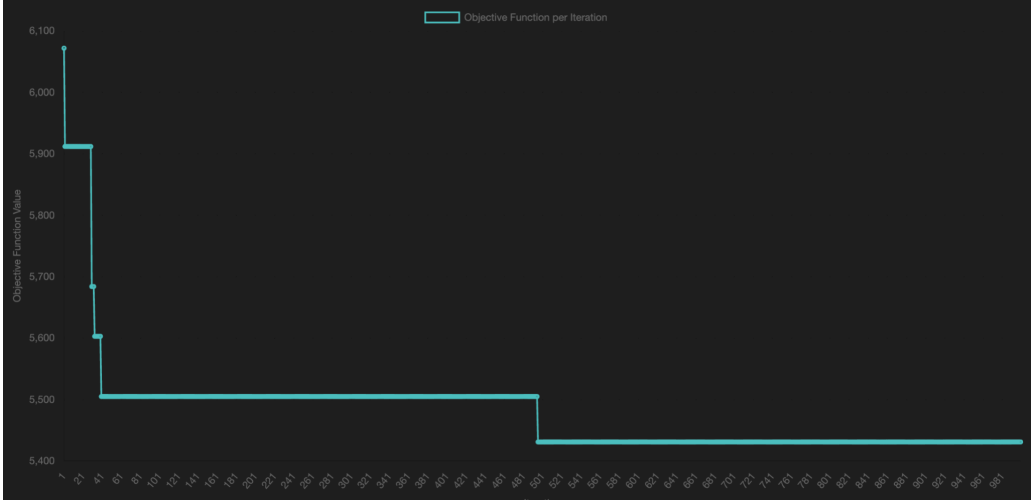
Hasil:

Algorithm Results

Duration: 498.91 seconds

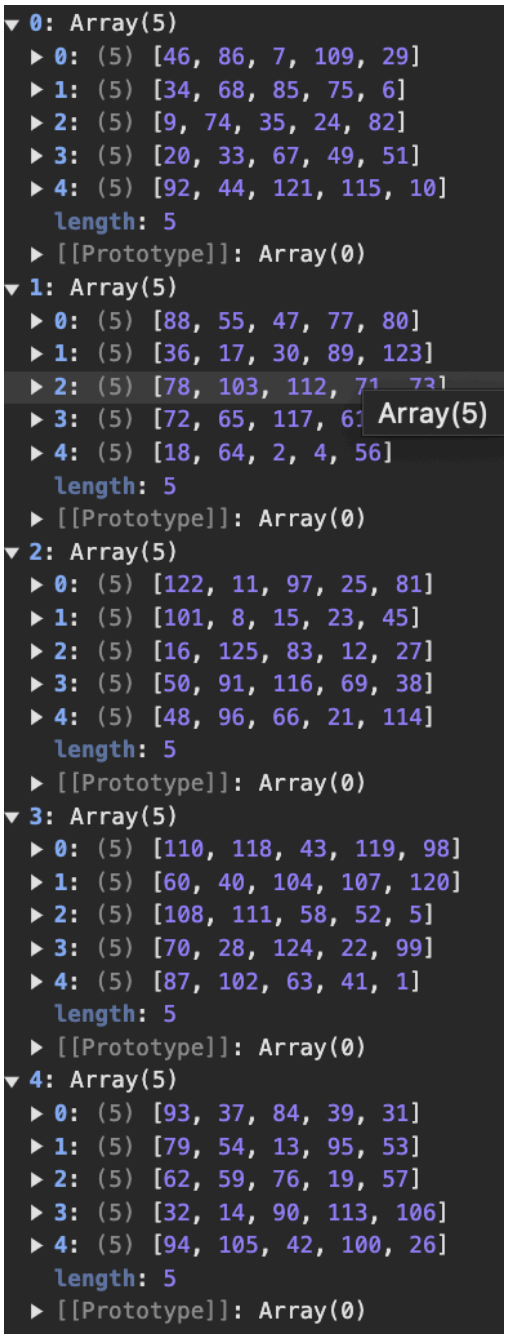
Final Objective Function: 5431

Iterations: 1000

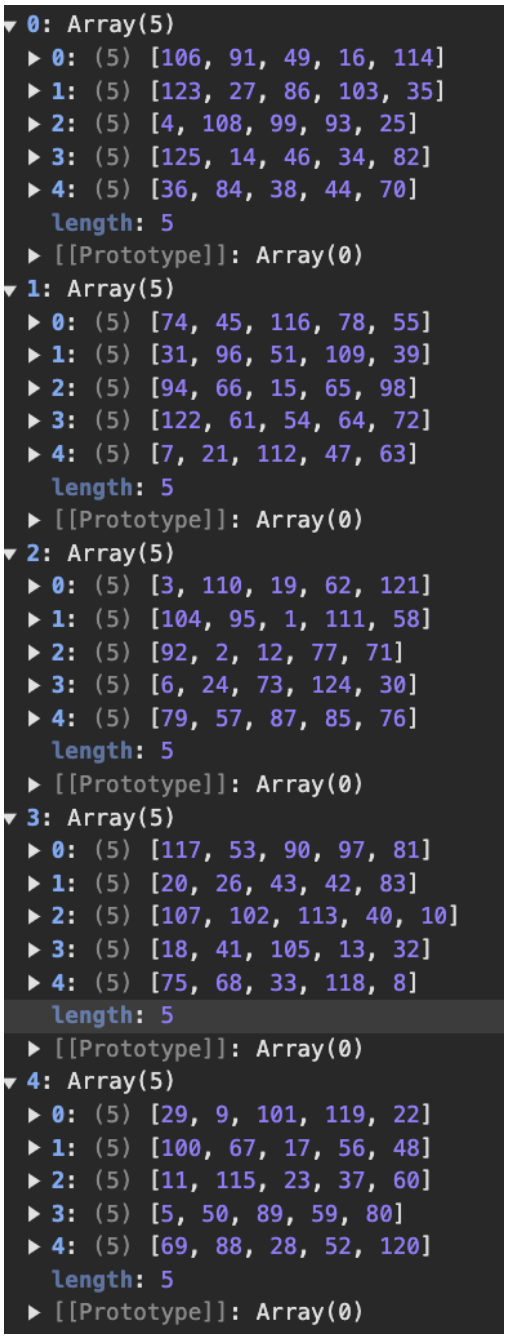


Percobaan 2

State Awal:



State Akhir:



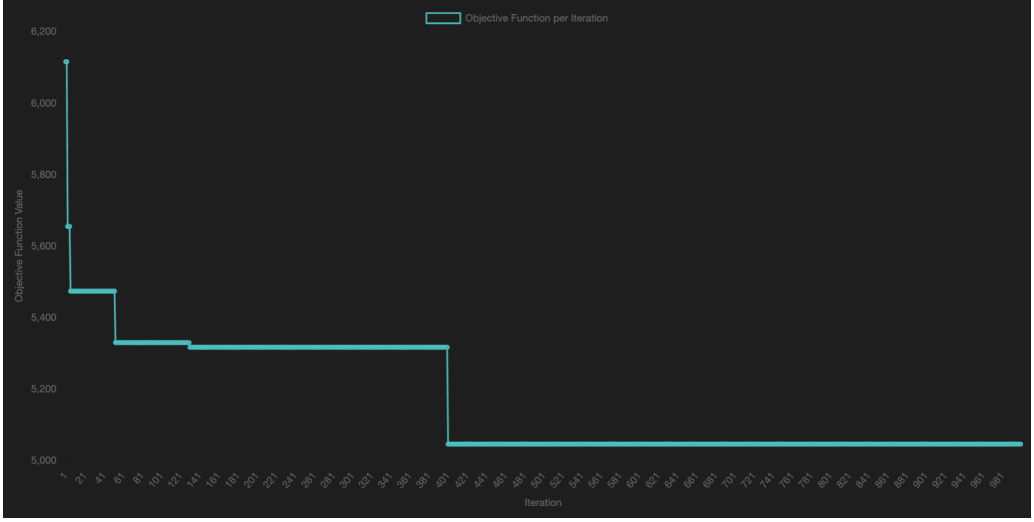
Hasil:

Algorithm Results

Duration: 659.83 seconds

Final Objective Function: 5046

Iterations: 1000



Percobaan 3

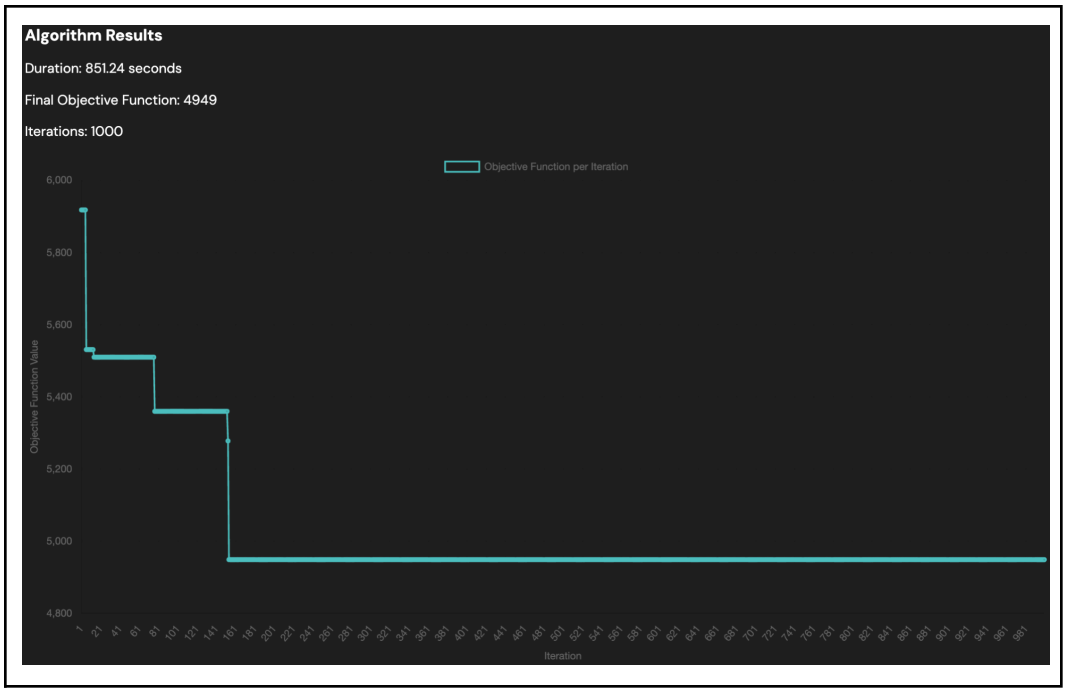
State Awal:

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

State Akhir:

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

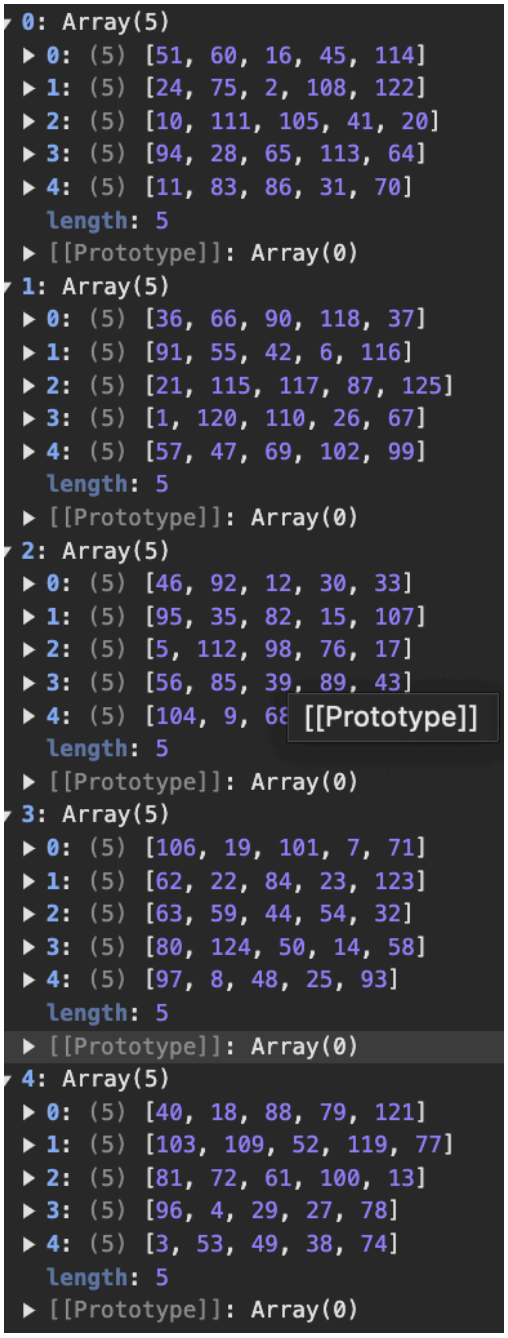
Hasil:


Algorithm Results
Duration: 851.24 seconds
Final Objective Function: 4949
Iterations: 1000
Objective Function per Iteration
Objective Function Value
6,000
5,800
5,600
5,400
5,200
5,000
4,800
Iteration

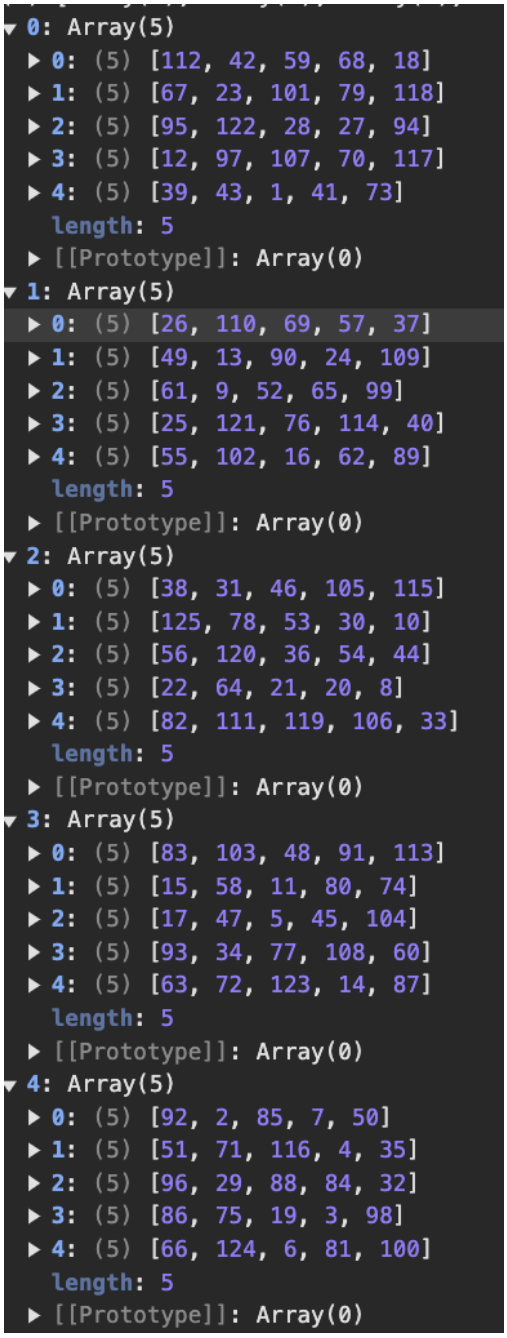
iii. Jumlah populasi = 100

Percobaan 1

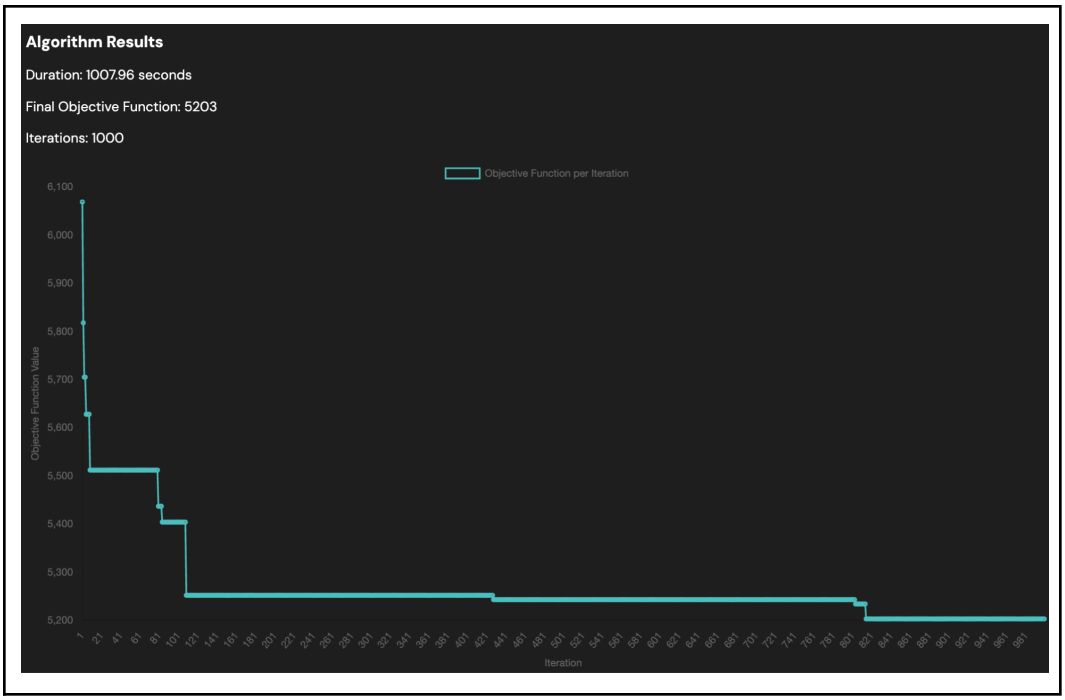
State Awal:



State Akhir:

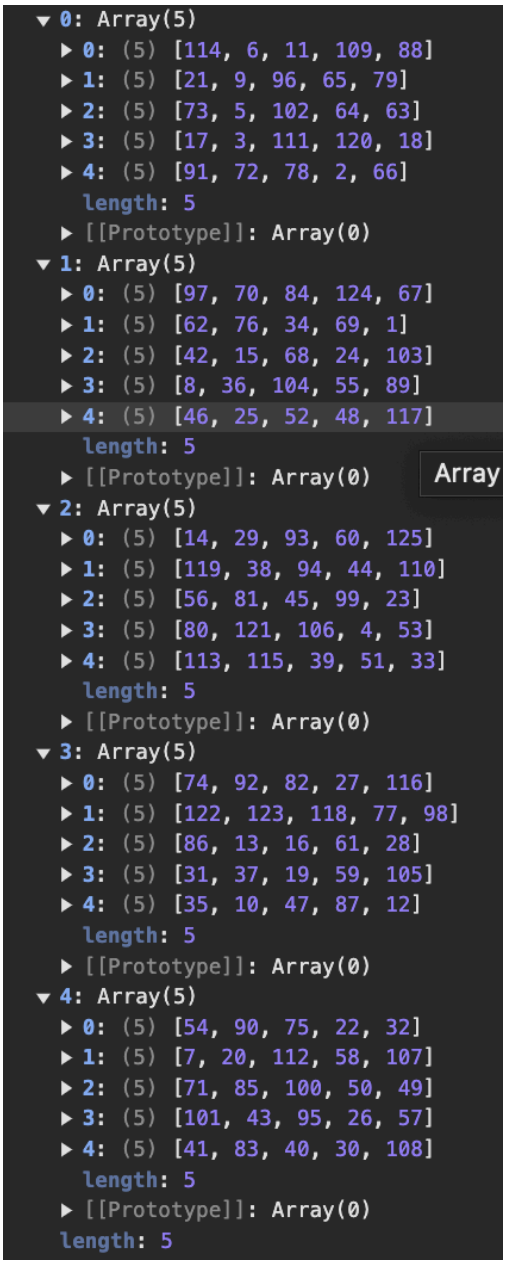


Hasil:

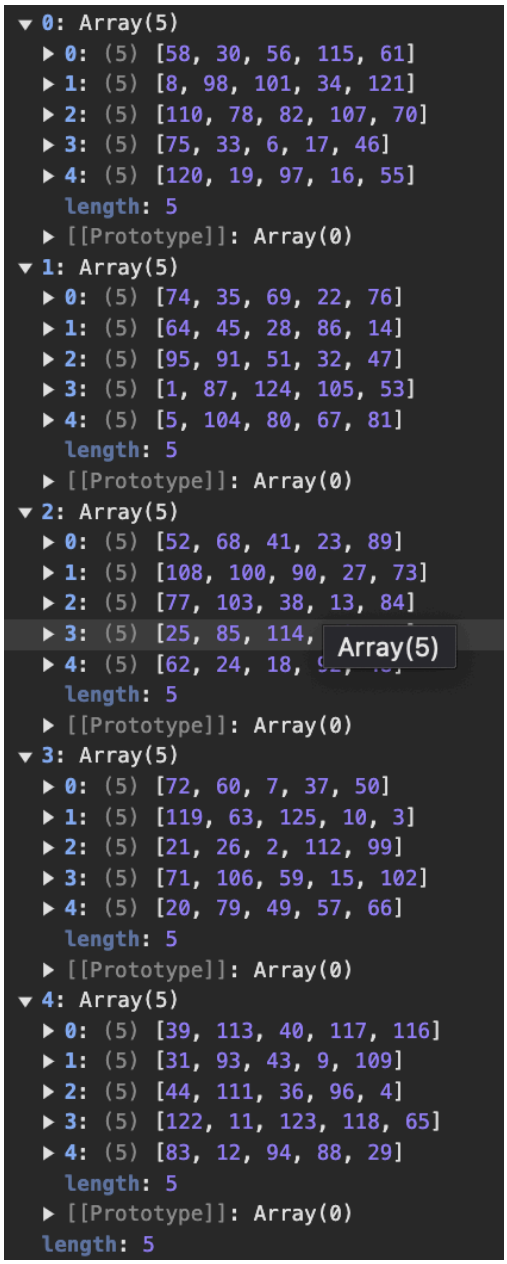

Algorithm Results
Duration: 1007.96 seconds
Final Objective Function: 5203
Iterations: 1000
Objective Function per Iteration
Objective Function Value
6,100
6,000
5,900
5,800
5,700
5,600
5,500
5,400
5,300
5,200
1
21
41
61
81
101
121
141
161
181
201
221
241
261
281
301
321
341
361
381
401
421
441
461
481
501
521
541
561
581
601
621
641
661
681
701
721
741
761
781
801
821
841
861
881
901
921
941
961
981
Iteration

Percobaan 2

State Awal:



State Akhir:



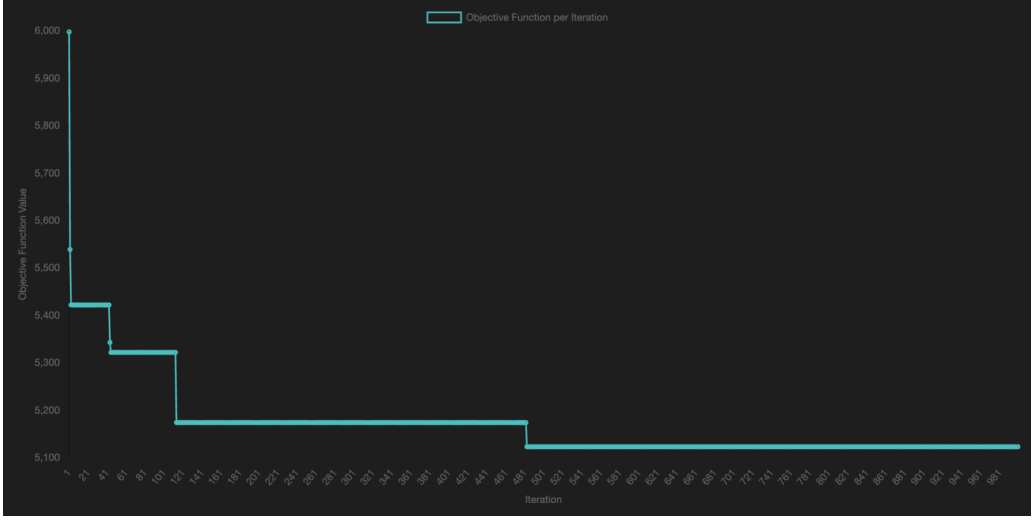
Hasil:

Algorithm Results

Duration: 1217.71 seconds

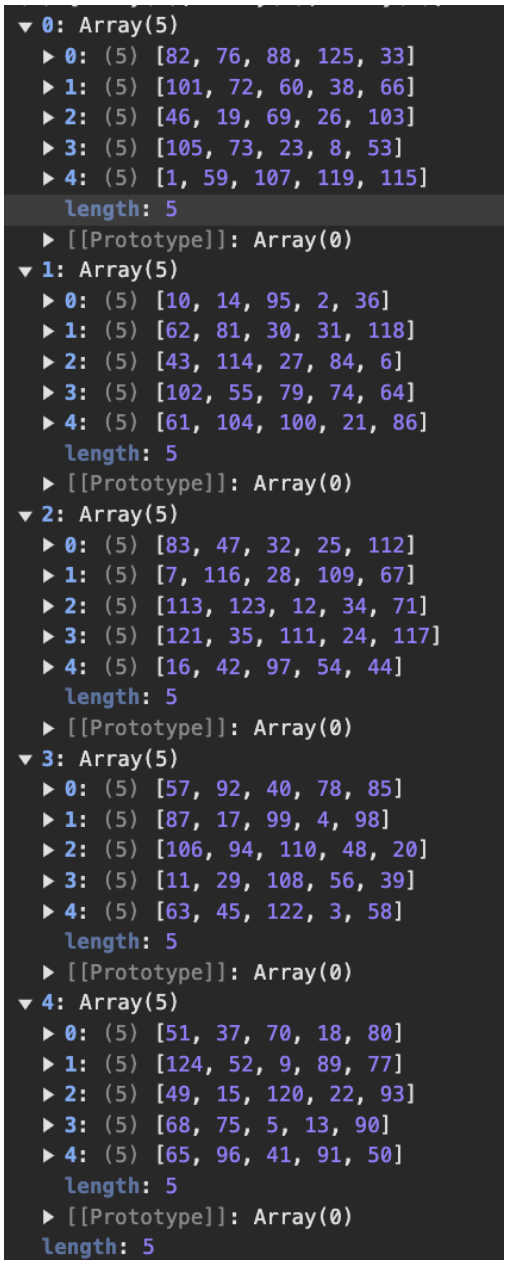
Final Objective Function: 5123

Iterations: 1000

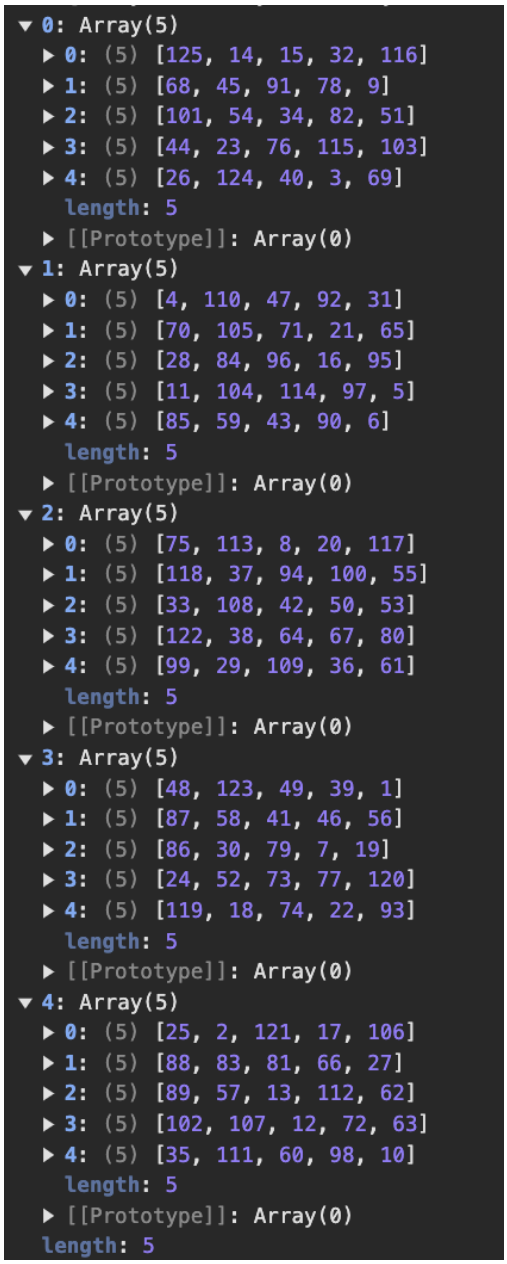


Percobaan 3

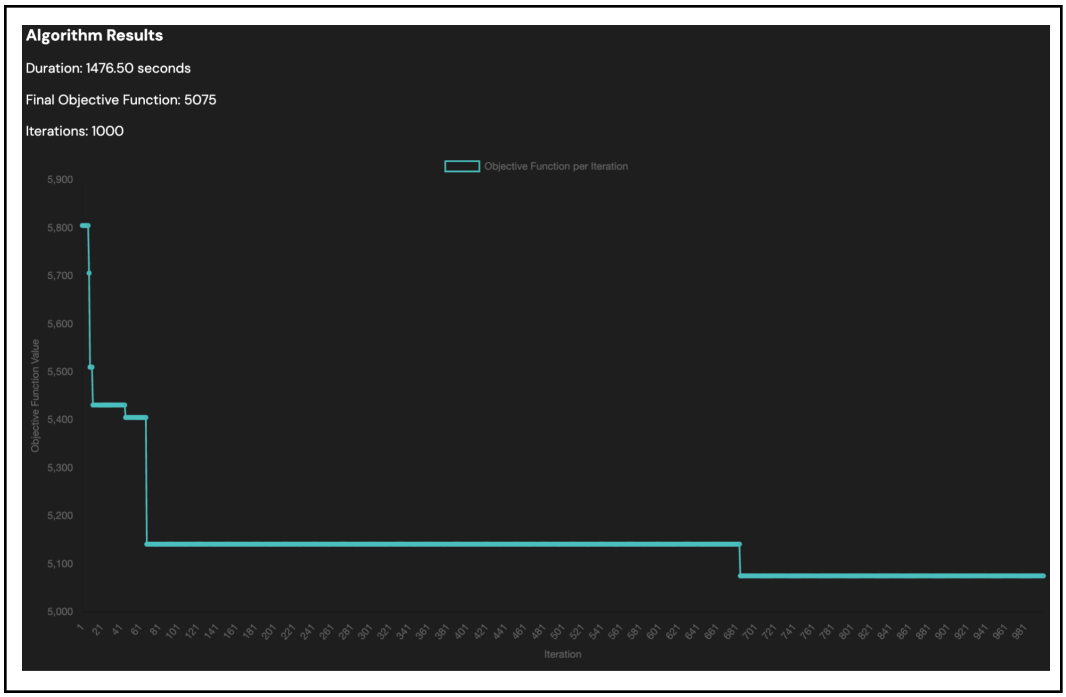
State Awal:



State Akhir:



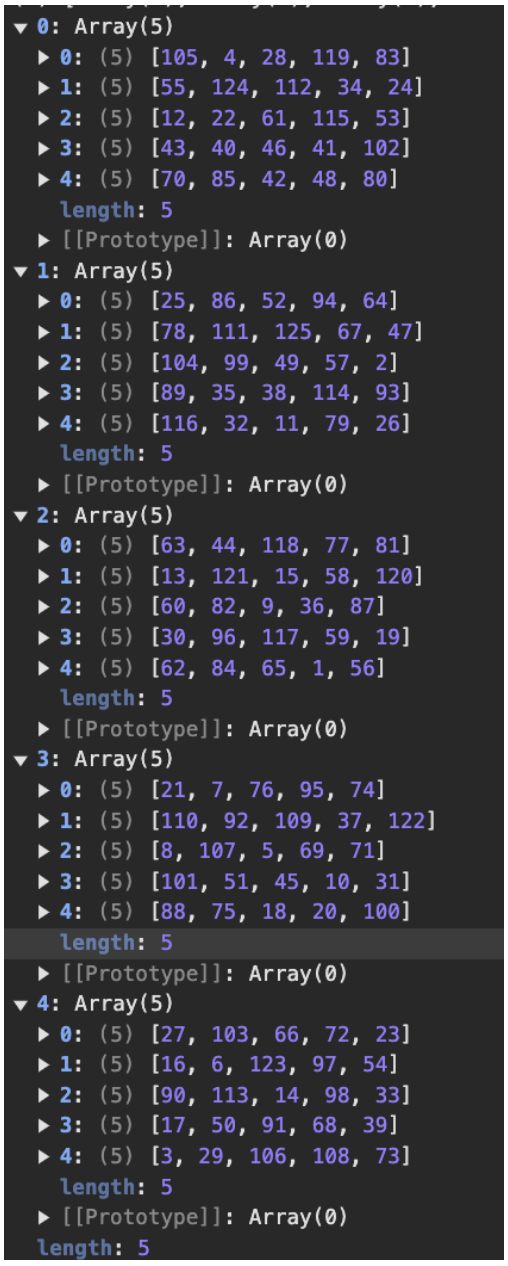
Hasil:


Algorithm Results
Duration: 1476.50 seconds
Final Objective Function: 5075
Iterations: 1000
Objective Function per Iteration
Objective Function Value
5,900
5,800
5,700
5,600
5,500
5,400
5,300
5,200
5,100
5,000
1
21
41
61
81
101
121
141
161
181
201
221
241
261
281
301
321
341
361
381
401
421
441
461
481
501
521
541
561
581
601
621
641
661
681
701
721
741
761
781
801
821
841
861
881
901
921
941
961
981
Iteration

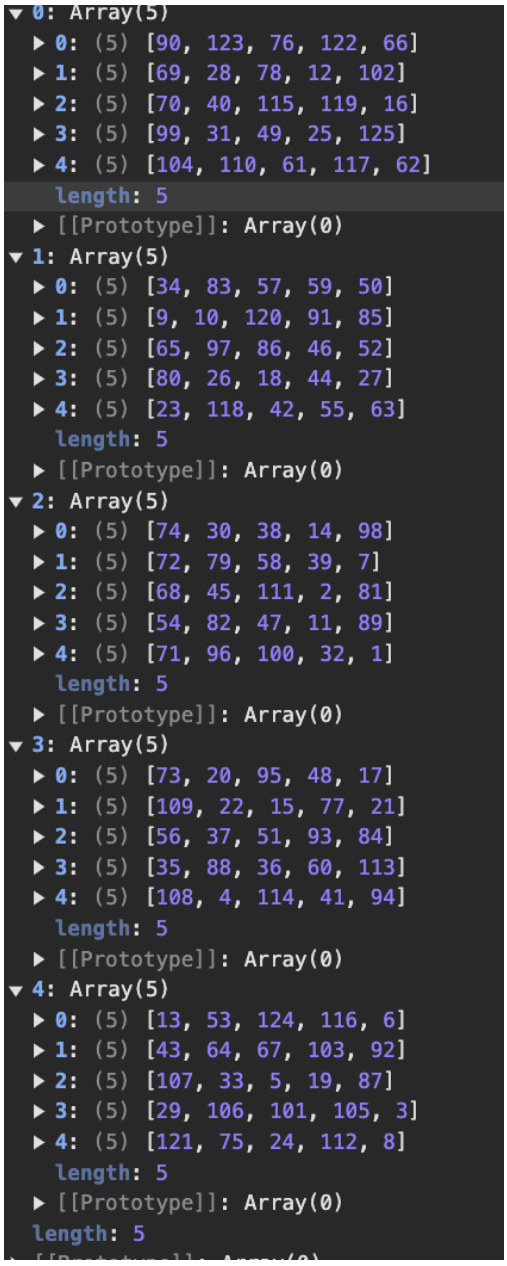
b. Variabel Kontrol (Jumlah Populasi = 100)
   i. Jumlah Iterasi = 100

Percobaan 1

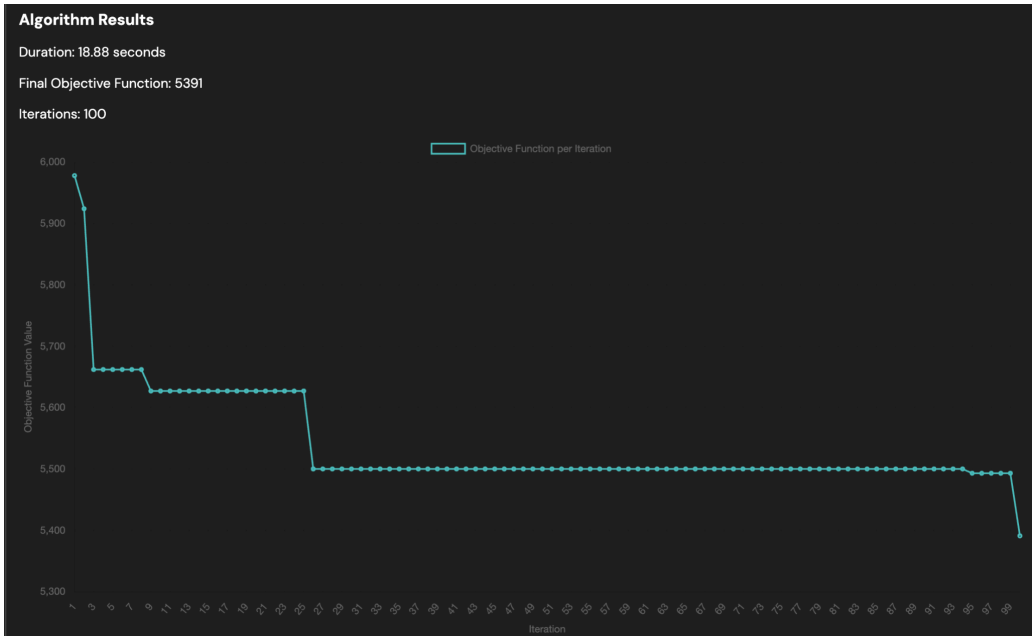
State Awal:



State Akhir:

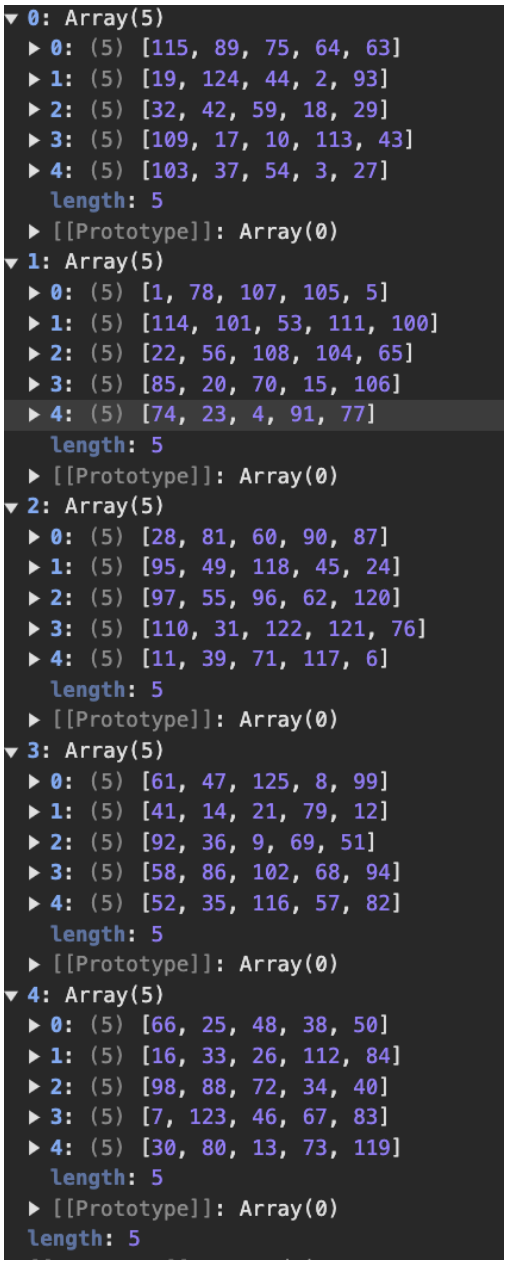


Hasil:

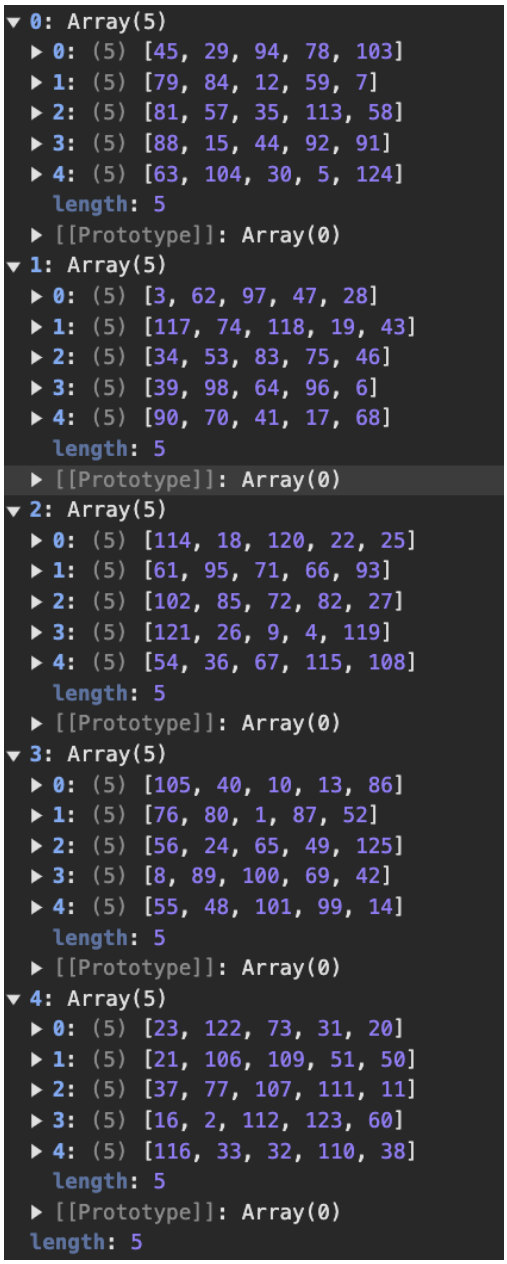

Algorithm Results
Duration: 18.88 seconds
Final Objective Function: 5391
Iterations: 100
Objective Function per Iteration
Objective Function Value
6,000
5,900
5,800
5,700
5,600
5,500
5,400
5,300
Iteration

Percobaan 2

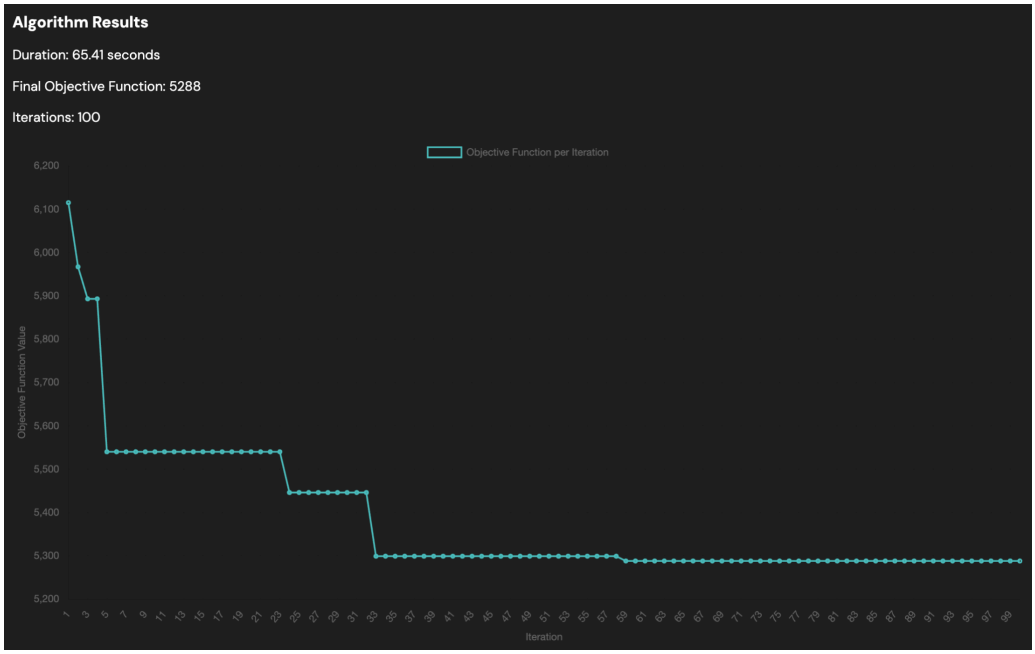
State Awal:



State Akhir:

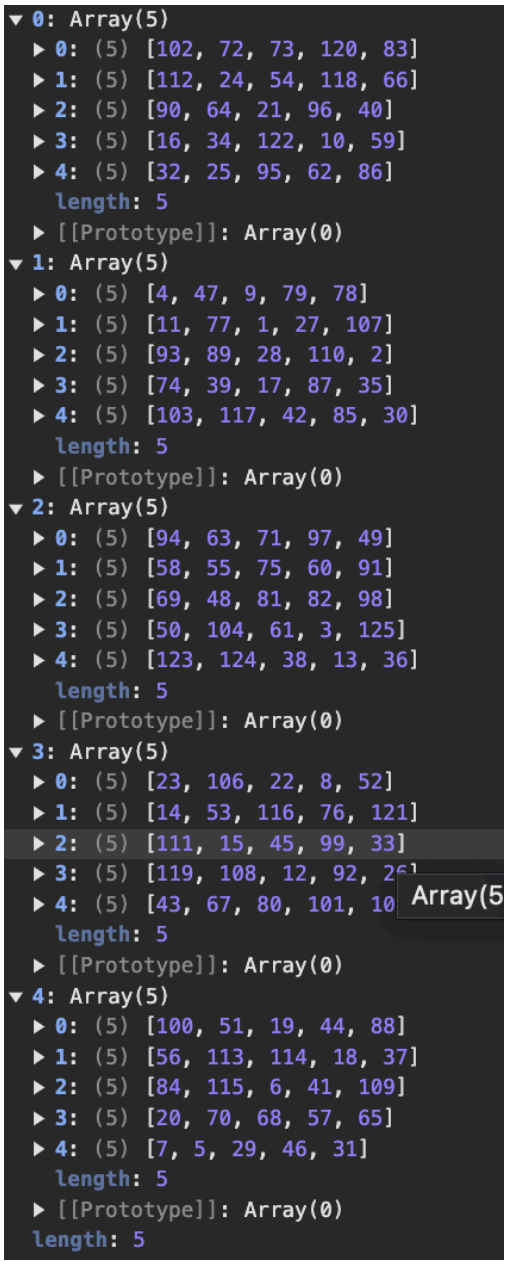


Hasil:

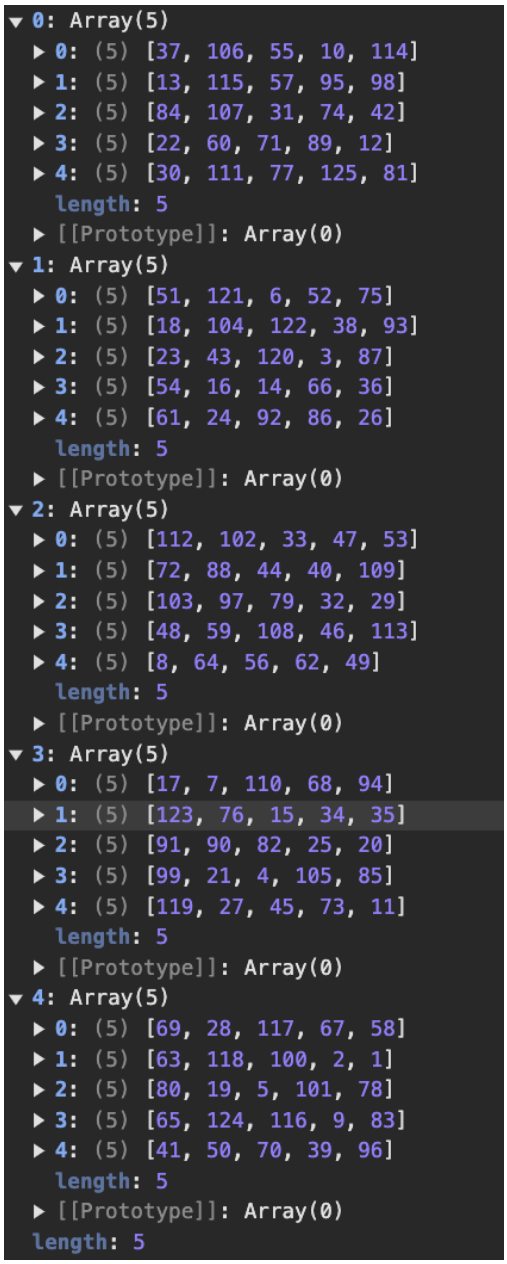

Algorithm Results
Duration: 65.41 seconds
Final Objective Function: 5288
Iterations: 100
Objective Function per Iteration
Objective Function Value
6,200
6,100
6,000
5,900
5,800
5,700
5,600
5,500
5,400
5,300
5,200
Iteration

Percobaan 3

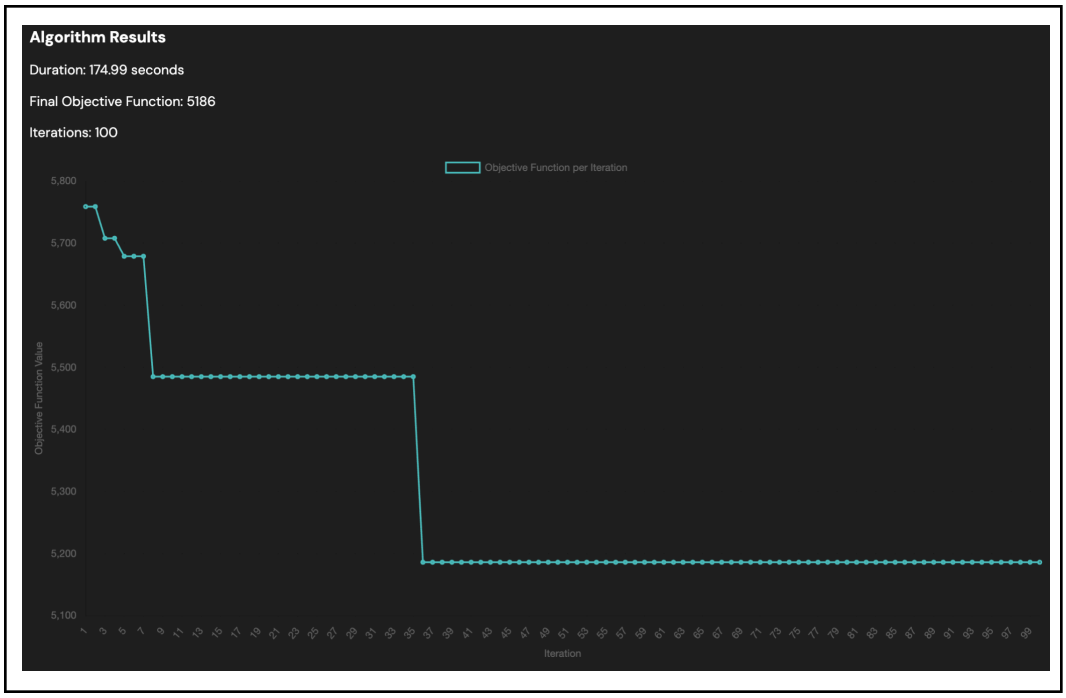
State Awal:



State Akhir:



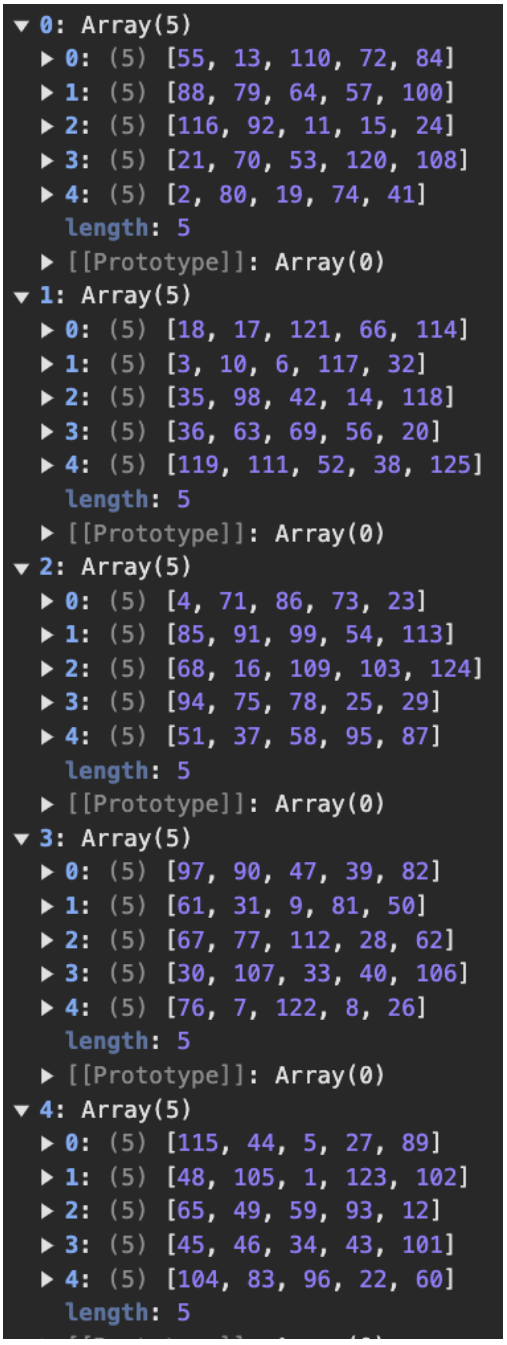
Hasil:


Algorithm Results
Duration: 174.99 seconds
Final Objective Function: 5186
Iterations: 100
Objective Function per Iteration
Objective Function Value
5,800
5,700
5,600
5,500
5,400
5,300
5,200
5,100
Iteration

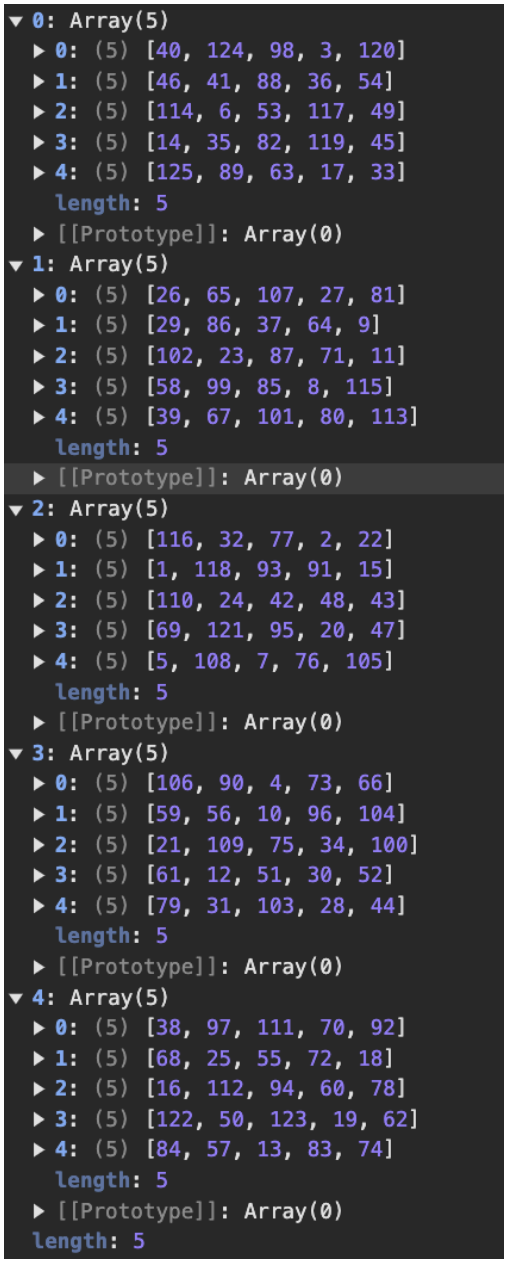
ii. Jumlah Iterasi = 500

Percobaan 1

State Awal:



State Akhir:



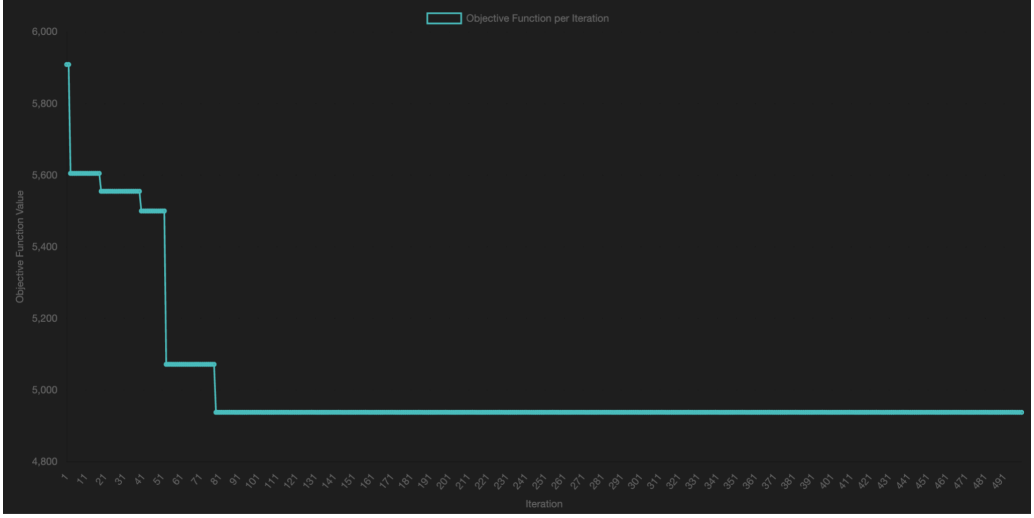
Hasil:

Algorithm Results

Duration: 58.99 seconds

Final Objective Function: 4938

Iterations: 500



Percobaan 2

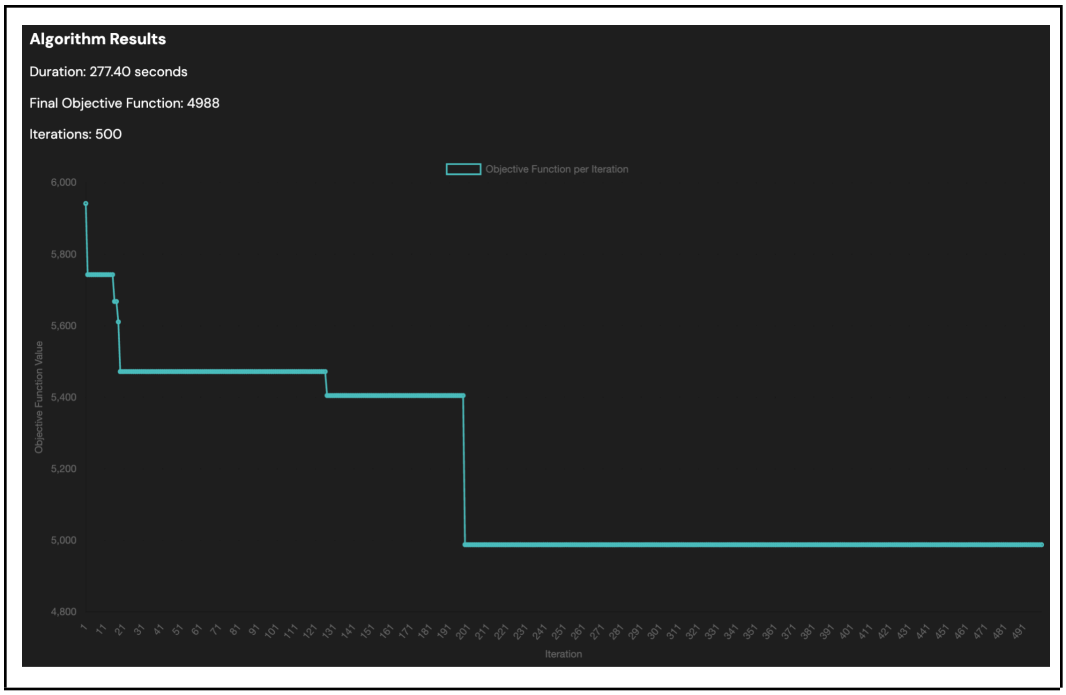
State Awal:

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

State Akhir:

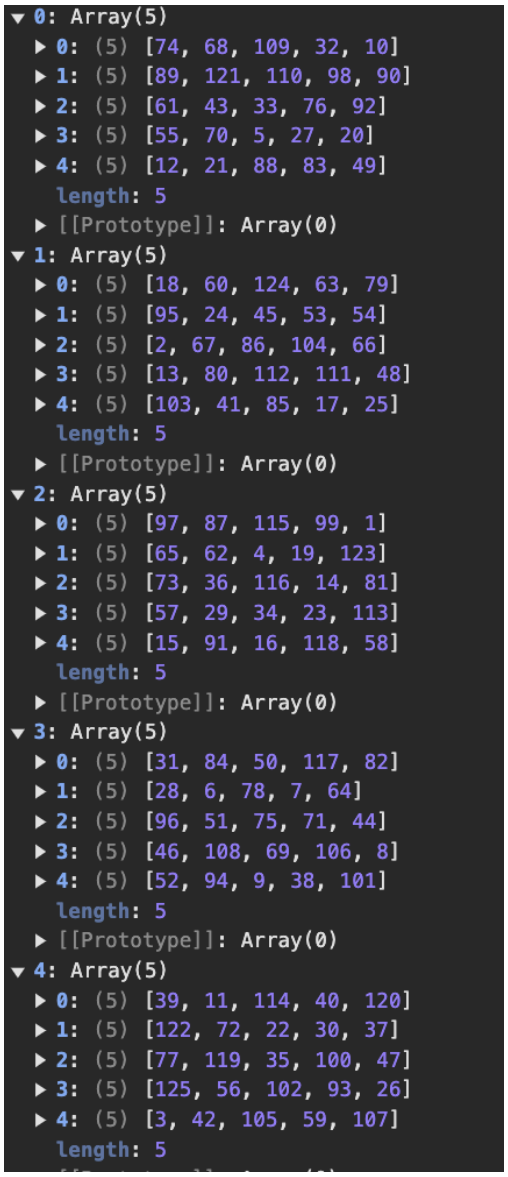
```
▼ 0: Array(5)
  ▶ 0: (5) [35, 1, 71, 107, 17]
  ▶ 1: (5) [63, 62, 55, 65, 122]
  ▶ 2: (5) [50, 29, 102, 86, 114]
  ▶ 3: (5) [124, 80, 23, 34, 93]
  ▶ 4: (5) [16, 51, 73, 70, 120]
    length: 5
  ▶ [[Prototype]]: Array(0)
▼ 1: Array(5)
  ▶ 0: (5) [98, 61, 26, 95, 30]
  ▶ 1: (5) [8, 24, 74, 111, 18]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

Hasil:

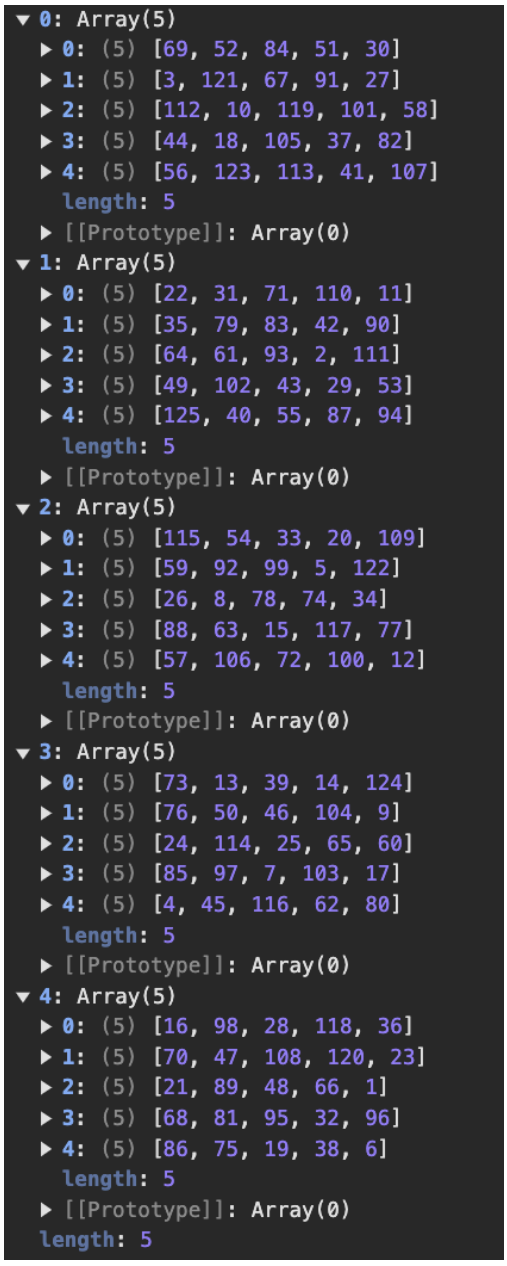

Algorithm Results
Duration: 277.40 seconds
Final Objective Function: 4988
Iterations: 500
Objective Function per Iteration
6,000
5,800
5,600
5,400
5,200
5,000
4,800
Objective Function Value
1
11
21
31
41
51
61
71
81
91
101
111
121
131
141
151
161
171
181
191
201
211
221
231
241
251
261
271
281
291
301
311
321
331
341
351
361
371
381
391
401
411
421
431
441
451
461
471
481
491
Iteration

Percobaan 3

State Awal:



State Akhir:



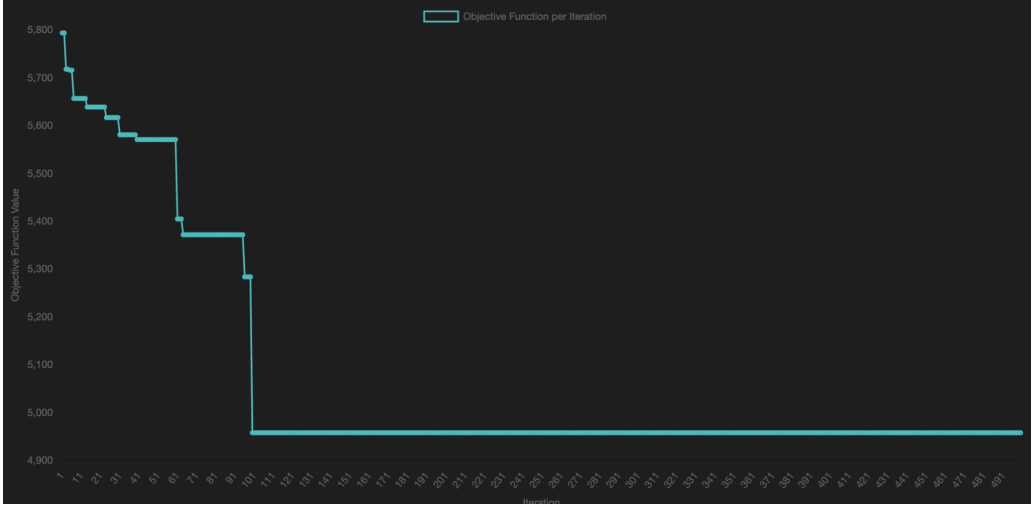
Hasil:

Algorithm Results

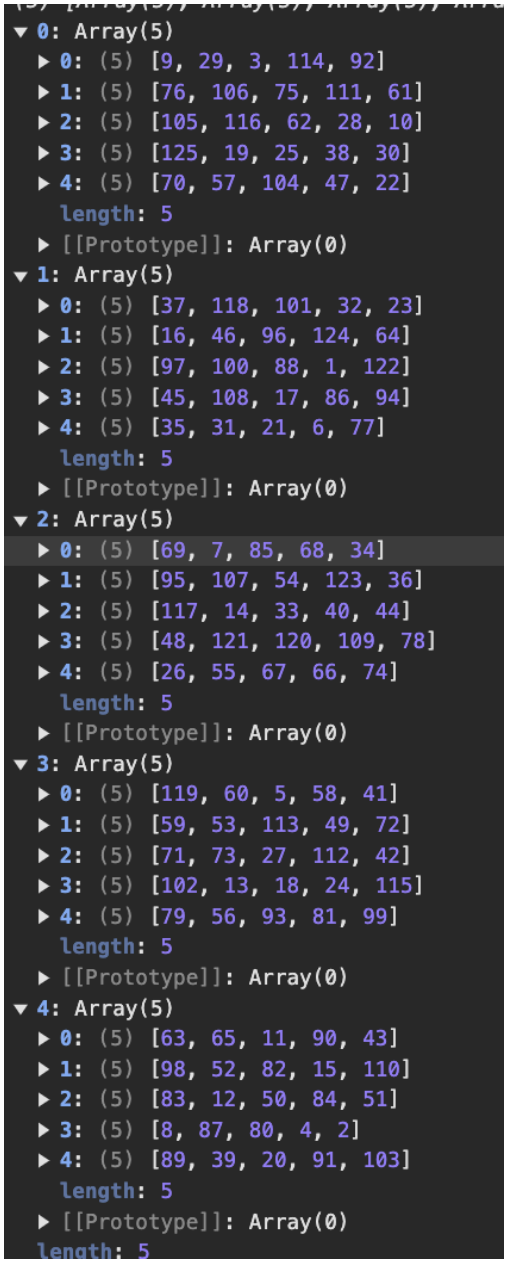
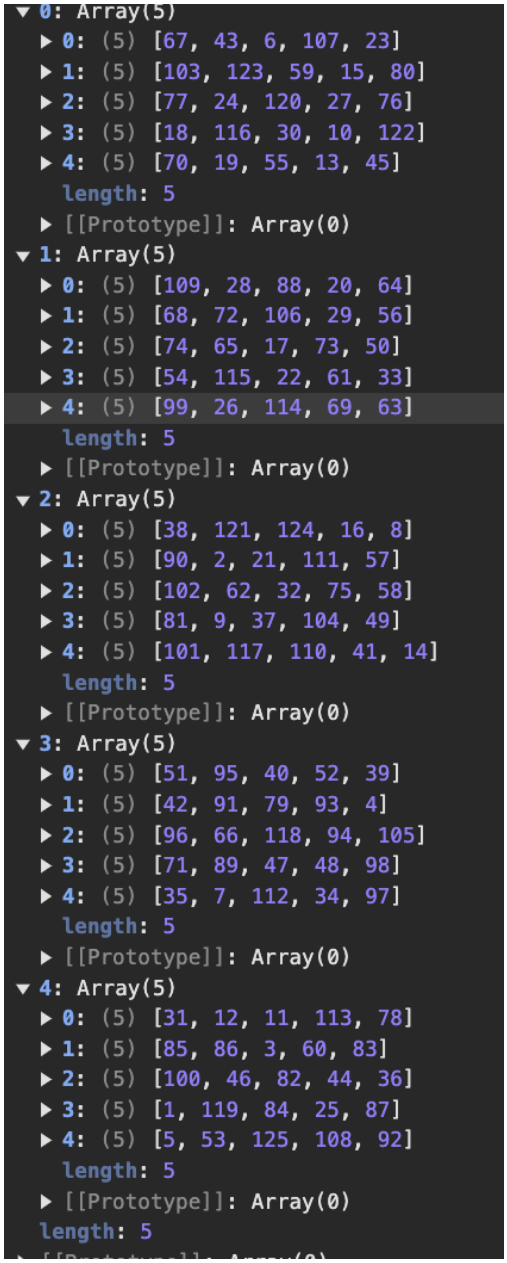
Duration: 83.31 seconds

Final Objective Function: 4958

Iterations: 500



iii. Jumlah Iterasi = 1000

Percobaan 1	
State Awal:  	State Akhir: 
Hasil:	

Algorithm Results

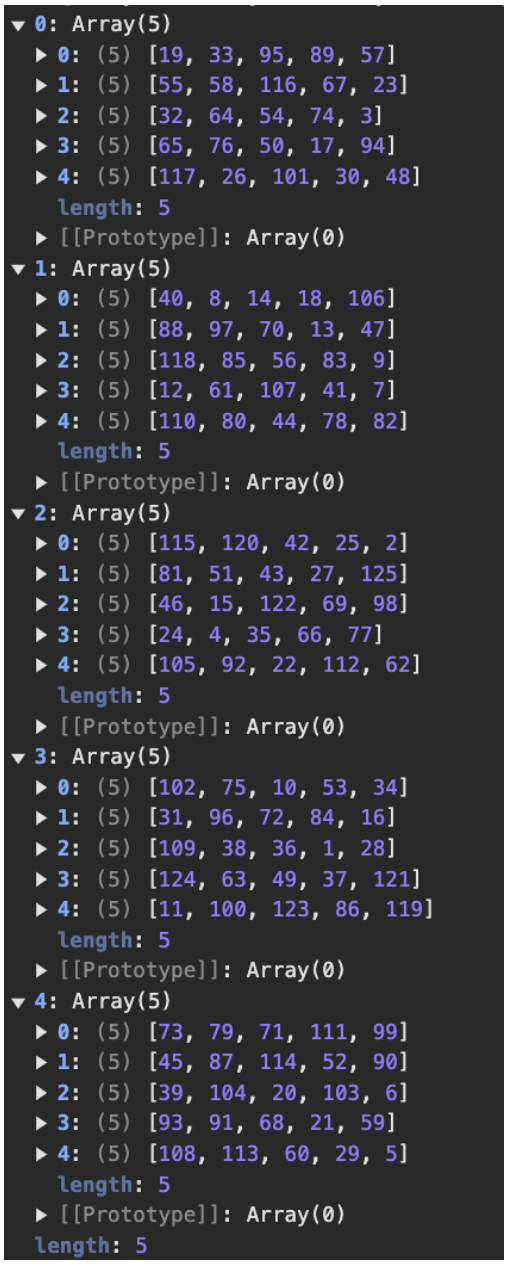
Duration: 109.74 seconds

Final Objective Function: 4962

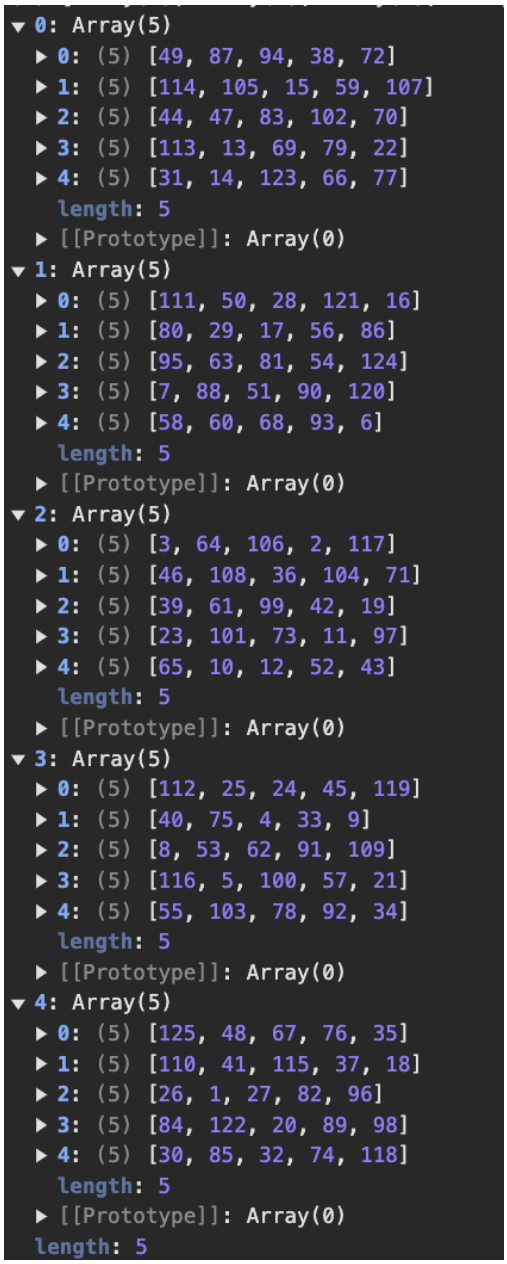
Iterations: 1000

Percobaan 2

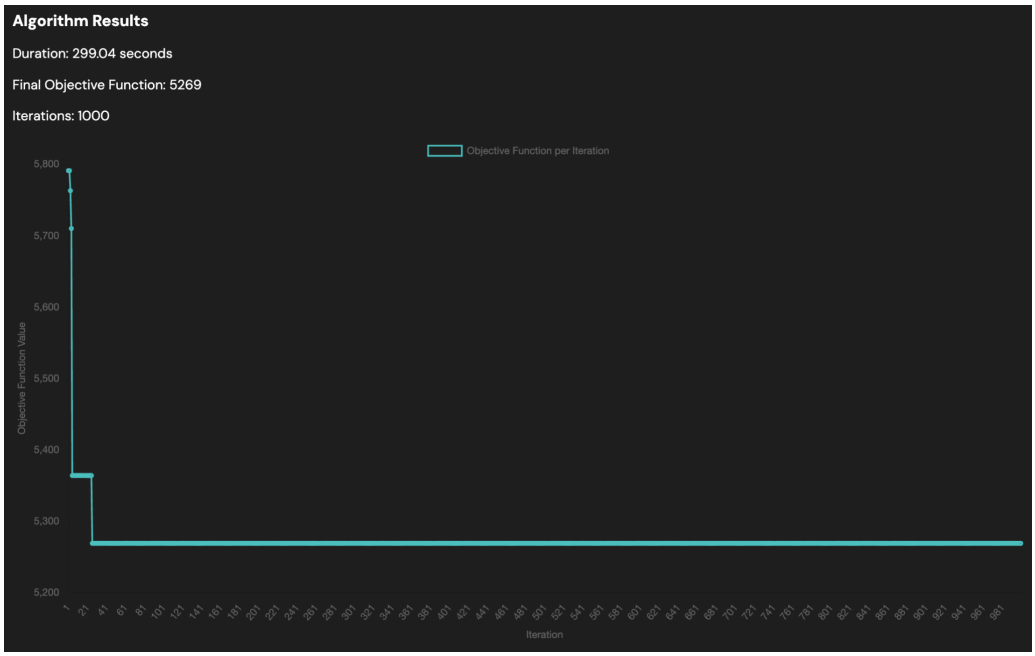
State Awal:



State Akhir:

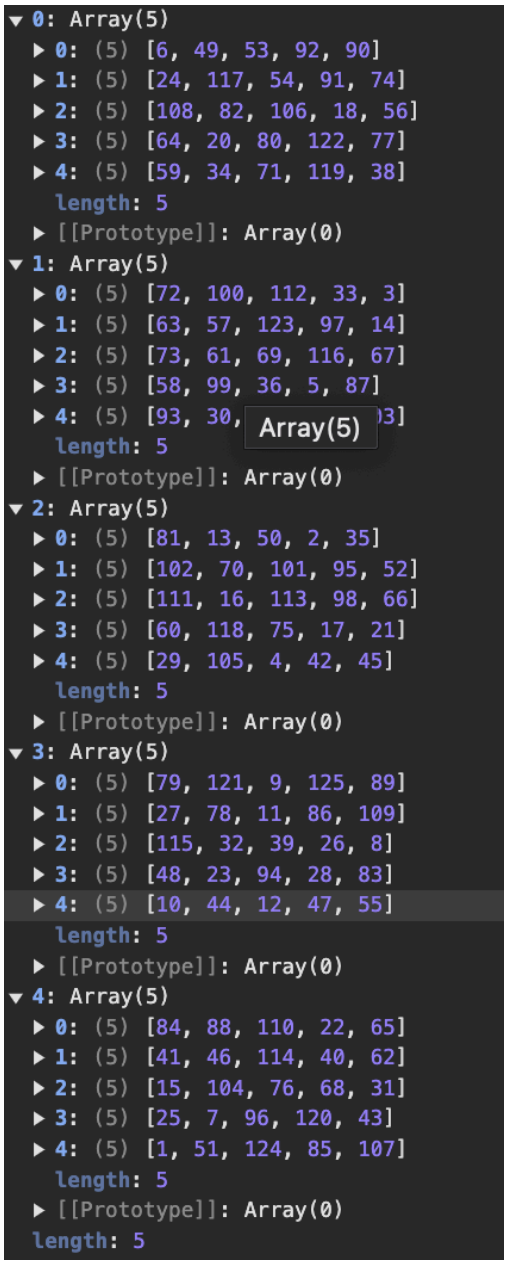


Hasil:

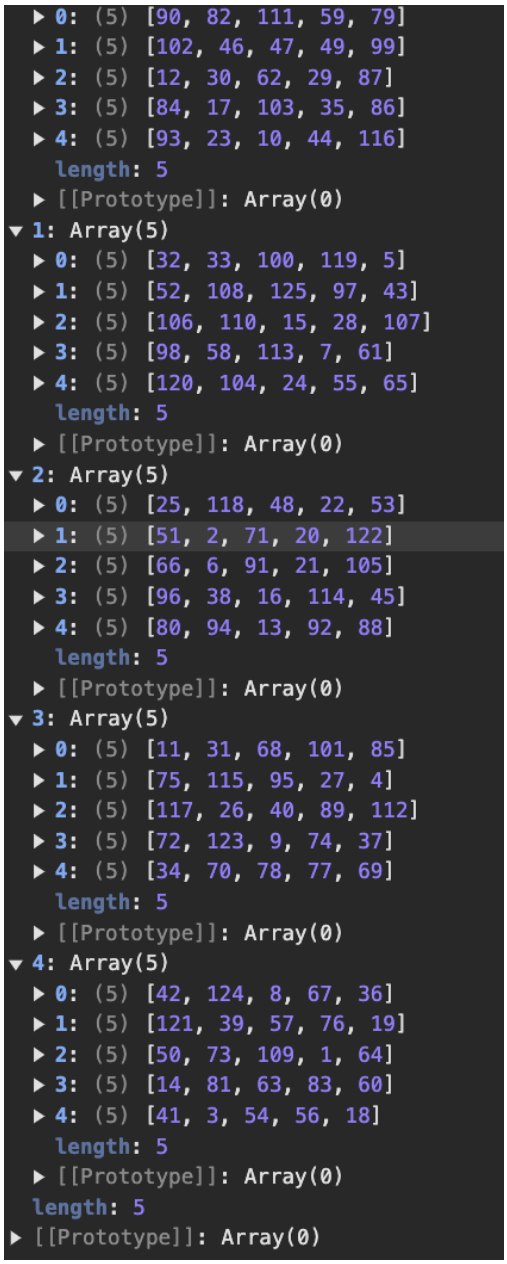

Algorithm Results
Duration: 299.04 seconds
Final Objective Function: 5269
Iterations: 1000
Objective Function per Iteration
Objective Function Value
5,800
5,700
5,600
5,500
5,400
5,300
5,200
1
21
41
61
81
101
121
141
161
181
201
221
241
261
281
301
321
341
361
381
401
421
441
461
481
501
521
541
561
581
601
621
641
661
681
701
721
741
761
781
801
821
841
861
881
901
921
941
961
981
Iteration

Percobaan 3

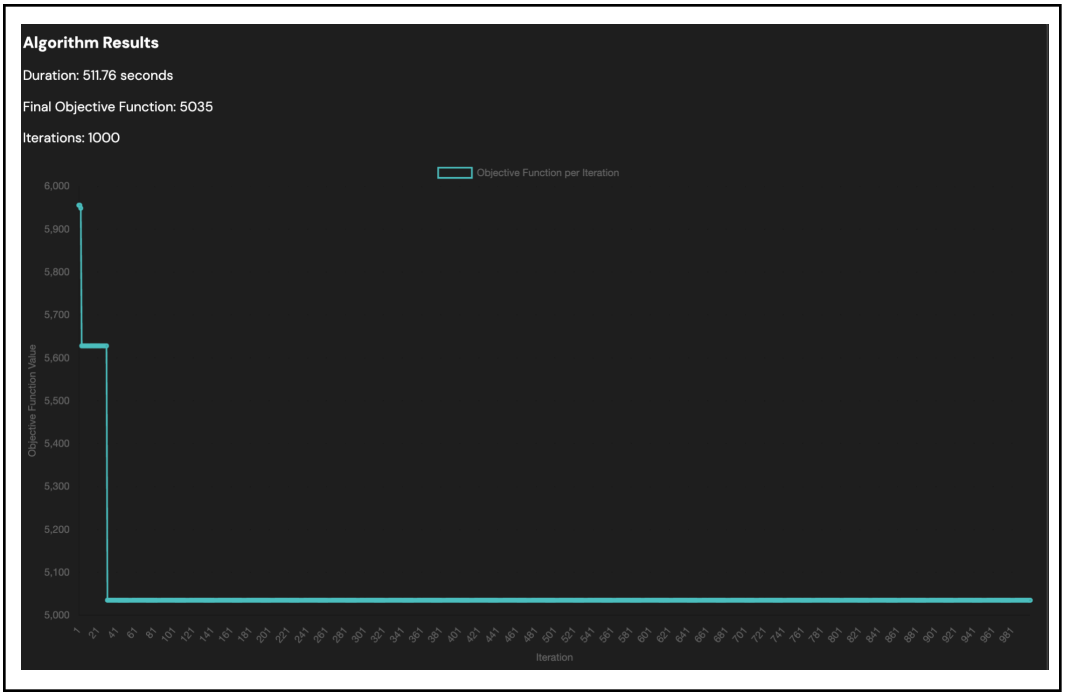
State Awal:



State Akhir:



Hasil:



d. Analisis dan Pembahasan

Berdasarkan observasi terhadap hasil eksperimen dari algoritma hill-climbing steepest ascent, dapat diamati nilai objective function yang diperoleh cukup konsisten dengan rata-rata poin sebesar 534. Selain itu, durasi pencarian dari algoritma ini juga terbilang relatif cepat yakni selama 15 detik dan memerlukan sejumlah 110 iterasi. Hal ini dikarenakan cara kerja algoritma yang lebih sederhana, yaitu hanya mencari tetangga terbaik dan membandingkannya dengan nilai state saat ini.

Sementara itu, pada eksperimen algoritma hill-climbing sideways move, diperoleh rata-rata durasi selama 60 detik yang memerlukan 180 iterasi, serta rata-rata nilai objektif sebesar 713, dengan distribusi yang kurang seragam. Ketidakseragaman distribusi nilai objektif disebabkan oleh adanya batasan jumlah sideways move untuk menghindari loop tak terbatas, namun menyebabkan algoritma cenderung berhenti lebih cepat tanpa mencapai perbaikan signifikan dalam beberapa kasus. Hal ini mengakibatkan hasil yang terjebak pada puncak lokal dan mengurangi kemampuan eksplorasi algoritma untuk menemukan solusi yang lebih optimal secara konsisten. Selain itu, dalam implementasinya, kami menggunakan heuristik *reset* sideways ketika ditemukan tetangga yang lebih baik daripada state saat ini, untuk meningkatkan eksplorasi algoritma, akan

tetapi hal ini juga menyebabkan peningkatan kompleksitas komputasi dan waktu eksekusi yang lebih panjang pada beberapa iterasi.

Pada eksperimen algoritma random restart hill climbing, diperoleh rata-rata waktu eksekusi sekitar 90 detik dengan banyak iterasi sejumlah 650 iterasi, serta rata-rata nilai objektif yang relatif konsisten sebesar 450. Peningkatan jumlah iterasi dan waktu eksekusi berbanding lurus dengan adanya jumlah pengulangan (*number of restarts*) yang bertujuan untuk memperbaiki hasil pencarian dengan hill-climbing steepest ascent.

Berdasarkan eksperimen algoritma stochastic hill-climbing, diperoleh rata-rata nilai sebesar 4000 dengan persebaran yang kurang seragam dan durasi rata-rata selama 35 detik. Ketidakseragaman algoritma stochastic hill-climbing merupakan akibat langsung dari pembangkitan tetangga secara acak, yang berbeda dengan ketiga algoritma hill-climbing sebelumnya. Hal ini menyebabkan algoritma memiliki kecenderungan untuk menemukan solusi yang lebih bervariasi, namun dengan stabilitas yang lebih rendah dalam mencapai nilai objektif yang optimal. Meskipun durasi eksekusi relatif cepat, performa algoritma ini sangat bergantung pada keberuntungan dalam pemilihan tetangga di setiap langkahnya.

Pada eksperimen algoritma simulated annealing, diperoleh rata-rata waktu eksekusi selama 50 detik dengan memanfaatkan 138000 iterasi, serta rata-rata nilai objektif yang cukup konvergen ke 600. Jumlah iterasi yang tinggi pada algoritma *simulated annealing* disebabkan oleh mekanisme pencarian yang menggabungkan eksplorasi dan eksploitasi secara bertahap melalui pendekatan probabilistik. Pada awalnya, algoritma menerima langkah-langkah yang mungkin memperburuk nilai objektif untuk menjelajahi ruang solusi secara luas, dengan harapan menemukan solusi global yang lebih baik. Proses ini memerlukan penurunan suhu yang lambat agar eksplorasi dapat berjalan optimal sebelum akhirnya fokus pada eksploitasi untuk memperbaiki solusi. Penurunan suhu yang bertahap dan banyaknya iterasi diperlukan untuk memastikan algoritma tidak terjebak di puncak lokal, sehingga waktu eksekusi yang panjang menjadi konsekuensinya.

Sementara itu, pada eksperimen algoritma genetik, digunakan 2 pendekatan yakni penggunaan jumlah iterasi dan jumlah populasi sebagai variabel terkontrol. Pada pendekatan jumlah iterasi sebagai variabel terkontrol, dapat diamati terdapat adanya

kecenderungan penurunan nilai objektif seiring bertambahnya jumlah iterasi, yang menunjukkan bahwa algoritma mampu melakukan eksploitasi solusi secara bertahap untuk mencapai nilai yang lebih optimal. Hal yang sama juga terjadi pada pendekatan jumlah populasi sebagai variabel terkontrol, di mana peningkatan jumlah populasi memungkinkan variasi genetik yang lebih luas, sehingga meningkatkan peluang algoritma untuk menemukan solusi yang lebih baik dan stabil di setiap generasi.

Salah satu faktor yang berkontribusi pada buruknya performa algoritma genetik dalam eksperimen ini, dengan rata-rata nilai objektif sebesar 5000, adalah metode pembentukan populasi yang digunakan. Dalam eksperimen, populasi baru dibentuk dengan menggantikan populasi lama secara sepenuhnya di setiap generasi. Metode ini mengurangi keberagaman genetik secara signifikan, yang membuat algoritma rentan terhadap *premature convergence*—di mana algoritma terlalu cepat terkunci pada solusi suboptimal tanpa cukup mengeksplorasi ruang solusi yang lebih luas. Ketika populasi lama sepenuhnya digantikan, informasi genetik yang potensial untuk membantu eksplorasi solusi yang lebih baik dapat hilang, menghambat kemampuan algoritma untuk melakukan perbaikan bertahap yang diperlukan untuk mencapai solusi yang lebih optimal. Selain itu, durasi eksekusi algoritma genetik cenderung lebih lama karena proses evaluasi populasi besar-besaran dan penggunaan operator seleksi, crossover, dan mutasi yang berulang di setiap iterasi. Kompleksitas ini diperparah ketika ukuran populasi besar dan jumlah iterasi tinggi, sehingga memerlukan sumber daya komputasi yang signifikan untuk mengolah setiap generasi dan menyelesaikan algoritma hingga mencapai hasil akhir.

Bab III
Kesimpulan dan Saran

Berdasarkan observasi hasil eksperimen, algoritma random restart hill-climbing dan simulated annealing menunjukkan performa yang lebih baik dibandingkan algoritma lainnya. Random restart hill-climbing mencatat rata-rata waktu eksekusi sekitar 90 detik dengan 650 iterasi, dan menghasilkan nilai objektif yang paling konsisten, yaitu sebesar 450. Keunggulan ini berasal dari pendekatan pengulangan (*restarts*) yang memungkinkan algoritma untuk memulai kembali pencarian dari titik awal yang berbeda, sehingga membantu mengatasi jebakan pada puncak lokal dan meningkatkan peluang menemukan solusi yang lebih optimal. Metode ini memberikan stabilitas yang lebih tinggi dalam mencapai hasil yang mendekati optimal.

Di sisi lain, algoritma simulated annealing juga menunjukkan hasil yang cukup konvergen dengan rata-rata nilai objektif sebesar 600 dalam waktu eksekusi 50 detik dan menggunakan 138000 iterasi. Keberhasilan algoritma ini terletak pada mekanisme penurunan suhu bertahap yang memungkinkan eksplorasi ruang solusi yang lebih luas di awal, disertai eksploitasinya yang lebih terfokus di akhir. Meskipun jumlah iterasinya tinggi, hal ini membantu simulated annealing menjelajahi berbagai solusi potensial dan meminimalkan risiko terjebak pada puncak lokal. Namun, durasi yang panjang tetap menjadi faktor yang perlu dipertimbangkan karena kompleksitas dan jumlah iterasi yang tinggi memerlukan sumber daya komputasi yang lebih besar.

Sebaliknya, algoritma genetik menunjukkan performa terburuk dalam eksperimen ini dengan rata-rata nilai objektif sebesar 5000. Kekurangan utama dari algoritma ini adalah metode pembentukan populasi yang menggantikan populasi lama secara sepenuhnya di setiap generasi, mengurangi keberagaman genetik dan menyebabkan *premature-convergence*. Hal ini menghambat kemampuan algoritma untuk menjelajahi ruang solusi yang lebih luas dan memperbaiki solusi secara bertahap. Selain itu, durasi eksekusi algoritma genetik cenderung lebih lama karena proses evaluasi populasi besar-besaran dan penggunaan operator seleksi, crossover, serta mutasi yang berulang di setiap iterasi. Dengan demikian, meskipun fleksibel dalam eksplorasi, algoritma genetik tidak sekompetitif algoritma local search lainnya dalam mencapai solusi optimal.

Bab IV

Pembagian Tugas

NIM	KONTRIBUSI
13522073	Membuat Form Membuat Plot Membuat Animasi Implementasi HillClimb Random Restart Fix bug Laporan
13522077	Membuat tampilan 3D Implementasi HillClimb Steepest Implementasi Simulated Annealing Fix bug Laporan
13522081	Membuat Form parameter Membuat animasi Implementasi HillClimb Stochastic Fix bug Laporan
13522115	Implementasi Genetic Algorithm Implementasi HillClimb Sideways Laporan Fix bug

Referensi

1. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010
2. "Magic Cubes," Trump. [Online]. Available: https://www.trump.de/magic-squares/magic-cubes/cubes-1.html. [Accessed: 01-Oct-2024]