BAB 5 IMPLEMENTASI

5.1 Struktur Kelas

Struktur kelas yang terdapat pada permasalahan penerapan metode *K-Means*-ACO untuk pengelompokan biji wijen berdasarkan sifat warna cangkang biji, kelas yang dipakai yaitu HomePage_ACO.java untuk memuat beberapa *method* dengan melakukan proses pengelompokan. Program tersebut akan ditampilkan dalam bentuk GUI (*Graphic User Interface*), sedangkan untuk memasukan data yang akan diproses menambahkan sebuah *library* jdk8.1 dan jxl.jar.

5.2 *Source Code*

Dalam menyelesaikan tahapan metode *K-Means*-ACO, dibutuhkan beberapa tahapan-tahapan penting. Tahapan terebut adalah Inisialisasi posisi awal koloni semut, menentukan keanggotaan data dengan *K-Means*, menghitung nilai rata-rata *fitness*, menghitung nilai probabilitas, menghitung *update pheromone*, menghitung *update* posisi, dan menghitung *Mean*.

5.2.1 Inisialisasi Posisi Awal Koloni Semut

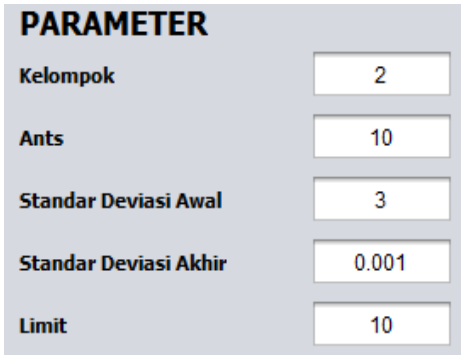
Terdapat 2 proses dalam tahap inisialisasi posisi awal koloni semut, yaitu:

1. Inisialisasi parameter
2. Menentukan posisi awal koloni semut

5.2.1.1 Inisialisasi Parameter

Dalam proses inisialisasi parameter terdapat dua komponen utama yaitu bagian eksternal yang berfungsi memasukan inputan parameter melalui user interface yang dilakukan oleh pengguna dan bagian internal yang berfungsi untuk memproses inputan pengguna.

Berikut adalah antarmuka untuk memasukkan nilai dari setiap parameter yang terdapat dalam metode *K-Means*-ACO, yang ditunjukkan pada Gambar 5.1.



Gambar 5.1 Antarmuka inisialisasi parameter

Berikut dibawah ini merupakan *source code* inisialisasi parameter untuk memproses nilai yang telah dimasukan oleh pengguna ditunjukkan pada *Source Code* 1.

```
1.  int jmlCluster = Integer.parseInt(TFKelompok.getText());
2.          int jmlIterasi = Integer.parseInt
3.  (TFIterasi.getText());
4.          int jmlSemut = Integer.parseInt(TFAnts.getText());
5.          int inisialisasiPheromone = 1;
6.          int kecepatanPheromone = 10;
7.          double koefisienPenguapan = 0.99;
8.          double standarDeviasiAwal = Double.parseDouble
9.  (TFStandarDeviasiAwal.getText());
10.         double standarDeviasiAkhir = Double.parseDouble
11. (TFStandarDeviasiAkhir.getText());
12.         int limit = Integer.parseInt(TFLimit.getText());
13.         int inputParameter = Integer.parseInt
14. (TFInputParameter.getText());
```

***Source Code* 1. Inisialisasi parameter**

Proses ini merupakan inisialisasi parameter awal dengan menggunakan metode *K-Means*-ACO untuk proses pengelompokan data pada biji wijen. Pada baris 4 sampai 6 merupakan inisialisasi yang sudah ditetapkan pada jurnal penelitian.

5.2.1.2 Menentukan Posisi Awal Koloni Semut

Source code inisialisasi posisi awal koloni semut ini adalah untuk menentukan sebuah posisi awal semut yang dilakukan secara acak dari jumlah semut dan jumlah c*luster*. Penentuan posisi semut yang dilakukan secara acak tetap pada batas minValue dan batas maxValue. Berikut merupakan *source code* inisialisasi posisi semut dapat dilihat pada *Source Code* 2.

```
1.      public double[] findMaxValue2D(double[][] data) {
2.          double[] maxValue2D = new double[data[0].length];
3.          System.out.print("Max :" + "\t");
4.          for (int i = 0; i < data[0].length; i++) {
5.              double maxValue = data[0][i];
6.              for (int j = 0; j < data.length; j++) {
7.                  if (data[j][i] > maxValue) {
8.                      maxValue = data[j][i];
9.                  }
10.             }
11.             maxValue2D[i] = maxValue;
12.             System.out.print(maxValue2D[i] + "\t");
13.         }
14.         System.out.println("");
15.         return maxValue2D;
16.     }

17      public double[] findMinValue2D(double[][] data) {
18.         double[] minValue2D = new double[data[0].length];
19.         System.out.print("Min :" + "\t");
20.         for (int i = 0; i < data[0].length; i++) {
21.             double minValue = data[0][i];
```

```
22.             for (int j = 0; j < data.length; j++) {
23.                 if (data[j][i] < minValue) {
24.                     minValue = data[j][i];
25.                 }
26.             }
27.             minValue2D[i] = minValue;
28.             System.out.print(minValue2D[i] + "\t");
29.         }
30.         System.out.println("");
31.         return minValue2D;
32.     }
    //Inisialisasi posisi awal semut
33.     double[][][] inisialisasiSemut(int jmlSemut, int
34. jmlCluster, double[] maxValue2D, double[] minValue2D) {
35.         System.out.println("");
36.         double[][][] semut = new
37. double[jmlSemut][jmlCluster][maxValue2D.length];

38.         System.out.println("|| Inisialisasi Semut ||");
39.         for (int i = 0; i < jmlSemut; i++) {
40.             System.out.print("Semut " + (i + 1) + "\t");
41.             for (int j = 0; j < jmlCluster; j++) {
42.                 for (int k = 0; k < maxValue2D.length; k++)
43. {
44.                     do {
45.                         semut[i][j][k] = minValue2D[k] +
46. (double) (Math.random() * maxValue2D[k]);
47.                     } while (semut[i][j][k] >
48. maxValue2D[k]);
49.                     System.out.print(semut[i][j][k] +
50. "\t");
51.                 }
52.             }
53.             System.out.println("");
54.         }
55.         return semut;
56.     }
```

***Source Code* 2. Menentukan posisi awal koloni semut**

Source code 2 adalah proses untuk melakukan inisialisasi posisi awal koloni semut, dimana terdapat 3 proses yaitu menentukan nilai Maksimum, nilai minimum, dan inisialisasi semut. Pada *method findMaxValue2D* yang berada di baris 1 sampai 16 merupakan proses untuk menemukan batas nilai maksimum pada data biji wijen, sedangkan method *findMinValue2D* yang berada di baris 17 sampai 32 merupakan proses untuk menentukan batas nilai minimum pada data biji wijen, dan method *inisialisasiSemut* yang berada pada baris 33 sampai 56 merupakan proses untuk melakukan posisi awal semut pada setiap titik pusat *cluster*. Inisialisasi posisi awal semut ini dilakukan secara acak dengan menggunakan *Math.random* yang ditunjukan pada baris 45.

5.2.2 Menentukan Keanggotaan Data dengan *K-Means*

Terdapat 2 proses dalam menentukan keanggota data dengan *K-Means* yaitu:

1. Menghitung jarak data ke titik pusat *cluster*.
2. Menentukan keanggotaan data.

5.2.2.1 Menghitung Jarak ke Data Titik Pusat *Cluster*

Source code menghitung jarak data digunakan untuk menentukan jarak masing – masing data ke masing – masing titik pusat *cluster* menggunakan Persamaan 2.1. *Method* ini menggunakan dua inputan yaitu titik pusat dan data latih. Berikut merupakan proses menghitung jarak data dapat dilihat pada *Source Code* 3.

```
1.  double[][] hitungJarakData(double[][] pusat, double[][]
2.  data) {
3.          double[][]jarakData=new
4.  double[pusat.length][data.length];
5.          double[][][] jarak = new
6.  double[pusat.length][data.length][data[0].length];
7.          for (int i = 0; i < pusat.length; i++) {
8.              for (int j = 0; j < data.length; j++) {
9.                  for (int k = 0; k < data[j].length; k++) {
10.                     //rumus jarak data ke titik pusat
11.                    jarak[i][j][k] = Math.pow((pusat[i][k] -
12. data[j][k]), 2);
13.                     jarakData[i][j] = jarakData[i][j] +
14. jarak[i][j][k];
15.                 }
16.                 //rumus jarak C1 dan jarak C2
17.                 jarakData[i][j] =
18. Math.sqrt(jarakData[i][j]);
19.             }
20.         }
21.         return jarakData;
22.     }
```

Source Code* 3. Menghitung jarak data ke titik pusat *cluster

Source code 3 adalah menghitung jarak data ke titik pusat *cluster* menggunakan nama *method* hitungJarakData. Pada baris 3 sampai 6 adalah proses inisialisasi parameter untuk menghitung jarak data, baris 7 sampai 20 adalah proses untuk menghitung jarak data ke titik pusat *cluster* dengan menggunakan rumus *Euclidean Distance* yang berada di baris 11 sampai 18.

5.2.2.2 Menentukan Keanggotaan Data

Source code menentukan keanggotaan data menggunakan dua masukan yaitu jarak data ke pusat *cluster* dan jumlah *cluster*. *Method* ini berfungsi untuk mengelompokan data ke dalam *cluster* menggunakan *K-Means*. Data dikelompokkan berdasarkan jarak minimal data ke pusat *cluster*. Berikut merupakan *source code* menentukan *cluster* dengan *K-Means* dapat dilihat pada *Source Code* 4.

```
1.  int[] tentukanCluster(double[][] jarak, int jmlCluster) {
2.          int[] posisiCluster = new int[jarak[0].length];
3.          for (int i = 0; i < jarak[0].length; i++) {
4.              double minJarak = jarak[0][i];
5.              posisiCluster[i] = 0;
6.              for (int j = 0; j < jarak.length; j++) {
7.                  if (jarak[j][i] < minJarak) {
8.                      minJarak = jarak[j][i];
9.                      posisiCluster[i] = j;
10.                 }
11.             }
12.         }
13.         return posisiCluster;
14.     }
```

***Source Code* 4. Menentukan keanggotaan data**

Source Code 4 adalah proses untuk menentukan keanggotaan data dengan nama *method* tentukanCluster. Baris 2 merupakan inputan program yang digunakan untuk proses inisialisasi posisi *cluster*, dan baris 3 sampai 12 adalah proses untuk menentukan keanggotaan data, hasil dari keanggotaan data ditentukan dari nilai jarak minimum.

5.2.3 Menghitung Nilai Rata-Rata *Fitness*

Tedapat 4 proses dalam menghitung nilai rata – rata *fitness* yaitu :

1. Menghitung jarak data *fitness*.
2. Menghitung jumlah anggota *cluster*.
3. Menghitung nilai *fitness* setiap *cluster*.
4. Menghitung *fitness* semut.

5.2.3.1 Menghitung Jarak Data *Fitness*

Source code menghitung *fitness* setiap data ke setiap *cluster* menggunakan dua masukan yaitu titik pusat *cluster* dan data latih. *Method* ini berfungsi untuk mendapatkan hasil perhitungan *fitness* setiap data pada setiap *cluster.* Berikut merupakan *source code* menghitung *fitness* setiap data ke setiap *cluster* dapat dilihat pada *Source Code* 5.

```
1.      double[][] hitungJarakData Fitness(double[][] pusat,
2.  double[][] data) {
3.          double[][] jarakData Fitness = new
4.  double[pusat.length][data.length];
5.          double[][][] jarak = new
6.  double[pusat.length][data.length][data[0].length];
7.          for (int i = 0; i < pusat.length; i++) {
8.              for (int j = 0; j < data.length; j++) {
9.                  for (int k = 0; k < data[j].length; k++) {
10.                     jarak[i][j][k] = Math.pow((pusat[i][k]
11. - data[j][k]), 2);
12.                     jarakData Fitness[i][j] =
13. jarakData Fitness[i][j] + jarak[i][j][k];
```

```
14.                 }
15.                 jarakData<i>Fitness</i>[i][j] =
16. jarakData<i>Fitness</i>[i][j] / data[j].length;
17.             }
18.         }
19.         return jarakData<i>Fitness</i>;
20.     }
```

Source Code* 5. Menghitung jarak data *fitness

Proses ini adalah untuk menghitung jarak data *fitness* dengan nama method hitungJarakData*Fitness*. Pada baris ke 3 dan 5 adalah proses untuk inisialisasi parameter jarak data *fitness* yang dihitung jarak rata-ratanya dari jumlah jarak data tersebut. Baris 7 sampai 18 adalah proses untuk menghitung jarak data *fitness*.

5.2.3.2 Menghitung Jumlah Anggota *Cluster*

Source code 6 adalah proses untuk menghitung jumlah anggota *cluster* menggunakan dua masukan yaitu posisi *cluster* dan jumlah *cluster*. *Method* ini digunakan untuk mencari nilai rata-rata *fitness.* Berikut merupakan proses menghitung jumlah anggota *cluster* dapat dilihat pada *Source Code* 6.

```
1.  double[] hitungJumlahAnggotaCluster(int[] posisiCluster,
2.  int jmlCluster) {
3.          double[] jmlAnggotaCluster = new
4.  double[jmlCluster];
5.          for (int i = 0; i < posisiCluster.length; i++) {
6.              jmlAnggotaCluster[(posisiCluster[i])]++;
7.          }
8.          System.out.println("");
9.          System.out.println("|| Jumlah Anggota Cluster
10. ||");
11.         for (int i = 0; i < jmlCluster; i++) {
12.             System.out.println("Cluster " + (i + 1) +
13. "\t" + jmlAnggotaCluster[i]);
14.         }
15.         return jmlAnggotaCluster;
16.     }
```

Source Code* 6. Menghitung jumlah anggota *cluster

Proses untuk melakukan perhitungan jumlah anggota *cluster* ditampilkan pada baris 3 sampai 14, proses ini dilakukan setelah mengetahui nilai keanggotaan data setiap *cluster*.

5.2.3.3 Menghitung Nilai *Fitness* Setiap *Cluster*

Source code menghitung rata-rata nilai *fitness* setiap *cluster* menggunakan tiga masukan yaitu jarak data *fitness,* posisi *cluster,* dan jumlah anggota *cluster*. *Method* ini berfungsi untuk mendapatkan hasil perhitungan rata – rata keseluruhan *fitness* pada setiap *cluster.* Berikut merupakan *source code* menghitung rata – rata nilai *fitness* setiap *cluster* dapat dilihat pada *Source Code* 7.

```
1.  double[] hitung<i>Fitness</i>(double[][] jarak<i>Fitness</i>, int[]
2.  posCluster, double[] jmlAnggotaCluster) {
```

```
3.          double[] fitnessCluster = new
4.  double[jmlAnggotaCluster.length];
5.          for (int i = 0; i < jmlAnggotaCluster.length; i++)
6.  {
7.              for (int j = 0; j < jarakFitness[0].length;
8.  j++) {
9.                  if (posCluster[j] == (i)) {
10.                     fitnessCluster[i] = fitnessCluster[i]
11. + jarakFitness[i][j];
12.                 }
13.             }
14.         }
15.         System.out.println("");
16.         System.out.println("|| Fitness Cluster ||");
17.         for (int i = 0; i < jmlAnggotaCluster.length; i++)
18. {
19.             System.out.print("Cluster " + (i + 1) + "\t");
20.             fitnessCluster[i]=fitnessCluster[i]/
21. jmlAnggotaCluster[i];
22.             if (Double.isNaN(fitnessCluster[i])) {
23.                 fitnessCluster[i] = 0.0;
24.             }
25.             System.out.println(fitnessCluster[i] + "\t");
26.         }
27.         return fitnessCluster;}
```

Source Code* 7. Menghitung nilai *fitness* setiap *cluster

Proses ini menggunakan nama *method* hitung*Fitness*. Baris 14-21 adalah proses untuk menghitung rata-rata nilai *fitness* setiap *cluster*, sedangkan pada baris ke 10 adalah proses untuk menjumlahkan nilai *fitness* pada masing-masing *cluster*. Pada baris 17 sampai 26 adalah proses untuk menghitung rata-rata nilai *fitness* dari kedua *cluster*, dan pada baris ke 20 adalah proses perhitungan nilai rata-rata *fitness* setiap *cluster*.

5.2.3.4 Menghitung *Fitness* Semut

Source Code menghitung *fitness* semut menggunakan dua masukan yaitu letak semut dan jumlah *cluster*. *Method* ini berfungsi untuk menentukan hasil akhir *fitness* semut. Berikut merupakan *source code* menghitung *fitness* semut dapat dilihat pada *Source Code* 8.

```
1.  double[] hitungFitnessSemut(double[][][] semut, int
2.  jmlCluster) {
3.          double[][] pusat = new
4.  double[semut[0].length][semut[0][0].length];
5.          double[] fitnessSemut = new double[semut.length];
6.          System.out.println("");
7.          System.out.println("|| Fitness Semut ||");
8.          for (int i = 0; i < semut.length; i++) {
9.              for (int j = 0; j < semut[0].length; j++) {
10.                 for (int k = 0; k < semut[0][0].length;
11. k++) {
12.                     pusat[j][k] = semut[i][j][k];
13.                 }
14.             }
15.
```

```
16.             double[][] jarakData = hitungJarakData(pusat,
17.   dataLatih);
18.             int[] posisiCluster =
19.   tentukanCluster(jarakData, jmlCluster);
20.             double[] jmlAnggotaCluster =
21.   hitungJumlahAnggotaCluster(posisiCluster, jmlCluster);
22.             double[][] jarakData_Fitness =
23.   hitungJarakData_Fitness(pusat, dataLatih);
24.             double[] _fitness_Cluster =
25.   hitung_Fitness(jarakData_Fitness, posisiCluster,
26.   jmlAnggotaCluster);
27.             double _fitness_Total = 0;
28.             boolean pilih = true;
29.             for (int j = 0; j < _fitness_Cluster.length;
30.   j++) {
31.                 if (_fitness_Cluster[j] == 0) {
32.                     pilih = false;
33.                 }
34.                 _fitness_Total = _fitness_Total +
35.   _fitness_Cluster[j];
36.             }
37.
38.             _fitness_Total = _fitness_Total /
39.   _fitness_Cluster.length;
40.             if (pilih == false) {
41.                 _fitness_Total = _fitness_Total * 1000;
42.             }
43.             _fitness_Semut[i] = _fitness_Total;
44.
45.             System.out.println("Semut " + (i + 1) + "\t" +
46.   _fitness_Semut[i] + "\t");
47.         }
48.         return _fitness_Semut;
        }
```

***Source Code* 8. Menghitung *fitness* semut**

Pada proses menghitung *fitness* semut ditunjukan pada baris ke 8 sampai 46. Dari masing-masing semut memiliki titik pusat untuk dihitung nilai *fitness* terhadap data wijen. Proses perulangan inisialisasi titik pusat semut ditunjukkan pada baris ke 8 sampai 12. Pada proses menentukan *fitness* semut digunakan bebarap *method* pada proses sebelumnya yaitu pada baris 15 – 25. Setelah itu dilakukan proses menjumlahkan nilai *fitness* setiap cluster ditunjukkan pada baris ke 33. Sedangkan untuk hasil nilai akhir *fitness*s dihitung rata – ratanya ditunjukkan pada baris 37.

5.2.4 Menghitung Nilai Probabilitas

Source code 9 adalah proses untuk menghitung nilai probabilitas menggunakan satu masukan yaitu *fitness* populasi. *Method* ini berfungsi untuk menentukan nilai probabilitas semut. Berikut merupakan *source code* menghitung *nilai probabilitas* dapat dilihat pada *Source Code* 9.

```
1.     double[] hitungProbabilitasRelatif(double[]
2.   _fitness_Populasi) {
3.         double[] probabilitasRelatif = new
4.   double[_fitness_Populasi.length];
```

```
5.          double jumlah = 0;
6.          System.out.println("");
7.          System.out.println("|| Probabilitas Relatif ||");
8.          for (int i = 0; i < fitnessPopulasi.length; i++) {
9.              jumlah = jumlah + fitnessPopulasi[i];
10.         }
11.         for (int i = 0; i < fitnessPopulasi.length; i++) {
12.             probabilitasRelatif[i] = fitnessPopulasi[i] /
13. jumlah;
14.             System.out.println("Populasi " + (i + 1) +
15. "\t" + probabilitasRelatif[i]);
16.         }
17.         System.out.println("");
18.         return probabilitasRelatif;
19. }
```

***Source Code* 9. Menghitung nilai probabilitas**

Untuk proses dari nilai probabilitas ini menggunakan nama *method* hitungProbabilitasRelatif, dimana hasil dari proses ini tergantung dari nilai *fitness* terbesar. Baris ke- 3 merupakan proses inisialisasi parameter probabilitas relatif dengan data dari nilai *fitness*, pada baris ke 8 sampai 10 adalah proses untuk menjumlahkan nilai *fitness* semut lalu dihitung rata-rata, sedangkan pada baris ke 11 sampai 16 adalah proses untuk menghitung nilai rata-rata.

5.2.5 Menghitung *Update Pheromone*

Source code 10 adalah proses menghitung *update pheromone* menggunakan enam masukan yaitu inisialisasi *pheromone* awal, nilai probabilitas, nilai *fitness* semut, jumlah semut, koefisien penguapan, kecepatan *pheromone*. *Method* ini berfungsi untuk menentukan nilai *pheromone* setiap semut. Berikut merupakan untuk menghitung *update pheromone* dapat dilihat pada Source Code 10.

```
1.  double[] updatePheromone(double[] pheromone, double[]
2.  probabilitasRelatif, double[] fitnessSemut, int jmlSemut,
3.  double koefisienPenguapan, int kecepatanPheromone) {
4.          double probabilitasTerbaik =
5.  maxValue(probabilitasRelatif);
6.          double fitnessTerbaik = minValue(fitnessSemut);
7.          System.out.println("");
8.          System.out.println("|| Update Pheromone ||");
9.          for (int i = 0; i < jmlSemut; i++) {
10.             if (probabilitasTerbaik ==
11. probabilitasRelatif[i]) {
12.                 pheromone[i] = koefisienPenguapan *
13. pheromone[i] + kecepatanPheromone / fitnessTerbaik;
14.             } else {
15.                 pheromone[i] = koefisienPenguapan *
16. pheromone[i];
17.             }
18.             System.out.println("Pheromone-" + (i + 1) +
19. "\t" + pheromone[i]);
20.         }
21.         return pheromone;}
```

Source Code* 10. Menghitung *update pheromone

Proses untuk menghitung *update phromone* ini menggunakan nama *method* update*Pheromone*, pada proses *update pheromone* ini menggunakan nilai *fitness* terkecil dan nilai probabilitas terbesar. Baris ke 4 dan 6 merupakan proses inisialisasi parameter untuk nilai probabilitas terbaik yaitu nilai probabilitas terbesar dan nilai *fitness* terbaik adalah nilai *fitness* minimum, sedangkan untuk proses perhitungan *update pheromone* ditunjukan pada baris ke 9 sampai 17. Pada baris ke 12 merupakan proses untuk melakukan perhitungan jika nilai probabilitasnya terbesar, sedangkan jika nilai probabilitas terkecil akan diproses pada baris ke 15.

5.2.6 Menghitung *Update* Posisi

Source code 11 adalah proses untuk menghitung *update* posisi menggunakan 4 masukan yaitu letak semut, standar deviasi, posisi terbaik, dan jumlah *cluster*. *Method* ini digunakan untuk memperbarui posisi semut untuk iterasi selanjutnya. Proses *update* posisi semut terjadi jika nilai *pheromone* terbaik tidak terletak pada tempat yang sama sebanyak limit yang diinputkan oleh pengguna. Proses *update* posisi yang belum konvergen dihitung menggunakan Persamaan 2.6. Pada *update* posisi ini ada suatu proses, yaitu menghitung *mean*.

Berikut merupakan *source code* untuk menghitung *update* posisi yang ditunjukan pada *Source Code* 11.

```
1.  double[][][] updatePosisiSemut(double[][][] semut, double
2.  standarDeviasi, int posisiTerbaik, int jmlCluster) {
3.          System.out.println("");
4.          System.out.println("|| Update Posisi Semut ||");
5.
6.          double[][] pusatCluster = new
7.  double[semut[0].length][semut[0][0].length];
8.          for (int i = 0; i < pusatCluster.length; i++) {
                for (int j = 0; j < pusatCluster[0].length;
9.  j++) {
10.                 pusatCluster[i][j] =
11. semut[posisiTerbaik][i][j];
12.             }
13.         }
14.
15.         double[][] jarakData =
16. hitungJarakData(pusatCluster, dataLatih);
17.         int[] posisiCluster = tentukanCluster(jarakData,
18. jmlCluster);
19.         double[] jmlAnggotaCluster =
20. hitungJumlahAnggotaCluster(posisiCluster, jmlCluster);
21.
22.         double[][] mean = hitungMean(dataLatih,
23. posisiCluster, jmlAnggotaCluster);
24.
25.         for (int i = 0; i < semut.length; i++) {
26.             if (i != posisiTerbaik) {
27.                 Random r = new Random();
28.                 System.out.print("Semut-" + (i + 1) +
29. "\t");
30.                 for (int j = 0; j < semut[0].length; j++)
31. {
```

```
32.                     for (int k = 0; k <
33. semut[0][0].length; k++) {
34.                         semut[i][j][k] = r.nextGaussian()
35. * standarDeviasi + mean[j][k];
36.                         System.out.print(semut[i][j][k] +
37. "\t");
38.                     }
39.                 }
40.                 System.out.println("");
41.             } else {
42.                 System.out.print("Semut-" + (i + 1) +
43. "\t");
44.                 for (int j = 0; j < semut[0].length; j++)
45. {
46.                     for (int k = 0; k <
47. semut[0][0].length; k++) {
48.                         System.out.print(semut[i][j][k] +
49. "\t");
50.                     }
51.                 }
52.                 System.out.println("");
53.             }
54.         }
55.
56.         return semut;
57.     }
58.
```

***Source Code* 11. Menghitung *update* posisi**

Pada proses *update* posisi baris ke 6 sampai 11 adalah proses untuk memanggil pusat *cluster* pada posisi terbaik. Dalam melakukan proses *update* posisi yaitu dengan menggunakan beberapa method sebelumnya. Pada baris ke 15 memanggil *method* hitungJarakData untuk menghitung jarak data ke titik pusat. Pada baris ke 17 memanggil *method* tentukanCluster untuk menentukan posisi *cluster* data. Pada baris ke 19 adalah proses memanggil *method* jmlAnggotaCluster untuk menjumlah data di dalam *cluster* yang akan digunakan sebagai inputan proses *mean*. Baris ke 21 untuk memanggil *method* hitungMean yaitu proses menentukan hasil mean, sedangkan pada perulangan baris ke 24 sampai 38 adalah proses dalam menghitung *update* posisi. Dalam proses *update* posisi jika semut tidak dalam posisi terbaiknya maka akan dihitung *update* posisi yang diproses pada baris ke 33, sedangkan jika posisi semut sudah terbaik maka posisinya tidak akan berubah seperti yang ditunjukan pada baris ke 41 sampai 48.

5.2.6.1 Menghitung *Mean*

Source Code menghitung *mean* menggunakan masukan yaitu data pada biji wijen, posisi *cluster*, dan jumlah anggota *cluster*. Pada proses menghitung mean ini digunakan untuk proses *update* posisi. Berikut dibawah ini adalah *source code* untuk menghitung *mean* yang ditunjukan pada *Source Code* 12.

```
1.  double[][] hitungMean(double[][] data, int[] posCluster,
2.  double[] jmlAnggotaCluster) {
```

```
3.          double[][] mean = new
4.   double[jmlAnggotaCluster.length][data[0].length];
5.          for (int i = 0; i < jmlAnggotaCluster.length; i++)
6.   {
7.              for (int j = 0; j < data.length; j++) {
8.                  if (posCluster[j] == (i)) {
9.                      for (int k = 0; k < data[0].length;
10.  k++) {
11.                         mean[i][k] = mean[i][k] +
12.  data[j][k];
13.                     }
14.                 }
15.             }
16.         }
17.         System.out.println("");
18.         System.out.println("|| Mean ||");
19.         for (int i = 0; i < jmlAnggotaCluster.length; i++)
20.  {
21.             System.out.print("Cluster " + (i + 1) + "\t");
22.             for (int j = 0; j < data[i].length; j++) {
23.                 mean[i][j] = mean[i][j] /
24.  jmlAnggotaCluster[i];
25.                 if (Double.isNaN(mean[i][j])) {
26.                     mean[i][j] = 0.0;
27.                 }
28.                 System.out.print(mean[i][j] + "\t");
29.             }
30.             System.out.println("");
31.         }
32.         return mean;
33.     }
```

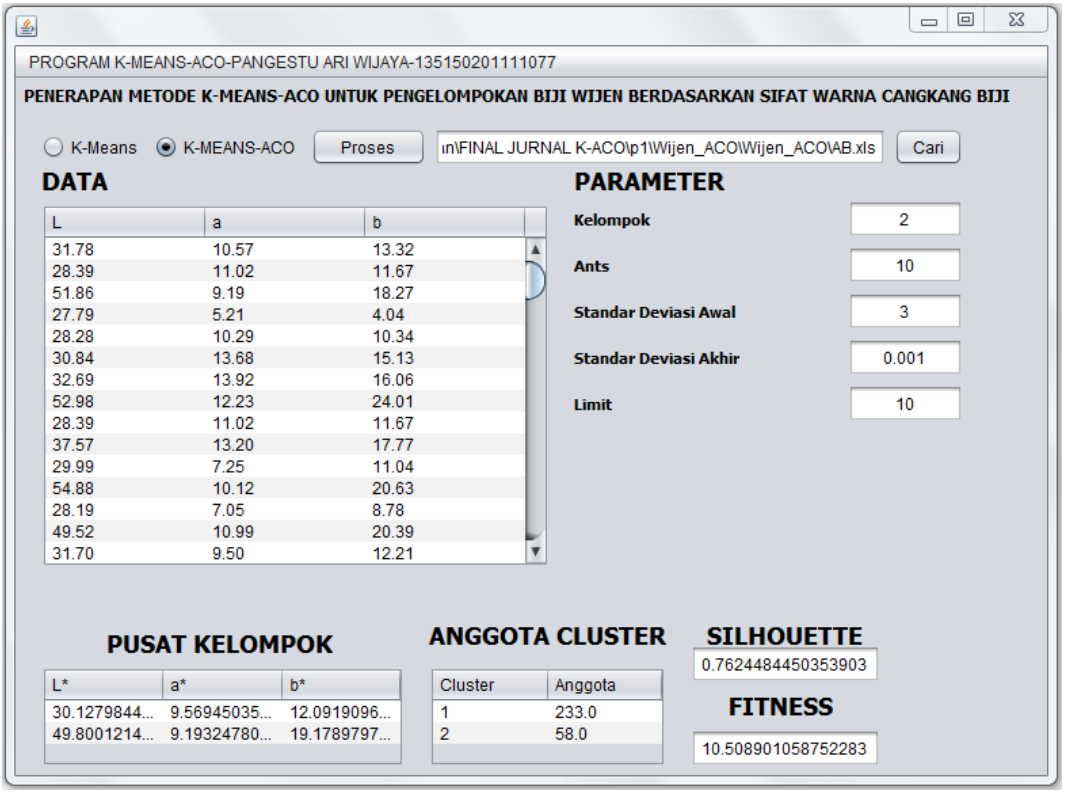
Source Code* 12. Menghitung *mean

Proses inisalisasi parameter untuk menghitung mean menggunakan inputan jumlah anggota *cluster* ditunjukan pada baris ke 3. Pada proses baris 5 sampai 12 merupakan proses untuk menghitung jumlah mean pada setiap *cluster*, lalu pada baris ke 19 sampai 29 dilakukan proses perhitungan nilai mean. Nilai mean diperoleh dari jumlah keseluruhan nilai mean pada setiap *cluster* dibagi jumlah anggota pada *cluster* tersebut yang ditunjukan dibaris 23.

5.3 Implementasi UI

Pada tahap ini implementasi *User Interface* atau UI hanya terdiri dari 1 halaman utama saja untuk melakukan *input* data, memilih metode untuk pengelompokan, dan menampilkan hasil dari pengelompokan. Untuk langkah pertama, pengguna akan meng*input*kan data latih biji wijen dengan format excel.xls yang berisikan data format warna L*,a*,b*, dan tahapan selanjutnya adalah memilih metode yang akan digunakan, yaitu *K-Means* dan *K-Means*-ACO.

Hasil dari proses pengelompokan tersebut dapat diketahui yaitu hasil dari pusat kelompok, jumlah anggota *cluster*, nilai *fitness* dan yang terakhir adalah nilai *silhoutte coeffisient* pada setiap kelompok di akhir proses. Berikut merupakan hasil dari tampilan *user interface* pada program pengelompokan biji wijen menggunakan metode *K-Means*-ACO yang ditunjukan pada Gambar 5.2.



Gambar 5.2 Hasil program *user interface* metode *K-Means*-ACO