BAB 5 IMPLEMENTASI

Pada bab implementasi ini akan membahas tentang implementasi sistem pengiriman baran J&T *Express* Surabaya berdasarkan proses perancangan yang telah dibangun sebelumnya. Pembahasan dalam tahap ini meliputi siklus dari algoritme genetika, mulai dari pembetukan populasi awal, reproduksi, evaluasi sampai seleksi serta membahas hasil implementasi antarmuka yang telah tersedia.

5.1 Implementasi Program

Implementasi program merupakan proses implementasian MTSPTW pada algoritme genetika kedalam kode-kode tertentu. Sehingga bisa dimanfaatkan oleh user untuk menyelesaikan permasalahan pengiriman baran J&T *Express.* Pada sub bab ini akan menunjukan beberapa *source code* beberapa fungsi yang ada pada program. Seperti yang ditunjukan pada Gambar 4.1 algoritme genetika memiliki beberapa proses penting yang harus diimplementasikan ke dalam program.

5.1.1 Inisialisasi Populasi Awal

Inisialisasi Populasi awal merupakan proses awal yang harus dikerjakan oleh program, untuk membentuk beberapa kromosom sebanyak popsize yang telah ditentukan diawal. Proses ini sendiri memanfaatkan fungsi random yang ada pada java dengan menambahkan beberapa kondisi *(if)* agar nilai random yang akan dimasukan kedalam kromosom tidak mengalami pengulangan.

No.	Kode Program
1	`for (int i = 0; i < ps; i++) {`
2	`   random = new int[50];`
3	`   for (int j = 0; j < 50; j++) {`
4	`      tmp = rand.nextInt(j + 1);`
5	`      random[j] = random[tmp];`
6	`      random[tmp] = j + 1;`
7	`   }`
8	`for (int l = 0; l < 50; l++) {`
9	`   kromosom[i][l] = random[l];`
10	`}`
11	`tmp = 0;`
12	`for (int j = 0; j < bSales; j++) {`
13	`  do {`
14	`    sSementara = rand.nextInt(((int) (50 / bSales) + 3));`
15	`  } while (sSementara < ((int) (50 / bSales) - 1));`
16	`  if (j < bSales - 1) {`
17	`     sales[i][j] = sSementara;`
18	`     tmp += sales[i][j];`
19	`  } else {`
20	`     sales[i][j] = 50 - tmp;`
21	`  }`
22	`}`

Gambar 5.1 Source Code Inisialisasi Populasi Awal

Tabel 5.1 Penjelasan Source Code *Crossover*

No.	Kode Program
1	Melakukan sebuah perulangan berdasarkan jumlah *popsize* yang ada
2	Inisialisasi variabel
3-7	Perulangan untuk melakukan random angka
4-6	Pemberian serta penukaran nilai untuk variabel random
8-10	Perulangan pemberian nilai ke variabel kromosom dari hasil random sebelumnya
11	Inisialisasi nilai tmp
12	Melakukan sebuah perulangan berdasarkan jumlah *sales* yang ada
13-15	Perulangan do while yang digunakan untuk menentukan banyaknya tempat yang dikunjungi *sales*
15	Kondisi agar jumlah tempat yang dikunjungi *sales* tidak terlalu sedikit
16-18	Kondisi untuk memberikan nilai ketika indeks kurang dari banyak *sales*
19-21	Kondisi untuk memberikan nilai ketika indeks sama dengan banyak *sales*

5.1.2 Reproduksi

Reproduksi merupakan proses pembentukan kromosom baru dari satu atau beberapa parent yang terpilih secara random yang berasal dari kromosom pada populasi awal atau sebelumnya. Proses reproduksi sendiri ada dua macam yaitu *crossover* dan *mutation*. Pada sistem yang dibuat kali ini, menggunakan 2 jenis *crossover,* yaitu *one cut point crossover* dan *two cut point crossover*, dan untuk *mutation* juga menggunakan 2 jenis yaitu *exchange* dan *insert*.

5.1.2.1 Crossover

Crossover merupakan salah satu proses pembentukan kromosom baru dengan memilih lebih dari satu kromosom yang sudah ada sebelumnya. Kromosom terpilih biasa disebut sebagai parent dan kromosom baru yang terbentuk biasa disebut child. Untuk membentuk sebuah kromosom baru, dibutuhkan minimal dua parent yang akan digabungkan dengan batasan berupan titik potong, dan pengecekan apakan gen yang akan dimasukan kedalam child sudah dimasukan sebelumnya agar menghindari terjadinya pengulangan gen.

No.	Kode Program
1	`for (int h = 0; h < offspring; h++) {`
2	`  for (int i = 0; i < 2; i++) {`
3	`    if (i % 2 == 0) {`
4	`      for (int j = 0; j < titik; j++) {`
5	`        anakC[h][j] = kromosom[parent[h][i]][j];`
6	`       }`
7	`      indeks = titik;`
8	`    } else {`
9	`       for (int j = 0; j < pData; j++) {`
10	`        cek = false;`
11	`        for (int k = 0; k < titik; k++) {`
12	`       if (anakC[h][k] == kromosom[parent[h][i]][j]) {`

```
13            cek = true;
14          }
15        }
16        if (cek == false) {
17          anakC[h][indeks] = kromosom[parent[h][i]][j];
18          indeks++;
19        }
20        if (indeks == pData) {
21          break;
22        }
23    }
24  }
25  for (int l = 0; l < 3; l++) {
26    salesC[h][l] = sales[parent[h][0]][l];
27  }
```

Gambar 5.2 Source Code *Crossover*

Tabel 5.2 Penjelasan Source Code *Crossover*

No.	Kode Program
1	Melakukan sebuah perulangan berdasarkan jumlah *offspring* / anak yang ada
2	Melakukan perulang sebanyak 2 kali, digunakan untuk mengetahui posisi parent
3	Kondisi yang digunakan untuk mengetahui posisi parent ada di parent pertama
4-6	Melakukan perulangan sampai titik potong pertama, serta pemberian nilai untuk variabel anak atau *offspring*
7	Pemberian nilai untuk variabel indeks
8-9	Kondisi yang digunakan untuk mengetahui posisi indeks di parent ke dua
10	Inisialisasi nilai cek
11	Perulangan untuk mengecek apakah individu pada indeks j sudah diinputkan atau belum
12-19	Kondisi ketika individu belum terinput dan pemberian nilai untuk *offspring* / anak
20-22	Kondisi untuk menghentikan perulangan ketika indeks sudah mencapai batas max individu
25-27	Pemberian nilai untuk *sales* pada *sales offspring* / anak

5.1.2.2 Mutation

Mutation merupakan salah satu proses pembentukan kromosom baru dengan memilih satu kromosom yang sudah ada sebelumnya untuk membentuk kromosom baru. Sama seperti halnya *crossover* pada *mutation* kromosom terpilih biasa disebut sebagai parent dan kromosom baru yang terbentuk biasa disebut child. Untuk membentuk sebuah kromosom baru, beberapa gen pada parent akan berubah posisi, misalnya pergantian posisi gen (*exchange*) dan pergeseran gen (insert).

No.	Kode Program
1	`for (int i = 0; i < offspringM; i++) {`
2	`  parent = random((int) ps - 1);`
3	`  titik1 = random((int) pData / 3);`
4	`  do {`
5	`  titik2 = random(pData - 1);`
6	`  } while (titik2 <= titik1);`
7	`  for (int j = 0; j < (int) b`*`Sales`*`; j++) {`
8	`  `*`sales`*`M[i][j] = `*`sales`*`[parent][j];`
9	`  }`
10	`  for (int j = 0; j < pData; j++) {`
11	`  if (j < titik1 || j > titik2) {`
12	`  anakM[i][j] = kromosom[parent][j];`
13	`  } else if (j == titik1) {`
14	`  anakM[i][j] = kromosom[parent][titik2];`
15	`  } else {`
16	`  anakM[i][j] = kromosom[parent][j - 1];`
17	`  }`
18	`  }`
19	`}`

Gambar 5.3 Source Code *Insertion mutation*

Tabel 5.3 Penjelasan Source Code *Mutation*

No.	Kode Program
1	Perulangan yang dilakukan sebanyak jumlah Offspring
2-3	Pemberian nilai random untuk memilih parent dan juga titik tukar untuk *mutation*
4-6	Perulangan do while yang dilakukan untuk menentukan nilai titik tukar *mutation* yang ke dua agar tidak sama dengan titik pertama
7-8	Perulangan untuk memberikan nilai pada *sales*
10-18	Perulangan untuk memberikan nilai pada kromosom child
11-12	Kondisi ketika nilai j lebih dari titik pertam dan kurang dari titik kedua
13-14	Kondisi ketika nilai J sama dengan titik 1
15-17	Kondisi ketikatidak memenuhi kondisi sebelumnya

5.1.3 Evaluasi

Evaluasi merupakan sebuah proses yang lebih dikenal dengan perhitungan nilai *fitness* yang dimili oleh setiap kromosom yang ada termasuk kromosom child. Pada sistem ini, untuk melakukan perhitungan *fitness* kromosom pada populasi awal akan digabung terlebih dahulu dengan *child* yang telah terbentuk dalam proses reproduksi sebelumnya. Proses perhitungan proses pada sistem ini menggunakan rumus 4.3 dan rumus 4.4yang digunakan untuk menentukan nilai *fitness* dan cost dari proses perhitungan dengan menggunakan tingkat kemacetan sebagai pinalty. Data yang digunakan pada proses ini adalah data waktu tempuh pada *Multiple Travelling Salesman Problem Time Window* atau jarak tempuh pada *Multiple Travelling Salesman Problem* dan juga tingkat kemacetan yang digunakan sebagai *pinalty* yang bisa mempengaruhi nilai dari kriteria pada setiap kromosom.

No.	Kode Program

```
for (int i = 0; i < banyakPopsize; i++) {
  int indeks Sales = 0;
  for (int j = 0; j < (int) b Sales; j++) {
    for (int k = 0; k <= sales Gabung[i][j]; k++) {
      if (k == 0) {
       total[i][j] += tingkatMacet[0][(kromosomGabung[i]
       [k+ indeks Sales])];
      } else if (k == sales Gabung[i][j]) {
       total[i][j] += tingkatMacet[0][(kromosomGabung[i]
       [k - 1 + indeks Sales])];
      } else {
       total[i][j] += tingkatMacet[(kromosomGabung[i][k+
       indeks Sales])][(kromosomGabung[i][k + indeks Sales -
       1])];
      }
    }
    indeks Sales += sales Gabung[i][j];
  }
}
for (int i = 0; i < banyakPopsize; i++) {
  indeksBobot = 0;
  int indeks Sales = 0;
  for (int j = 0; j < (int) b Sales; j++) {
    for (int k = 0; k <= sales Gabung[i][j]; k++) {
      if (k == 0) {
       waktuPinalty[i][indeksBobot] = 1 + (tingkatMacet[0]
       [(kromosomGabung[i][0 + indeks Sales])]/total[i][j]);
      } else if (k == sales Gabung[i][j]) {
       waktuPinalty[i][indeksBobot] = 1 +(tingkatMacet[0]
       [(kromosomGabung[i] [k - 1 + indeks Sales])]
        /total[i][j]);
      } else {
        waktuPinalty[i][indeksBobot]=1+(tingkatMacet
        [(kromosomGabung[i][k + indeks Sales])]
        [(kromosomGabung[i][k + indeks Sales - 1])]
        / total[i][j]);
      }
      indeksBobot++;
    }
    indeks Sales += sales Gabung[i][j];
  }
}
for (int i = 0; i < banyakPopsize; i++) {
  indeksBobot = 0;
  int indeks Sales = 0;
  for (int j = 0; j < (int) b Sales; j++) {
    for (int k = 0; k <= sales Gabung[i][j]; k++) {
     if (k == 0) {
       cost[i][j] += dataTmp[0][(kromosomGabung[i]
       [0+indeks Sales])] * waktuPinalty[i][indeksBobot];
     } else if (k == sales Gabung[i][j]) {
        cost[i][j] += dataTmp[0][(kromosomGabung[i][k - 1 +
         indeks Sales])] * waktuPinalty[i][indeksBobot];
      } else {
         cost[i][j] += dataTmp[(kromosomGabung[i][k +
         indeks Sales])][(kromosomGabung[i][k + indeks Sales
         - 1])] * waktuPinalty[i][indeksBobot];
```

```
58          }
59          indeksBobot++;
60        }
61      indeksSales += salesGabung[i][j];
62    }
63  }
64
```

Gambar 5.4 Source Code pencarian nilai *Cost*

Tabel 5.4 Penjelasan Source Code pemberian nilai tingkat kemacetan

No.	Kode Program
1-18	Perulangan untuk menghitung total dari nilai tingkat kemacetan pada masing-masing *sales* disetiap popsize
2	Inisialisasi indeks*Sales*
3	Perulangan untuk setiap seles
4	Perulangan sebanyak jumlah tempat yang di tuju seales
5-7	Kondisi untuk pemberian nilai total ketika indeks k = 0
8-10	Kondisi untuk pemberian nilai total ketika indeks k = jumlah pengiriman *sales*
11-15	Kondisi untuk pemberian nilai total ketika indeks lebih lebih besar dari 0 dan lebih kecil dari jumlah pengiriman tiap *sales*
16	Pemberian nilai indeks*Sales*
19-42	Perulangan untuk menghitung total dari Waktu *pinalty* tiap tempat pada masing-masing *sales* disetiap popsize
20-21	Inisialisasi indeks*Sales* dan indeksBobot
22	Perulangan untuk setiap seles
23	Perulangan sebanyak jumlah tempat yang di tuju seales
24-27	Kondisi untuk pemberian nilai Waktu *pinalty* ketika indeks k = 0
28-31	Kondisi untuk pemberian nilai Waktu *pinalty* ketika indeks k = jumlah pengiriman *sales*
32-37	Kondisi untuk pemberian nilai Waktu *pinalty* ketika indeks lebih lebih besar dari 0 dan lebih kecil dari jumlah pengiriman tiap *sales*
38	Pemberian nilai indeksBobot
40	Pemberian nilai indeks*Sales*
43-64	Perulangan untuk menghitung total dari *cost* pada masing-masing *sales* disetiap popsize
44-45	Inisialisasi indeks*Sales*, indeksBobot
46	Perulangan untuk setiap seles
47	Perulangan sebanyak jumlah tempat yang di tuju seales
48-50	Kondisi untuk pemberian nilai *cost* ketika indeks k = 0
51-53	Kondisi untuk pemberian nilai *cost* ketika indeks k = jumlah pengiriman *sales*
54-57	Kondisi untuk pemberian nilai *cost* ketika indeks lebih lebih besar dari 0 dan lebih kecil dari jumlah pengiriman tiap *sales*
59	Pemberian nilai indeksBobot
61	Pemberian nilai indeks*Sales*

No.	Kode Program
1	`for (int i = 0; i < banyakPopsize; i++) {`
2	`  for (int j = 0; j < (int)b`*`Sales`*`; j++) {`
3	`    `*`fitness`*`[i] += (600 - cost[i][j]);`
4	`  }`
5	`}`

Gambar 5.5 Source Code nilai *fitness*

Tabel 5.5 Penjelasan Source Code nilai *fitness*

No.	Kode Program
1	Perulangan sebanyak nilai *popsize*
2	Perulanya sebanyak *sales* yang ada
3	Pemberian nilai *fitness* dengan melakukan pengurangan nilai konstanta dengan nilai *cost*

5.1.4 Seleksi

Seleksi adalah proses untuk mempertahankan kromosom yang memiliki nilai optimum, dan menghapus kromosom yang kurang optimum. Proses seleksi sendiri dilakukan dengan menggunakan metode *Elitism*. Metode tersebut dilakukan dengan mengurutkan nilai fitnees yang optimum ke yang tidak optimum. Proses pengurutan sendiri dilakukan dengan metode sorting *Bublesort* yaitu dengan membandingkan nilai satu dengan yang lainnya kemudian nilai yang lebih optimum akan diletakan pada posisi teratas dan seterusnya.

No.	Kode Program
1	`for (int i = 0; i < `*`fitness`*`.length; i++) {`
2	`  for (int j = i; j < `*`fitness`*`.length; j++) {`
3	`    if (`*`fitness`*`[j] < `*`fitness`*`[i]) {`
4	`      tmp = `*`fitness`*`[i];`
5	`        `*`fitness`*`[i] = `*`fitness`*`[j];`
6	`        `*`fitness`*`[j] = tmp;`
7	`        for (int k = 0; k < 50; k++) {`
8	`          int tmp1 = kromosomGabung[i][k];`
9	`          kromosomGabung[i][k] = kromosomGabung[j][k];`
10	`          kromosomGabung[j][k] = tmp1;`
11	`        }`
12	`        for (int k = 0; k < 3; k++) {`
13	`          int tmp1 = `*`sales`*`Gabung[i][k];`
14	`          `*`sales`*`Gabung[i][k] = `*`sales`*`Gabung[j][k];`
15	`          `*`sales`*`Gabung[j][k] = tmp1;`
16	`        }`
17	`    }`
18	`  }`
19	`}`

Gambar 5.6 Source Code Seleksi

Tabel 5.6 Penjelasan Source Code Seleksi

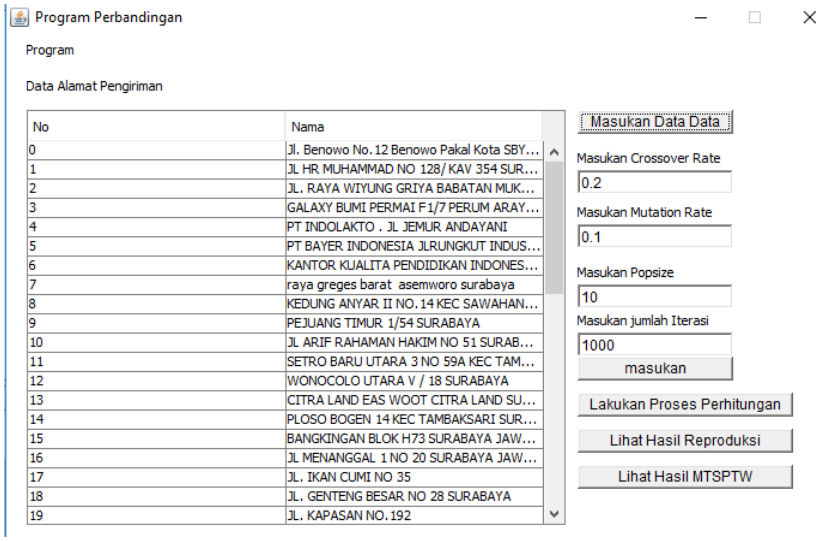
No.	Kode Program
1	Melakukan sebuah perulangan berdasarkan jumlah *fitness* yang ada
2	Melakukan sebuah perulangan berdasarkan jumlah *fitness yang ada* untuk membandingkan nilai *fitness* i dengan *fitness* selanjutnya
3	Kondisi ketika nilai *fitness* i lebih kecil dari *fitness* j
4-6	Proses pembalikan antara nilai *fitness* i dengan *fitness* j
7-11	Perulangan untuk m*update* kromosom
12-16	Perulangan untuk update *sales*

5.2 Implementasi Antarmuka

Implementasi antarmuka sistem penentu Algoritme tercepat antara *Multiple Travelling Salesman Problem* dan *Multiple Travelling Salesman Problem Time Wndow* nantinya akan berguna untuk mempermudah melihat hasil dari perhitungan yang dilakukan oleh sistem. Selain itu antarmuka juga membuat sistem yang dibuat lebih fleksibel atau yang dimaksud fleksibel adalah mengganti beberapa variabel yang perlu dimasukan oleh pengguna (nilai *crossover rate, mutation rate,*dan iterasi).

5.2.1 Implementasi Antarmuka Halaman Awal

Halaman awal, merupakan halaman awal ketika user membuka program. Halaman ini menampilkan data alamat yang harus dituju oleh *sales* yang ada. Halaman ini juga berguna untuk memasukan nilai *crossover rate, mutation rate, popsize* dan iterasi. Nilai *crossover rate* dan *mutation rate* diisikan dengan variabel *double* karena nilai yang dibutuhkan untuk cr dan mr tersebut bernilai antara 0-1. Sedangkan untuk nilai *popsize* dan iterasi diiisikan dengan variabel *integer*, karena nilai yang dibutuhkan adalah berapa banyak kromosom yang akan dibentuk pada proses pembentukan populasi awal serta berapa banyak iterasi yang dilakukan. Tampilan Antarmuka halaman awal dapat dilihat pada Gambar 5.7.

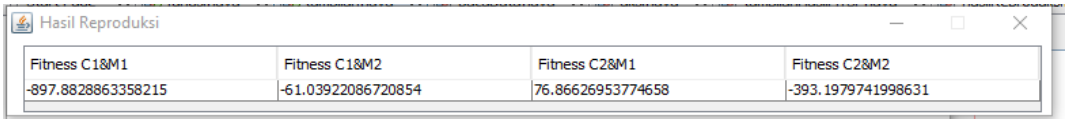


Gambar 5.7 Halaman Awal

5.2.2 Implementasi Antarmuka Halaman Hasil

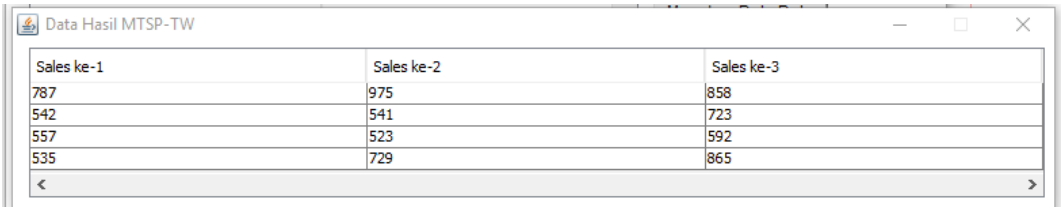
Halaman Hasil adalah halaman yang menunjukan hasil atau nilai terakhir yang dilakukan (nilai akhir iterasi). Halaman Hasil sendiri terbagi atas 3 bagian yaitu:

1. Hasil Reproduksi
2. Hasil *Multiple Traveling Salesman Problem Time Window*
3. Hasil *Multiple Traveling Salesman Problem*



Gambar 5.8 Halaman Hasil Reproduksi

Untuk halaman hasil reproduksi yang terlihat pada Gambar 5.8, memperlihatkan hasil dari kombinasi *crossover* dan *mutation*, dari hasil iterasi terakhir. Pada kolom pertama memperlihatkan hasil dari kombinari *one cut point crossover* dengan *insertion mutation,* pada kolom kedua memperlihatkan hasil dari kombinari *one cut point crossover* dengan *exchange mutation,* pada kolom ketiga memperlihatkan hasil dari kombinari *two cut point crossover* dengan *insertion mutation,* pada kolom keempat memperlihatkan hasil dari kombinari *two cut point crossover* dengan *exchange mutation*.



Gambar 5.9 Halaman Hasil *Multiple Traveling Salesman Problem Time WIndow*

Untuk halaman hasil *Multiple Traveling Salesman Problem Time Window* yang terlihat pada Gambar 5.9, memperlihatkan hasil dari waktu tempuh untuk setiap *sales*. Waktu tempuh tersebut berdasarkan kromosom terakhir yang terpilih dari proses algoritme genetika. Selain itu pada tampilan hasil *Multiple Traveling Salesman Problem Time Window* juga memperlihatkan hasil tiap kombinasi yang dilakukan. Pada baris pertama memperlihatkan kombinari *one cut point crossover* dengan *insertion mutation,* pada baris kedua memperlihatkan hasil dari kombinari *one cut point crossover* dengan *exchange mutation,* pada baris ketiga memperlihatkan hasil dari kombinari *two cut point crossover* dengan *insertion mutation,* pada baris keempat memperlihatkan hasil dari kombinari *two cut point crossover* dengan *exchange mutation*.